3D MODELING OF ROAD INFRASTRUCTURES ACCORDING TO CITYGML 3.0 AND ITS CITYJSON ENCODING

A. Yarroudh[1,*], G.-A. Nys[1], R. Hajji[2]

[1] Geomatics Unit, University of Liège, Allée du six Août 19, 4000 Liège, Belgium - (ayarroudh, ganys)@uliege.be
[2] College of Geomatic Sciences and Surveying Engineering, Institute of Agronomy and Veterinary Medicine, BP 6202 Rabat, Morocco - r.hajji@iav.ac.ma

Commission I, WG I/2

KEY WORDS: 3D City Model, Road infrastructure, CityGML 3.0, CityJSON, MongoDB.

ABSTRACT:

This study proposes a systematic approach for standardized 3D modeling of road infrastructures based on CityGML 3.0 and its CityJSON encoding. The approach involves generating a LoD2 3D model of the road infrastructure based on a semi-automatic extraction of linear features from mobile mapping LiDAR data. This enables geometric and semantic modeling of the roads. A codification system is proposed to assign predefined codes to each linear feature, allowing each section and intersection and each road surface to be modeled separately. The resulting model is converted to a CityJSON file, stored in a document-oriented database and visualized through a web application. The proposed approach provides a cost-effective alternative to traditional manual modeling methods while maintaining a high level of accuracy and consistency. It also considers the validation of data schema and geometric primitives to ensure that any non-conformity with CityJSON schemas, and any topological and/or geometric errors can be detected and then corrected. This is important since schema changes in new versions of CityJSON can result in compatibility issues, while geometric and topological errors can affect the accuracy of 3D models and ultimately lead to inaccurate simulation outcomes or analysis results.

1. INTRODUCTION

3D city models are digital representations of urban areas and landscapes in a 3D format featuring, among others, buildings, roads, vegetation, water and other city objects. So far, most researches have focused on 3D building models. This is partly due to their prominent role in the urban tissue but also because of insufficient information and data feeds for other thematic topics (Beil and Kolbe, 2017).

In recent years, with the increasing need to address urban planning and sustainable city management, the 3D modeling of other city objects types such as roads becomes a necessity. Indeed, detailed road models are essential for a range of applications, including navigation, autonomous driving and urban planning (Zhang et al., 2019). Furthermore, information provided on roadways is increasingly available which makes it possible to reconstruct the 3D geometry of the road infrastructure.

Considering that most current standards primarily focus on a linear representation of road networks, our study aims to develop an approach for standardized 3D modeling of road infrastructures based on CityGML 3.0 (Kutzner et al., 2020) and its CityJSON encoding (Ledoux et al., 2019). The proposed approach involves developing a systematic process for producing 3D road models that are geometrically consistent, efficiently structured and stored, and accessible for online visualization and inspection through a web application.

To briefly summarize, the main contributions of this work are twofold:

- A systematic approach for creating geometrically consistent 3D road models in CityJSON format.
- An optimized method for structuring, storing and visualizing of these models.

The reminder of the paper is organized as follows: Section 2 gives an overview about existing road modeling standards. Section 3 provides a detailed description of the proposed methodology. We presents and discusses the results in section 4 and conclude the paper in section 5.

2. RELATED WORKS

Different standards have addressed 3D modeling of road network based either on linear or surface representations. Among them, we can cite Geographic Data Files (GDF), OpenDrive, LandInfra, RoadXML, ASB & OKSTRA and CityGML (Boersma, 2019, Beil and Kolbe, 2017).

Most of these standards focus on a linear representation of roads, with the exception of CityGML and LandInfra, whose geometry is surface-based (Boersma, 2019). While a linear representation is generally sufficient for applications such as navigation and traffic simulations, other applications may require a detailed geometric representation of the road surface. Unlike CityGML, LandInfra has no concept of Level of Detail, nor a separate class for intersections (Beil et al., 2020). Therefore, it is not possible to differentiate between multi-scale representations of 3D semantic city models.

CityGML is actually considered as the technological backbone of 3D city models. Whereas previous versions standardized a GML (Geography Markup Language) exchange format, CityGML 3.0 standardizes the underlying information model and can therefore be implemented in a variety of non-GML

[*] Corresponding author

technologies. The Transportation module is one of the thematic modules that has been revised in CityGML 3.0 to address the limitations of the previous versions.

Transportation objects are now defined as subclasses of the abstract TransportationSpace class (Kutzner et al., 2020). These objects can be subdivided into sections, which can be regular road segments, intersections and roundabouts (Figure 1).

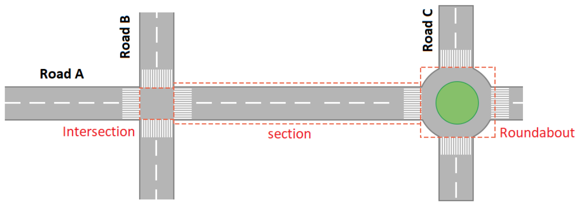


Figure 1. Illustration of road sections in CityGML.

Objects can also be subdivided into TrafficAreas, such as roads and sidewalks, and AuxiliaryTrafficAreas like green spaces. In order to adapt the semantics of the Transportation module to the CityGML 3.0 concept of space, the classes TrafficSpace and AuxiliaryTrafficSpace have been introduced, with the two areas now representing the lower boundaries of the two spaces. Each TrafficSpace can have an optional ClearanceSpace (Kutzner et al., 2020) (Figure 2).

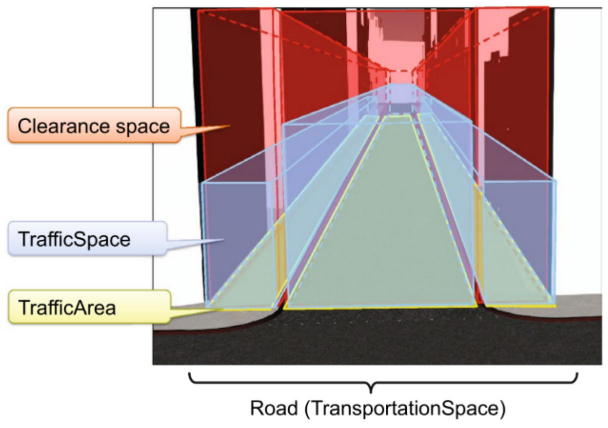


Figure 2. Space concept for Transportation objects in CityGML 3.0 (Kutzner et al., 2020).

The concept of LoD has been also redefined in version 3.0. Unlike the previous version, road objects in the Transportation module of CityGML 3.0 are modeled in 4 levels of detail: LoD0-LoD3 (Beil and Kolbe, 2017). In LoD0, roads are modeled in a linear representation. From LoD1 onwards, road surface can be modeled by linear and/or surface objects respectively. In addition to the road axis, linear representations for pedestrian zones and cycling lanes become possible in LoD2. Finally, LoD3 representations contain a TrafficSpace object for each individual traffic lane (Figure 3).

Although CityGML proposes a solid data model for structuring 3D city models, it also shows some technical problems. One of these problems is that CityGML files are very difficult to examine and extract information from.

CityGML files present problems on three levels: XML, GML and CityGML (Ledoux et al., 2019). XML (eXtensible Markup

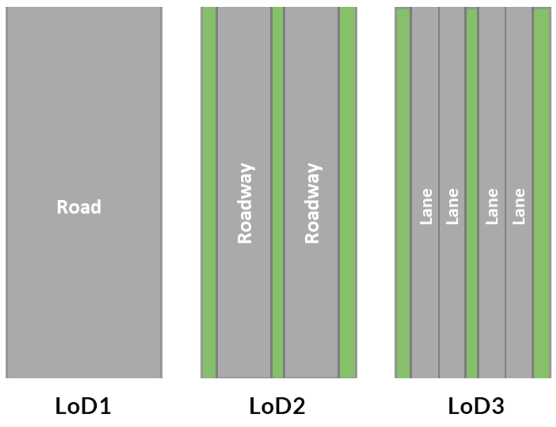


Figure 3. Surface representation of roads in LoD1-3.

Language) suffers from an excessive number of tags, reducing readability and increasing data size. GML has multiple ways to store the same geometry, requiring developers to handle various variations. CityGML inherits the advantages and disadvantages of XML and GML, including large file sizes, deep hierarchies, diverse surface storage, lack of topology storage, and the possibility of having different Coordinate Reference Systems (CRS) for different objects. Additionally, the absence of complete JavaScript parsers for CityGML complicates web-based file exchange and processing.

Therefore, to respond to these drawbacks, CityJSON was introduced as an OGC community standard on August 13, 2021. It is a conceptual model and exchange format for 3D data that implements almost all of the CityGML data model. JSON is generally preferred by developers. This preference is mainly due to the fact that JSON is much simpler than XML, since JSON is a data format while XML is a markup language, and it is therefore much easier to develop software for JSON than for XML (Ledoux et al., 2019). Another important aspect is that the CityGML data schema has been flattened and all hierarchies removed. Therefore, the city objects can be accessed directly by their identifiers.

Additionally, CityJSON files are much lighter than CityGML as compression is supported in several ways. Vertices indexing is a solution given that vertices are usually shared between several surfaces, and repeating them can be costly in terms of storage space. CityJSON files can also be compressed if vertex coordinates are transformed to integer values using a transformation matrix. The use of geometry-templates also allows CityJSON files to be further compressed, as certain objects only need to be declared once (Ledoux et al., 2019).

CityJSON is designed to be both compact and user-friendly for developers, allowing files to be effortlessly visualized, manipulated, and edited. As a result, numerous tools and APIs (Application Programming Interface) have already been created and are readily accessible. Ninja Viewer and Measur3D are two web-based applications that provide visualization and inspection capabilities for CityJSON files.

Ninja Viewer is a lightweight and interactive web viewer for CityJSON files (Vitalis et al., 2020). It uses Three.js library to create 3D scenes in a web browser, allowing users to visualize and explore the geometry and semantics of 3D city models.

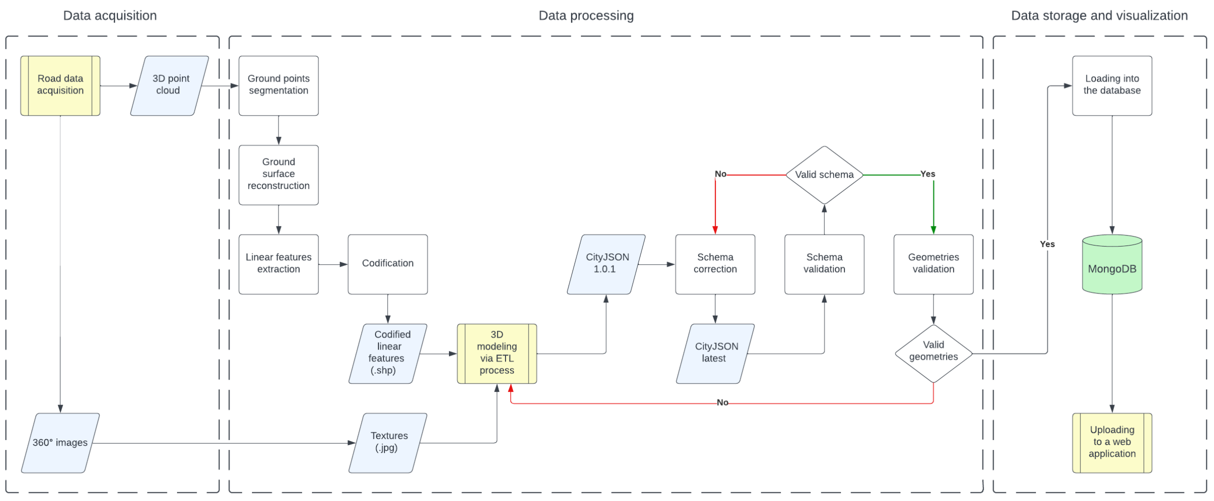
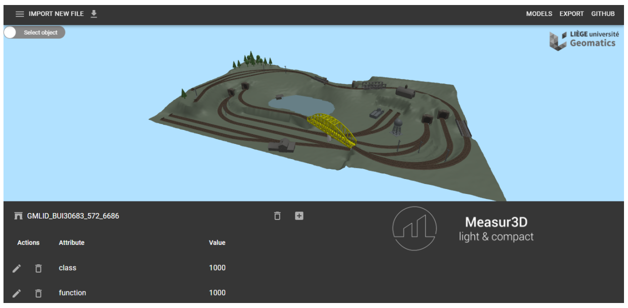


Figure 4. Our proposed method for 3D modeling, storage and visualization of road infrastructure.

However, Ninja Viewer does not provide a server-side architecture for data storage and retrieval.

Conversely, Measur3D is a more comprehensive solution that not only offers visualization but also includes functionalities for storing, managing, and retrieving CityJSON files (Nys and Billen, 2021). It is built as a MERN (MongoDB, Express, React, Node) application, which allows for the storage of CityJSON files in a MongoDB database. Measur3D leverages the capabilities of MongoDB, such as its document-oriented nature and flexible schema, to provide efficient storage and retrieval of 3D city models. Overall, Measur3D provides a complete RESTful-API for managing CityJSON files, making it a powerful tool for working with 3D city models.



GMLID_BUI30683_572_6686

Actions	Attribute	Value
	class	1000
	function	1000

Figure 5. Measur3D user interface.

3. METHOD

This study aims to reach geometrically consistent 3D road models in CityJSON format. The proposed approach involves geometric and semantic modeling of the road infrastructure based on the CityGML 3.0 data model. As illustrated in Figure 4, the method relies primarily on a semi-automatic extraction of linear features of the road, such as curbs and road boundaries, from a three-dimensional point cloud. These linear features are then used to construct various semantic surfaces, which ensures valid topology between different components of the road infrastructure.

The generated 3D road model is translated into a valid CityJSON file. This validation is performed at two levels: firstly, validation of the data schema, i.e., checking if the file syntax is correct and compliant with CityJSON specifications. Secondly, validation of the geometric primitives in accordance with the ISO 19107 standard, which specifies conceptual schemas for describing spatial characteristics of geographic features.

The output file is stored in MongoDB database using Measur3D API. Finally, the data is extracted from the database and then provided to users through a web application. Measur3D capabilities are used, and changes are made to the client layer to support semantic surfaces' visualization and inspection.

In the following sub-sections, we provide a detailed description of the proposed methodology.

3.1 Data acquisition

Road data acquisition was carried out using a mobile mapping system installed on a moving vehicle. The mission consisted of two parts: data collection in the field and post-processing of the collected data. Other information is acquired during the mission, including GNSS (Global Navigation Satellite System) and IMU (Inertial Measurement Unit) observations for the position and the orientation of the point cloud, in addition to 360° panoramic images for textures.

Post-processing is necessary to deliver an accurate, colored and high-resolution point cloud (Figure 6). This involves initially processing the vehicle's trajectory, then the raw point cloud and panoramic images. During the mission, the positioning system records the absolute trajectory of the platform. This means that the vehicle's position needs to be corrected for better precision. This is done using GNSS observations from a static measurement base set up on a known coordinate point (X,Y,Z) during the mission. The accuracy of the post-processed trajectory is around 2 cm in planimetry and 5 cm in altimetry.

On the basis of the corrected trajectory, the raw point cloud and panoramic images are tied to the coordinate system. In addition, images are used to assign an RGB (red, green, blue) color to each point in the cloud.

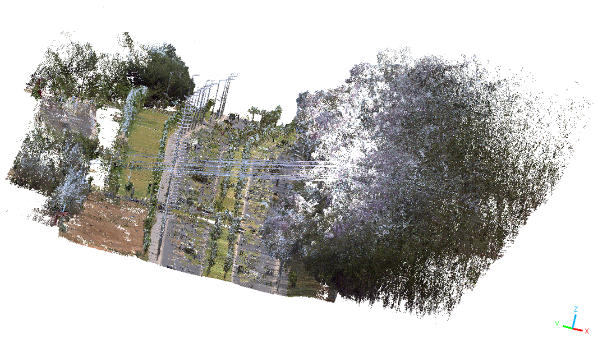


Figure 6. Geo-referenced and colorized point cloud.

3.2 Pre-processing

A noisy point cloud can affect the accuracy of the 3D models. Therefore, in order to use LiDAR (Light Detection and Ranging) data correctly and efficiently in the road modeling process, a cleaning step is necessary to eliminate noise in the measurement. By using CloudCompare software, two cleaning operations are performed: manual coarse cleaning using point cloud interactive segmentation tools, and automatic coarse cleaning using distance-based filters (Figure 7). The coarse cleaning eliminates details that do not belong to the road surface. However, noise removal for a large dataset is more complicated and two filters are used for automatic cleaning:

The Statistical Outlier Removal (SOR) filter is used to remove the noisy points within a local neighborhood. It works by computing the mean and standard deviation of the distance between each point and its k-nearest neighbors. Points that are farther away from their neighbors than a specified threshold are considered outliers and removed.

The Noise Filter works in the same way as the SOR filter, but considers the distance to the underlying surface instead of the distance to neighbors. Around each point in the cloud, a plane is fitted locally, based on a radius or a constant number of neighbors, and then any distant point from the plane is removed.

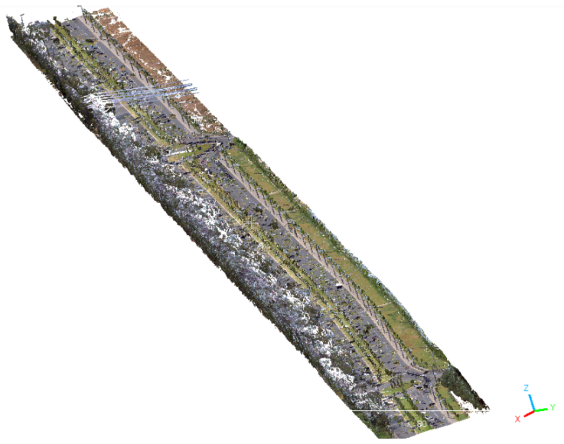


Figure 7. Noise-cleaned point cloud.

3.3 Ground surface extraction

A prior segmentation of the point cloud was performed to separate ground points from overground points, and thus create a Digital Terrain Model (DTM).

The Cloth Simulation Filter (CSF) is used for this operation. Among many ground filtering algorithms, the CSF algorithm presented the best filtering results and produced the lowest total errors (Serifoglu Yilmaz et al., 2018). The filter is based on the Cloth Simulation method, which is an algorithm used in 3D computer graphics to simulate fabric attached to an object. The first step consists of inverting the LiDAR point cloud, then a rigid fabric is used to cover the inverted surface. By comparing the fabric nodes to the corresponding LiDAR points, these nodes can be used to generate an approximation of the ground surface. Finally, ground points can be extracted from the LiDAR data by comparing the original points to the generated ground surface (Zhang et al., 2016).

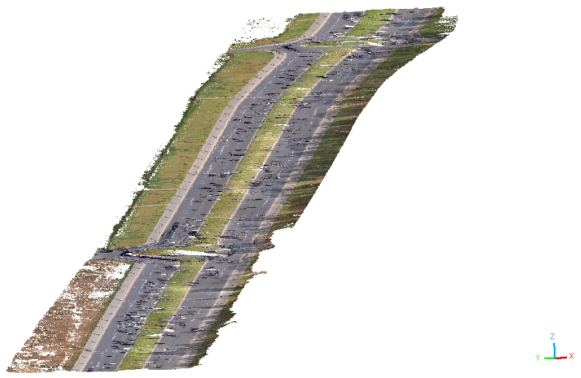


Figure 8. Results of CSF ground points extraction.

3.4 Extraction of linear features

In order to geometrically model the road surface, the ground surface was used to extract the linear features of the road (Figure 9). This was done using Autodesk InfraWorks which allows a semi-automatic extraction of linear features (Figure 10). It first generates the ground surface from the ground points, then performs extraction by simply selecting at least two points along the breakline. This linear structure will then allow a geometric and semantic modeling of the road surfaces. Other lines are added manually to model the road surface according to CityJSON specifications.

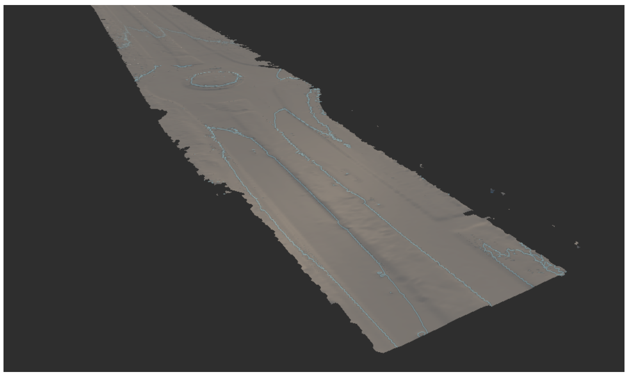


Figure 9. Ground surface used to extract linear features of road.

3.5 Codification system

As described in Table 1, a codification method has been proposed to assign a predefined code to each extracted linear feature. Therefore, each section or intersection and each road surface can be modeled separately.

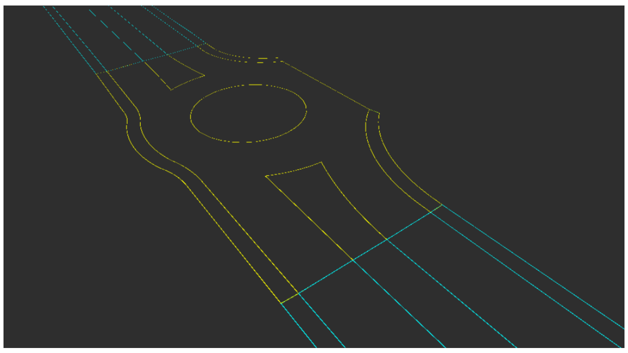


Figure 10. Linear features extracted from the ground surface.

Code	Linear feature
TR	Sidewalk edge
CH	Roadway edge
VT	Green space edge
CHTR	Boundary between sidewalk and roadway
CHVT	Boundary between green space and roadway
CHVTH	Boundary between the central green space of a roundabout and a roadway (H for Hole)

Table 1. Linear features codification system.

For each break line belonging to a single semantic surface, the code consists of two characters. If it's a common boundary between two semantic surfaces, four characters are used, referring to both surfaces. These characters refer to the name of the linear feature in French. In addition to these codes, each linear feature has a *tag* attribute that specifies the section, intersection or roundabout to which this line belongs. For example, lines that belong to the first section might have S1 as *tag* value, S2 for the second section, R1 for the first roundabout and so on (Figure 11).

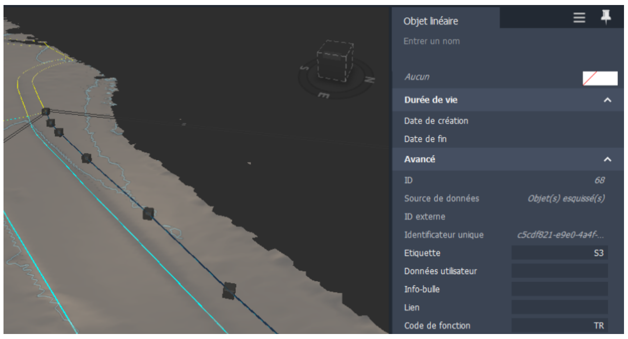

Objet linéaire
Entrer un nom
Aucun
Durée de vie
Date de création
Date de fin
Avancé
ID 68
Source de données Objet(s) esquissé(s)
ID externe
Identificateur unique c5cdf821-e9e0-4a4f-...
Etiquette S3
Données utilisateur
Info-bulle
Lien
Code de fonction TR

Figure 11. Linear feature attributes.

3.6 3D modeling

The 3D road model is created by fusing these lines to generate Composite or Multi Surface geometries. This is done using Safe Feature Manipulation Engine (FME) Workbench, which is a spatial ETL (Extract, Transform, Load) software.

As shown in Figure 12, the first step consists of separating the different classes (sections, intersections and roundabouts) to process them individually. Next, the objects of each class are separated using the *tag* attribute.

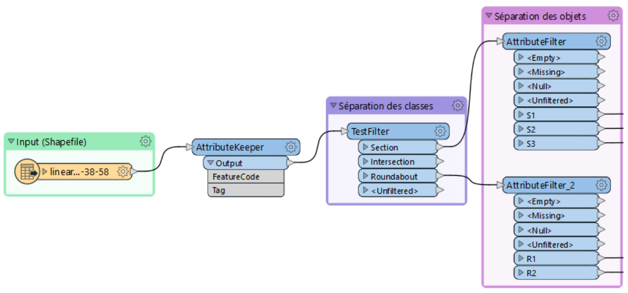


Figure 12. Process of separating sections, intersections and roundabouts.

The road surfaces of each section or intersection are modeled separately. Linear features are filtered based on their pre-assigned codes and then combined to build three semantic surfaces: green space, roadway and sidewalk. The lines of each surface are connected to to create longer poly-lines then converted to polygons by linking their start and end vertices (Figure 13).

The orientation of the generated polygons should be adjusted to control the extrusion direction when generating solids. If the left hand rule orientation type is chosen, the vertices of the outer border of the polygon are arranged in the opposite direction of clockwise. Therefore, the polygons are oriented towards the positive direction of the Z-axis. This can be verified by calculating the surface normal. Subsequently, an extrusion is performed to create solids from the generated polygons. These solids are then converted to Composite Surfaces in order to comply with the geometry types of the Road class in CityGML 3.0.

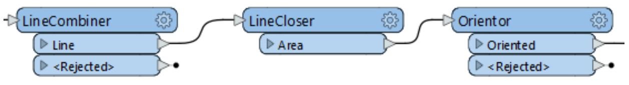


Figure 13. Transforming linear features into polygons.

Semantic information is assigned to each modeled surface by adding corresponding attributes, according to CityJSON specifications. Additionally, the images are used to texture the model (Figure 14).

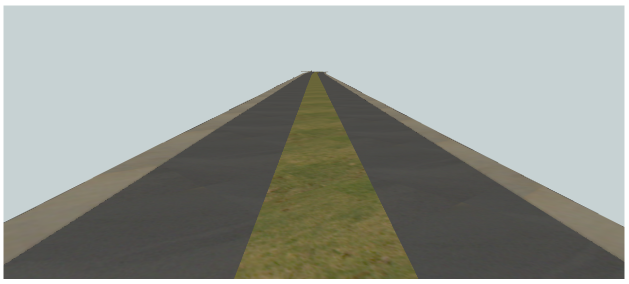


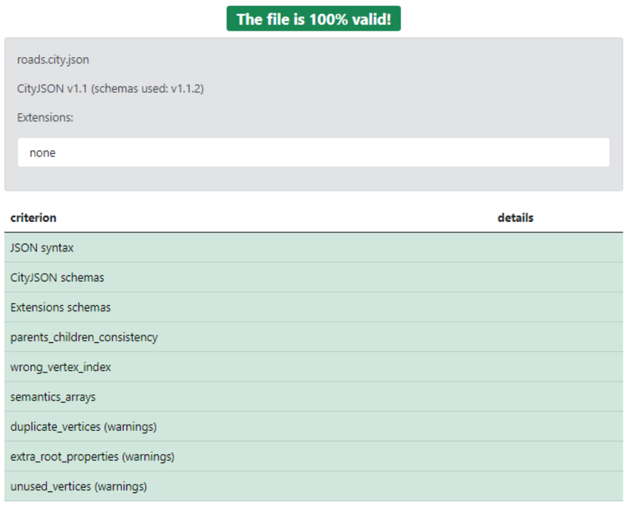
Figure 14. The textured 3D road model.

3.7 CityJSON validation

The resulting model was exported to a CityJSON file. The data schema and geometrical primitives need to be validated to ensure that any non conformity with CityJSON schemas, and any topological and/or geometric errors can be detected and then corrected.

Schema validation was performed using cjval, the official validator for CityJSON files. It verifies the JSON syntax first, followed by the compliance with CityJSON schemas. As shown in Figure 15, the file syntax is valid after applying a patch to make the following changes:

- The version of the CityJSON file (version);
- The coordinate system (referenceSystem);
- The transform matrix;
- Road object geometry type (MultiSurface or CompositeSurface instead of MultiLineString);
- Semantic surfaces (semantics).


The file is 100% valid!

roads.city.json

CityJSON v1.1 (schemas used: v1.1.2)

Extensions:

none

criterion	details
JSON syntax	
CityJSON schemas	
Extensions schemas	
parents_children_consistency	
wrong_vertex_index	
semantics_arrays	
duplicate_vertices (warnings)	
extra_root_properties (warnings)	
unused_vertices (warnings)	

Figure 15. Data schema validation results.

However, compliance with data schemas alone is not sufficient to determine whether a CityJSON file is fully valid. The validity of geometries is also necessary to properly use 3D road models (Bendiksen, 2021). Therefore, it is essential that the basic 3D surfaces, which define the road surface in three dimensions, conform to international standards. To detect these geometric and topological errors, (Ledoux, 2018) proposed a new version of $Val3dity$, an open-source software for validating 3D primitives according to the definitions of the ISO 19107 standard. If a geometry error is reported, the proposed approach is to initially identify the object carrying the error and the vertices causing the invalidity. Correcting these errors requires reviewing the applied transformations used to generate the 3D model. In this case, two geometry errors were detected: non-planarity error and self-intersection error.

A polygon must be planar, within a certain tolerance. This plane is adjusted to the polygon's vertices using the least squares method, and if the distance of a vertex exceeds the tolerance, a planarity error is detected. Self-intersection error can occur due to duplicate nodes or a polygon intersecting itself at a point. For a 3D polygon, self-intersection is checked against its projection onto the best-fit plane (determined by the least squares method) based on the polygon's vertices.

3.8 Geometry errors correction

The detected self-intersection errors are caused by combining lines and closing them to form polygons. This error is corrected by calculating the intersections between the lines that form the polygon before connecting them and creating vertices wherever an intersection occurs (Figure 16).

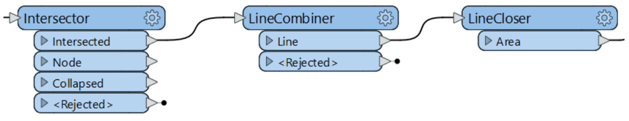


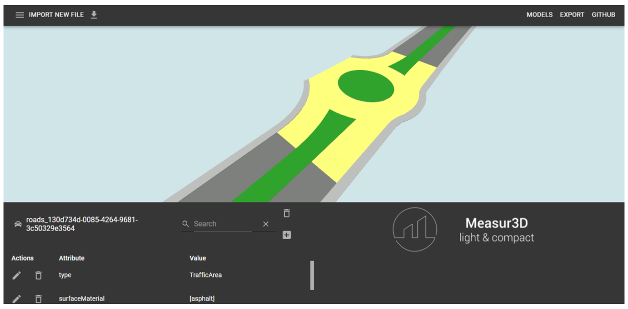
Figure 16. Self-intersection error correction process.

Correcting non-planarity errors in 3D polygons seems to be more challenging. One possible solution is to create a Triangulated Irregular Network (TIN) surface by using the combined poly-line vertices instead of closing them to form a single 3D polygon. A polygon is triangulated by creating line segments between the vertices of the polygon without crossing its exterior, thereby subdividing the polygon into triangles (Ohori et al., 2012).

A second approach can be considered. Instead of triangulating the non-planar polygon, the vertices are overlaid on a planar surface, which can be the plane best fitted by the least-squares method to the point cloud. The resulting polygon is therefore planar.

3.9 Storage and visualization

A valid CityJSON dataset is then stored in a MongoDB database. Measur3D (https://github.com/GANys/Measur3D) enables storage, retrieval and visualization of CityJSON files. It offers a RESTful API for manipulating 3D city models on a document-oriented database. The API endpoints are used to send data to the servers, and thus import CityJSON models into the MongoDB database. Data schemas are already defined by Measur3D. Additionally, the stored data can be retrieved, managed and viewed in a web application.



Actions	Attribute	Value
	type	TrafficArea
	surfaceMaterial	[asphalt]

Figure 17. 3D road model visualized in Measur3D.

Mesaur3D's user interface features three main components: a list of stored CityJSON files, a 3D model viewer and an attribute manager. However, the current version does not support semantic surfaces. The viewer and attribute manager allow to inspect the objects of a 3D city model and their attributes, without access to their semantics. This is justified by the fact that Measur3D client-layer transforms the entire object into a mesh so it can be added to the 3D scene. Each mesh comprises all the combined geometric primitives of the object. Consequently, it is not possible to access semantic surfaces.

The proposed solution consists in parsing the surfaces that make up the object's geometry, instead of converting the entire geometry into a mesh. The code contribution can be viewed at: https://github.com/GANys/Measur3D/tree/dev-anass. As shown in Figure 17, the viewer now supports the inspection of semantic surfaces and their attributes.

4. RESULTS AND DISCUSSION

The proposed methodology brings a valuable contribution to the 3D modeling of road infrastructures. By using mobile mapping LiDAR data, the approach can semi-automatically generate 3D models that are geometrically consistent and accurate. This ensures correct outcomes for analysis and simulation tests on these models. This method proposes a cost-effective alternative to traditional manual modeling approaches while maintaining high levels of accuracy and consistency. Moreover, the use of an ETL approach enables the reconstruction of road models in compliance with CityGML 3.0 Transportation module, which is essential for the integration of these models into 3D city databases. The workflow, based on a sequence of transformers and scripts, enables complete control of the modeling process, and therefore quick and efficient troubleshooting of errors or irregularities within the final model.

As new versions of CityJSON are constantly being released, changes in the data schema can result in compatibility issues. To avoid such problems, the output file must be regularly corrected to conform to the latest published version. Additionally, geometric and topological errors, such as self-intersections or non-planar surfaces, can affect the accuracy of the 3D models, ultimately leading to inaccurate simulation outcomes or analysis results. Therefore, it is essential to ensure that all objects in the city have valid geometries to guarantee accurate and reliable simulations or analysis.

Managing CityJSON datasets using a file system has its drawbacks in terms of performance. Conversely, storing and retrieving 3D road models from a database is faster and more efficient. Given that CityJSON data is semi-structured, the use of a document-oriented NoSQL database offers the best storage solution. Nevertheless, the documents to be stored must respect some predefined schemas to guarantee consistency.

However, it is important to note that a semi-automatic approach in an urban context can have limitations. In such contexts, road networks extend over several kilometers, and a supervised approach can be inefficient and time-consuming in terms of performance. Therefore, future research should explore the development of an automatic approach to the detection and codification of linear features from point cloud data to improve the modeling process's efficiency and scalability.

5. CONCLUSION

Our study presents a systematic approach for generating a standardized 3D model of road infrastructures that is both cost-effective and accurate. Our approach involves a semi-automatic extraction of linear features from mobile mapping LiDAR data, which enables geometric and semantic modeling of roads. We also propose a codification system for assigning predefined codes to each linear feature, allowing for separate modeling of each section, intersection, and road surface. The resulting model is validated then stored in a document-oriented database and visualized through a web application. The proposed methodology brings a valuable contribution to the 3D modeling of road infrastructures, and future research should explore the possibility to improve the modeling process's efficiency and scalability.

REFERENCES

Beil, C., Kolbe, T. H., 2017. CITYGML AND THE STREETS OF NEW YORK - A PROPOSAL FOR DETAILED STREET SPACE MODELLING. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-4/W5, 9–16. https://isprs-annals.copernicus.org/articles/IV-4-W5/9/2017/.

Beil, C., Ruhdorfer, R., Coduro, T., Kolbe, T. H., 2020. Detailed Streetspace Modelling for Multiple Applications: Discussions on the Proposed CityGML 3.0 Transportation Model. *ISPRS International Journal of Geo-Information*, 9(10). https://www.mdpi.com/2220-9964/9/10/603.

Bendiksen, T., 2021. Creating a workflow of 3D building data in a municipality context. https://lup.lub.lu.se/student-papers/search/publication/9058579. Student Paper.

Boersma, F., 2019. Modelling different levels of detail of roads and intersections in 3D city models. http://resolver.tudelft.nl/uuid:ebfc48f8-4704-47d3-9654-cd00c765e0af.

Kutzner, T., Chaturvedi, K., Kolbe, T. H., 2020. CityGML 3.0: New Functions Open Up New Applications. *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, 88(1), 43-61. https://doi.org/10.1007/s41064-020-00095-z.

Ledoux, H., 2018. val3dity: validation of 3D GIS primitives according to the international standards. *Open Geospatial Data, Software and Standards*, 3(1), 1. https://doi.org/10.1186/s40965-018-0043-x.

Ledoux, H., Arroyo Ohori, K., 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, Software and Standards*, 4(1), 4. https://doi.org/10.1186/s40965-019-0064-0.

Nys, G.-A., Billen, R., 2021. From consistency to flexibility: A simplified database schema for the management of CityJSON 3D city models. *Transactions in GIS*, 25, 3048 - 3066.

Ohori, K., Ledoux, H., Meijers, M., 2012. Validation and Automatic Repair of Planar Partitions Using a Constrained Triangulation. *Photogrammetrie - Fernerkundung - Geoinformation*, 2012, 613-630.

Serifoglu Yilmaz, C., Yilmaz, V., Gungor, O., 2018. Investigating the performances of commercial and non-commercial software for ground filtering of UAV-based point clouds. *International Journal of Remote Sensing*, 39, 5016-5042.

Vitalis, S., Labetski, A., Boersma, F., Dahle, F., Li, X., Arroyo Ohori, K., Ledoux, H., Stoter, J., 2020. CITYJSON + WEB = NINJA. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, VI-4/W1-2020, 167–173. https://isprs-annals.copernicus.org/articles/VI-4-W1-2020/167/2020/.

Zhang, W., Qi, J., Wan, P., Wang, H., Xie, D., Wang, X., Yan, G., 2016. An Easy-to-Use Airborne LiDAR Data Filtering Method Based on Cloth Simulation. *Remote Sensing*, 8(6). https://www.mdpi.com/2072-4292/8/6/501.

Zhang, X., Zhong, M., Liu, S., Zheng, L., Chen, Y., 2019. Template-Based 3D Road Modeling for Generating Large-Scale Virtual Road Network Environment. *ISPRS International Journal of Geo-Information*, 8(9). https://www.mdpi.com/2220-9964/8/9/364.