ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume X-4/W3-2022
7th International Conference on Smart Data and Smart Cities (SDSC), 19–21 October 2022, Sydney, Australia

PY3DTILERS: AN OPEN SOURCE TOOLKIT FOR CREATING AND MANAGING 2D/3D GEOSPATIAL DATA

L. Marnat*,[1] C. Gautier,[1] C. Colin,[2,3] and G. Gesquière[2]

[1]Université de Lyon, UCBL, INSA Lyon, LIRIS, UMR-CNRS 5205, F-69621 Villeurbanne, France
(lorenzo.marnat, corentin.gautier)@universite-lyon.fr
[2]Université de Lyon, Univ Lyon 2, UCBL, INSA Lyon, LIRIS, UMR-CNRS 5205, F-69676 Bron, France
(clement.colin, gilles.gesquiere)@univ-lyon2.fr
[3]Berger-Levrault, Limonest, France

Commission IV, WG IV/9

KEY WORDS: 3D, Urban data, 3D Tiles, Visualization, Standard

ABSTRACT:

In recent years, the production of 3D geospatial data using formats such as IFC, CityGML and GeoJSON, has increased. Visualizing this data on the web requires solving a variety of problems, such as the massive amount of 3D objects to be visualized at the same time and the creation of geometry suitable for a 3D viewer. Cesium and OGC introduced the 3D Tiles format in 2015 to solve these issues. They have created a specific format optimized for streaming and rendering 3D geospatial content, based on the glTF format developed by Khronos. The recency of the 3D Tiles format implies the need to experiment around this format and to test its interoperability with other geospatial and urban data formats. There is also the will to innovate on the organization of 3D objects in order to offer a better control on the visualization. Therefore, there is a need for an open source tool capable of converting 3D geospatial data into 3D Tiles to visualize them on the web, but also to test and develop new methods of spatial clustering and creating Levels of Detail (LoD) of urban objects. We propose Py3DTilers in this paper, an open source tool to convert and manipulate 3D Tiles from the most common 3D geospatial data models: CityGML, IFC, OBJ, and GeoJSON. With this tool, we ensure that the generated 3D Tiles respect the specification described by the OGC, in order to be used in various viewers. We provide a generic solution for spatially organizing objects and for creating LoDs, while allowing the community to customize these methods to go further in finding efficient solutions for visualizing geospatial objects on the web.

1. INTRODUCTION

For the last twenty years, new standards to describe urban objects have emerged, such as CityGML (Kolbe et al., 2005) for 3D GIS or IFC[1] for BIM. These new standards are increasingly used thanks to the recent development of methods to acquire 3D geospatial data. These data are used, for example, to assist in urban planning and analysis (Hor et al., 2018, Jaillot et al., 2017). These standards describe structured data models, containing thematic and geometric information, and are mainly built around a desire to exchange information. The visualization of geometric information of these data in real time on the web is therefore complex, particularly due to their data model heterogeneity. The number of objects to display can also be a problem when we want to visualize a city in its entirety.

To address the emerging need for solutions to visualize massive, large-area, multi-scale 3D geospatial content, Cesium and OGC have introduced the 3D Tiles[2] community standard. The conversion of the most common geospatial data models to this format is a solution increasingly adopted by the community (Chen et al., 2018, Gaillard et al., 2018, Jaillot et al., 2020) to visualize these data on the web. 3D Tiles is an evolving standard supported by the community. It is necessary to propose an open source tool to create 3D Tiles while offering the possibility to experiment, stress this standard, and innovate. In this purpose, there is need for a tool to:

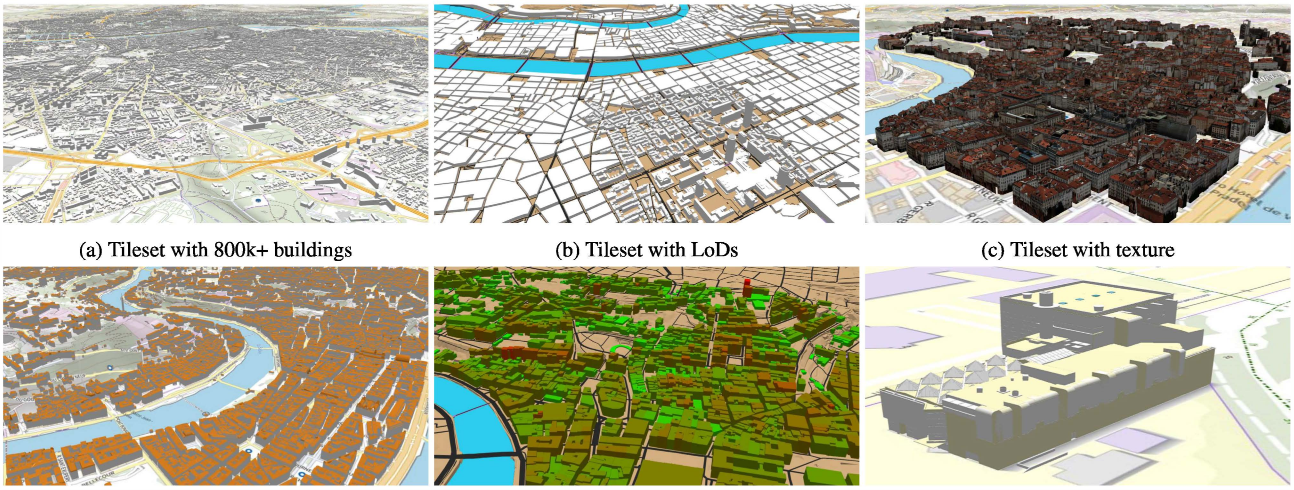
- Create and test extensions to the 3D Tiles format.
- Create and test different generic methods of object distribution and LoDs creation to ensure efficient rendering, or improved relevance to the user (Gaillard et al., 2018).
- Support and test the possible evolution of 3D Tiles
- Support different standards to be reusable on all data sources.

In this article, we propose Py3DTilers[3], an open source tool to explore the potentialities of 3D Tiles. It allows the creation of 3D Tiles from the most popular standards (CityGML, IFC, OBJ[4], GeoJSON[5]). This work also allows to add semantic information related to geometries, in order to be able to visualize them in the same context on the web in real time. This semantic information can be attributes and types from the input data, or additional information computed during the creation of the 3D Tiles. Moreover, a particular attention was addressed to the architecture of this tool in order to allow the easy creation of 3D Tiles from other formats, but also to be able to create, modify, and test new methods of organization of objects in space and LoD creation.

This work allows us to test different ways of composing data with 3D Tiles. It is also a necessary work to achieve different

[1] https://technical.buildingsmart.org/standards/ifc/ifc-schema-specifications/
[2] http://docs.opengeospatial.org/cs/18-053r2/18-053r2.html
[3] https://github.com/VCityTeam/Py3DTilers
[4] https://www.loc.gov/preservation/digital/formats/fdd/fdd000507.shtml
[5] https://datatracker.ietf.org/doc/html/rfc7946

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-X-4-W3-2022-165-2022 | © Author(s) 2022. CC BY 4.0 License.



(a) Tileset with 800k+ buildings (b) Tileset with LoDs (c) Tileset with texture

(d) Tileset colored by CityGML classes (e) Tileset colored by height (f) Tileset colored by IFC classes

Figure 1. Examples of 3D Tiles created with Py3DTilers

objectives. The first use case can be the computerized maintenance management system (Colin et al., 2022) where a multi-scale view of the same territory is needed (from the city to the interior building). Many tests have been performed with the tools proposed in this article to create a multi-scale 3D Tiles. A second use case can be the possible composition of data in a given space for a web documentary (Gautier et al., 2022), where the management of different types of data is also necessary. The choice was to use 3D Tiles as a pivot format.

We present in Section 2 the 3D Tiles format and its uses in various works on the visualization of geospatial data. The architecture of Py3DTilers, its functionalities and *Tilers* are presented in Section 3. Some examples of 3D Tiles created with Py3DTilers will be presented in Section 4, followed by a discussion in Section 5 about the difficulties encountered as well as current and possible future improvements.

2. RELATED WORKS

2.1 3D Tiles

3D Tiles is an open source community standard, described by Cesium and OGC. It has been designed to support the massive visualization of 3D geospatial content, while taking into account streaming and rendering aspects. It differs from the I3S [6] format thanks to its inclusion in the OGC standards. It is used by numerous different actors and evolve following the Khronos proposals, which ensure a strong link between computer graphics and geo-information science. Its flexibility and transparency allows the possibility to create extensions adapted to a specific need (Jaillot et al., 2020). This standard makes it possible to describe a tree of 3D geospatial *tiles* called *tileset*. This tree allows a spatial organization of the tiles optimized for the rendering of spatial objects, in particular by supporting various tiling methods (K-d tree, octree ...) but also the concept of Hierarchical Level Of Detail. The method used to create the tree and the tiles can have a direct impact on the visualization of objects (Zhan et al., 2021). It is therefore necessary to have a tool to test different tiling methods in order to optimize the rendering and visualization of 3D models from different sources.

Tiles can have different formats:

- Batched 3D Model (B3DM): Heterogeneous 3D models. E.g. textured terrain and surfaces, 3D building exteriors and interiors, massive models.
- Instanced 3D Model (I3DM): 3D model instances. E.g. trees, windmills, bolts.
- Point Cloud: Massive number of points.
- Composite: Combining tiles of different formats into one tile.

The B3DM and I3DM formats describe geometries using the glTF[7] format, which is a geometry format described by Khronos that aims to facilitate the streaming and rendering of 3D models on the web.

Each tile contains a set of *features*: 3D models representing, for example, buildings, trees, etc. It is possible to associate specific semantic to each *feature* using *Feature Tables*, like the height or the color, or to a whole tile using *Batch Tables*.

2.2 Geospatial data visualization using 3D Tiles

Recent research has made it possible to improve the visualization of geospatial data on the web, first by using the glTF format (Schilling et al., 2016) and then by converting the data to the 3D Tiles format. This solution is justified by the performance of this format for this type of data, in particular thanks to the spatial organization of the objects in tiles. An example of this performance is demonstrated by (Kulawiak and Kulawiak, 2017), allowing a smooth visualization of about 500Gb of point cloud data, at a precision of about 19 points/m², on an area of about 1400km².
At the city scale, (Gaillard et al., 2018, Mao et al., 2020) focused on the conversion of CityGML data to 3D Tiles in two approaches. The first one for the exchange and representation of city data; the second one allowing the visualization of simulations of the city. Also, the use of 3D Tiles and its extensions allowed to visualize the evolution of the city over time (Jaillot et al., 2020).
For building data, we find solutions to convert IFC data into 3D Tiles. The work of (Chen et al., 2018) presented a solution completely based on open source tools. (Xu et al., 2020) improved

[6] https://github.com/Esri/i3sspec

[7] https://github.com/KhronosGroup/glTF

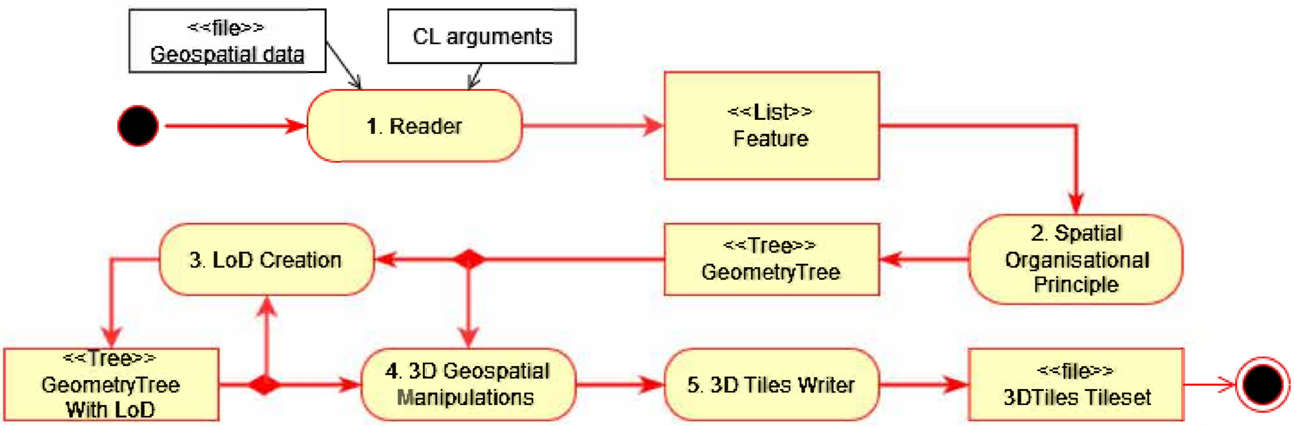


Figure 2. Activity diagram of the 3D Tiles creation process

this solution by allowing to add semantic information from BIM in Batch Tables. Finally, (Zhan et al., 2021) have demonstrated that the method of organizing objects into tiles plays a crucial role in the fluidity and comprehension of the navigation in BIM data.

Open source solutions using databases are also starting to appear: 3DCityDB (Yao et al., 2018) for CityGML and BIMserver (Beetz et al., 2010) for IFC. They allow glTF exports, in order to visualize these data on the web. The work (Hijazi et al., 2020) has shown that the combination of these two solutions allows visualization of heterogeneous data on the web. Nevertheless, the 3D Tiles overlay is missing to allow the spatial organization of objects. Moreover, to test different methods of spatial organization of objects or to treat other data formats, it seems complex to modify the behavior of these two tools at the same time.

There is a set of solutions on the market, FME[8], rhinocity[9] which leaves little control to the user in the creation of tilesets and does not allow the modification of the tile creation process, and thus to experiment around the 3D Tiles format. Others are open, but insufficient. For instance, F4DConverter[10] only allows the creation of geometry in F4D format, usable only by the Mago3D software[11].
Py3DTiles[12] is an open source Python library allowing the representation and creation of 3D Tiles. This library has been developed by the Liris laboratory and the company Oslandia. It allows to manipulate two 3D Tiles formats: Batched 3D Models (B3DM) and Point Clouds (PNTS). Py3DTiles integrates the possibility to read and write these two formats of 3D Tiles, but is especially developed with the aim of creating 3D Tiles in PNTS format from LAS files (lidar point cloud). pg2b3dm[13], based on this library, could allow to create its own methods of spatial organization of objects and creation of LoD. However, it does not provide a solution for extracting geometry from popular formats for the creation of 3D Tiles because it is based on geometry stored in PostGIS database. Also, it does not take into account the current evolution of 3D Tiles. Py3DTiles is used as a basis of this work.

[8] https://www.safe.com/
[9] https://www.rhinoterrain.com/en/rhinocity.html
[10] https://github.com/Gaia3D/F4DConverter
[11] http://www.mago3d.com/
[12] https://gitlab.com/Oslandia/Py3DTiles
[13] https://github.com/Geodan/pg2b3dm

3. PY3DTILERS

Py3DTilers offers open source tools to create 3D Tiles in B3DM format. Py3DTilers is based on the Py3DTiles library, to which it adds *Tilers*. Each *Tiler* is a tool that builds 3D Tiles from a specific data format. The 3D Tiles can be created from a multitude of formats: GeoJSON, OBJ, CityGML, IFC or from existing 3D Tiles. Each format has its own *Tiler*, which reads the geometries and associated data and then represents them in memory as objects named *features*.

The process of transformation and writing of these objects in 3D Tiles is shared by all the *Tilers*. During this transformation, Py3DTilers offers a large number of options allowing to carry out 3D geospatial manipulations (translation, scaling, conversion of projection system), customize the method of distribution of the *features* in tiles and add levels of detail.

3.1 Py3DTiles

Py3DTilers uses the Py3DTiles library to represent 3D Tiles in memory and then modify them. Multiple additions, modifications and corrections have been made to this library in order to robustify the creation of 3D Tiles in B3DM format. This allows Py3DTilers to produce 3D Tiles in the B3DM format that conform to the standard.

Modifications have been made to represent all the notions specific to 3D Tiles in the form of classes. The new classes, for example, introduced the notions of tile, tileset or extension. This allows to have an object oriented representation of an entire 3D Tiles tileset. This new version of Py3DTiles also facilitates the creation and writing of glTF materials.

Changes have been made to integrate Batch Tables and Feature Tables. These tables allow to keep attributes related to the 3D models of the tile. The attributes that must be present in each of the tables differ depending on the type of 3D Tiles being created (B3DM or PNTS). The modifications ensure that the resulting B3DMs contain all the necessary attributes.

A JSON schema validation system has been implemented to ensure that the tilesets produced respect the 3D Tiles specification. These schemas also allow to define new extensions. In our case, two extensions have been integrated. The first is the *Batch Table Hierarchy*, a standard extension to the 3D Tiles Batch Table. This extension allows to store attributes in a more flexible way than with a basic Batch Table. In particular, this extension integrates the notion of hierarchy and typing of the models of a tile. The second extension is a temporal extension. It allows to represent the temporality of the 3D models of the tiles and to store the operations to be performed to go from one year to another.

3.2 Global Architecture - Feature

The task of the Py3DTilers is to transform the data provided by the user into 3D Tiles. Each Tiler is able to read a specific data format. The process of creating 3D Tiles is described in figure 2.

- *Step 1: Reader* The *Tiler* reads the data in order to extract the objects and their associated data and geometry. If the geometries are 2D or are not triangulated, the *Tiler* also takes care of transforming them into sets of triangles forming 3D models. The read data will then be represented by a list of *features*. Each *feature* is an object characterized by an identifier, a triangulated geometry and additional semantic data. *Features* allow to abstract from the input format: the objects read from data will always be represented as *features*, whatever the input format. This abstraction allows for a unique process of transforming the *features* into 3D Tiles, which is not dependent on the input format. This also allows objects from different sources to be manipulated in the same 3D Tiles creation process.

- *Step 2: Spatial Organisational Principle*. The *features* are distributed in a tree of geometries, where each node of the tree has a subset of the *features*. This distribution allows to create a spatial or semantic partition of the objects according to the visualization modalities. In our case, there are three main possible distributions: a spacial partitioning with a k-d tree[14] or a geographical partitioning according to an external data (for example, districts, blocks of buildings, etc). The third and last possibility is to keep the distribution induced by the data format without performing any additional operation.

- *Step 3: LoD Creation*. The creation of levels of detail is optional, the user can choose to add one, several or none. The levels of detail take the form of new *features* with their own geometry. These *features* will also be added to nodes, which will be placed in the tree of geometries as parents of the already existing nodes. We obtain a tree where the root nodes are the nodes containing the 3D models with the lowest level of detail, and where the level of detail increases as we go down towards the root nodes.

- *Step 4: 3D Geospatial Manipulations*. Optional geospatial manipulations are applied to the geometries of the models. These operations are chosen by the user and include, for example, reprojection, scaling or translation on the (X, Y, Z) axes.

- *Step 5: 3D Tiles Writer*. The tree of geometries is transformed into 3D Tiles. Each node of the tree becomes a tile of the tileset. The hierarchy between the nodes is preserved in the tiles: a node with 3 child nodes will become a tile with 3 child tiles. The *features* are written in the content of their respective tiles, materialized by B3DM files. The semantic data of the *features* are stored in these files in the Batch Tables and Feature Tables. The files also contain the geometries in glTF format, encoded in binary. Each 3D model is differentiated from the others by a *batch id*. This id is also used to make the link between a model and its data in the Batch and Feature Tables.

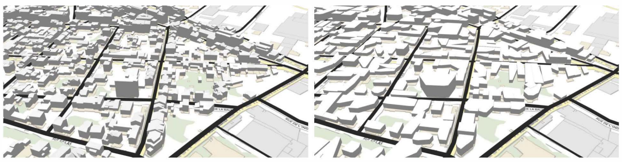
[14] https://en.wikipedia.org/wiki/K-d_tree

3.3 Possible operation on features

3.3.1 Geometric error and LODs: A Level of Detail (LoD) corresponds to the complexity of a 3D model. A 3D model can have several levels of detail, each more or less detailed. Thus, it is possible to switch from one LoD to another depending on the needs. For example, if a model is too far from the camera to be seen in detail, choosing to reduce its level of detail allows to save rendering time without affecting the visualization.

Py3DTilers allows the creation of several levels of detail for geometries. There are two methods of LoD creation in Py3DTilers yet, but other methods can be implemented. These levels of detail are simplifications of models of one or more *features*. The LoDs are contained in tiles, with one tile per level of detail. These tiles contain the 3D models corresponding to a lower level of detail of the models of their child tiles.

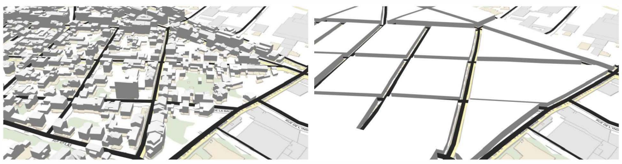
The first LoD creation method consists in computing a convex footprint of the *features*. Those footprints are then extruded in order to obtain simplified 3D models for each *feature*. As shown in figure 3, used on 2.5D buildings, this method allows to obtain shapes with fewer walls and no roof details.



(a) Original buildings (b) Extruded footprints

Figure 3. 3D extrusion of buildings footprints

The second method uses a set of polygons to split a tile into several blocks. Each block is transformed into a 3D volume representing several *features*. In figure 4, these blocks are defined by the network of roads of the city, where all the buildings in the same block are represented by a single model.



(a) Original buildings (b) Blocks

Figure 4. Blocks of buildings defined by the roads

In order to allow the passage from one level of detail to another, 3D Tiles integrate a refinement system allowing to prioritize the tiles when rendering. Refinement is used to choose which tiles to show or hide. A tile has a refinement method and can have a parent and one or more children. The two possible refinement methods are ”REPLACE” or ”ADD”. In the first case, a tile with children will be replaced by them during a refinement. In the second case, the child tiles are added to the parent tile.

In a 3D Tileset, the geometric error is the value that defines a threshold error above which a tile will be refined. A tile must always have a lower geometric error than its parent. The tile at the root of a tileset must always have the highest geometric error. Leaf tiles must have a smaller geometric error than all other tiles. The geometric error allows to customize the visualization by choosing at which distance from the camera the tiles should be refined. A tile with a high error will be refined even if the camera is still far from the 3D model. On the contrary, a tile with a low error will be refined only when the camera is close to it. Py3DTilers integrates the possibility to choose a geometrical error for each tile according to its position in the tileset hierarchy. Thus, it is possible to define a geometrical error for all leaf tiles, root tiles, or the tiles of any other level of the hierarchy.

3.3.2 Styling: It is possible to create materials with Py3DTilers that will be written in the glTF content of the 3D Tiles. These materials can have either a texture (linked to an image) or a solid color defined by an RGBA value. When creating 3D models, each *feature* has an assigned material and several *features* can share the same material.

The materials are created according to the user's modalities. The user can choose to create textured materials if the input data already has textures (figure 1c). The texture images will be written in texture atlases, with one atlas per textured tile. It is also possible to create colored materials according to the attributes of the *features*, for example, the height (figure 1e) or the class (figures 1d and 1f). The colors applied to the materials are customizable via JSON files.

3.3.3 Common 3D object manipulations:

- *Translation :* allows to move 3D models on the X, Y and Z axes. A 3D vector will be subtracted from the coordinates of each vertex when creating the 3D Tiles. This operation makes it possible to correct position errors of the data, as shown in figure 5, or to place a non-georeferenced model at geospatial coordinates. Conversely, it is possible to use translation to center a 3D model around the coordinates (0, 0, 0), which is necessary in many 3D softwares.

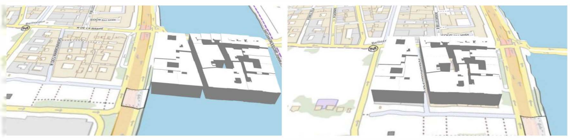


Figure 5. Buildings translated on X axis

- *Scaling*: allows to change the scale of the geometries, by increasing or reducing it. In the same way, it allows to switch from one unit of measurement to another (for example, from centimeters to meters or vice versa). These operations can be useful in order to correct problems with the size of geometries or to display in the same context *features* with different scales or units of measurement in the source data.

- *Reprojection*: allows to modify the Coordinates Reference System (CRS) of a dataset. A CRS is a reference system allowing to locate geographical coordinates on the globe. A CRS can be global or localized on a more restricted geographical area. The choice of the CRS is crucial in the processing and representation of geospatial data. It is closely related to the position and extent of the data on the globe. Many data providers or GIS software impose a specific CRS. Py3DTilers allows to easily project 3D Tiles from one CRS to another by indicating during the transformation which is the input projection and which projection is wanted as output.

3.3.4 Geometry visualization: In addition to producing 3D Tiles, Py3DTilers allows to export to the OBJ format. This format is supported by most 3D visualization tools and can be easily edited or inspected. Exporting to OBJ allows to quickly check the appearance of a 3D Tiles tileset or to export 3D Tiles in a format supported by a larger number of tools. This option is all the more important as there are very few tools allowing the visualization and inspection of 3D Tiles. However, the OBJ format loses a lot of information from 3D Tiles. The tiling and the division into separate entities of the models of a tile are not preserved in the OBJ file. In addition, all attributes attached to the models in the Batch Table and Feature Table are lost.

3.4 Tilers

3.4.1 CityGML: CityGML is a data model used to represent city objects and the urban landscape. This model describes buildings, bridges, water bodies and terrain. The 3D objects are stored as a set of surfaces, where each surface can be linked to a texture. Py3DTilers is able to create 3D Tiles from CityGML files through a 3DCityDB database, a geospatial database implementing the CityGML standard.

In order to create the 3D Tiles, the city objects of the chosen type as well as their surfaces are retrieved from the database. According to the choice of the user, the surfaces of the same object are either merged to form a single 3D model, or left independent. The extension *Batch Table Hierarchy* allows to keep the surface/object hierarchy of CityGML in the tiles.

3.4.2 IFC: The IFC data model is mainly used to represent infrastructures (buildings, bridges, tunnels, etc.) for Building Information Modelling (BIM). It describes geometric and semantic data of the objects that compose them (walls, ceilings, water network). Moreover, the hierarchy of the objects is described, for example, all the floors associated with a building, the rooms associated with each floor, etc. Each object is positioned relative to its parent. In order to create a tileset, the IFC objects with geometries are retrieved using an open source library named IfcOpenShell[15], as well as based on their position in the real world. Geometries can be described using different representations (tesselated, brep, boolean operation) within the model. The translation of IFC geometry into a 3D model is also performed using the same library. It is possible to group objects by IFC Class or by IfcGroup, which is a grouping of objects that can be done in a BIM modeling software.

3.4.3 GeoJSON: The GeoJSON format is used to represent geospatial data where the geometry is in the form of polygons, lines, points or a combination of these shapes. Each independent geometry is called a "feature". Attributes can be associated to each of these GeoJSON features.

With Py3DTilers, it is possible to read GeoJSON files and transform them into 3D Tiles. To do this, Py3DTilers reads each GeoJSON feature of a file and retrieves its geometry and attributes. The coordinates of the geometries can be either 2D or 3D. If the geometry is in the form of polygons, these are extruded and triangulated to obtain a 3D model. If the geometry

[15] http://ifcopenshell.org/

is in the form of lines, we apply a buffering operation: the lines are transformed into polygons by adding a thickness. Once the lines are buffered, they are extruded and triangulated to make models.

When transforming GeoJSON features into 3D models, the height of the extrusion, the thickness of the lines or the altitude at which the 3D models will be placed can be either:

- defined by a value present in the attributes of the features. The user can indicate which attribute to read for these values.
- arbitrarily chosen by the user. The chosen values will be used for all features.

3.4.4 OBJ: Py3DTilers allows to read 3D models in OBJ format. The models are loaded in memory with the open source Python library 'pywavefront'[16]. The geometry of each 3D model is then cut into a set of triangles to prepare the transformation into glTF. During the transformation, it is also possible to keep the texture of the OBJ models and write it in the 3D Tiles.

3.4.5 3D Tiles tileset: Py3DTilers is able to read and load 3D Tiles (in B3DM format only) into memory. Once loaded, the geometry of each 3D model and its associated Batch Table data are stored as *features*. The data related to the tiles (bounding volumes, geometric errors, etc.) are also stored in memory. This allows to make modifications to an existing tileset using all the operations available in Py3DTilers. For example, it is possible to project a tileset in another CRS or to move it on the X, Y, Z axes. The 3D Tiles thus modified are then written to disk in a new tileset.

Reading 3D Tiles also allows to merge several 3D Tiles tilesets. Merging preserves the hierarchy and tile distribution of the 3D Tiles. It is not possible to insert a tileset into the hierarchy of another tileset. Reading and rewriting 3D Tiles also does not allow the creation of new tiles or new levels of detail.

4. EXPERIMENTATION

4.1 Validation

The 3D Tiles produced by Py3DTilers conform to the specification of the OGC. To ensure this conformity, a double validation is performed: first by the JSON schemas validator system, then by Cesium's 3d-tiles-validator[17] tool. The first validation ensures that the tile and tileset fields are correctly written and that no necessary fields are left out. The validation tool of Cesium verifies that the content of the tiles is correctly written and encoded.

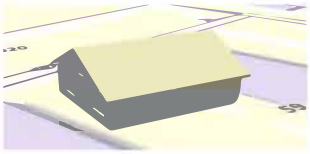
The geometries are preserved during the transformation into 3D Tiles. The representations of the geometries vary between each format, and the treatment of these representations for their visualizations varies from one software to another. Nevertheless, as shown in the figure 6, the generated 3D models respect the geometry of the input data identically.
Moreover, Py3DTilers preserves the *feature* partitioning of the source: each distinct object in the input data will be a distinct 3D model in the tile that contains it. Finally, Py3DTilers can keep additional semantic data associated with the geometries in the Feature and Batch Tables of the tiles.

[16] https://github.com/pywavefront/PyWavefront
[17] https://github.com/CesiumGS/3d-tiles-validator

(a) CityGML in FME

(b) 3D Tiles in UD-Viz

(c) Ifc in Bim Vision[a]



(d) 3D Tiles in UD-Viz

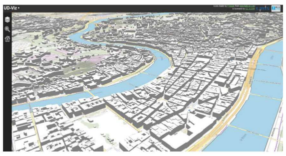
[a] https://bimvision.eu/

Figure 6. Comparison of the geometries from CityGML and Ifc with the geometries transformed as 3D Tiles

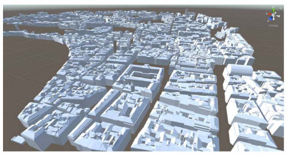
4.2 Visualization

It is possible to view the tilesets created with Py3DTilers with all the tools adapted to the display of 3D Tiles (figure 7). In particular, it is possible to import the 3D Tiles into Cesium ion, in order to visualize them in Cesium, as long as the 3D Tiles have been written in the EPSG:4978 projection. The 3D Tiles can also be viewed in any projection in iTowns[18], and in UD-Viz[19] (based on iTowns). UD-Viz allows handling these 3D Tiles extensions. Unity[20] can also display 3D Tiles, thanks to the Unity3DTiles[21] project proposed by NASA. In this case, it is advised to either choose a projection with low orders of magnitude of the coordinates, or to use the translation option of Py3DTilers to obtain non-georeferenced 3D Tiles.

(a) Cesium



(b) UD-Viz



(c) Unity

Figure 7. Visualisation of 3D Tiles with different tools

The different *Tilers* of Py3DTilers allow the creation of 3D Tiles representing different geospatial data layers: buildings, relief, rivers, roads, and bridges, as shown in figure 8. Buildings and bridges can, for example, be produced from CityGML or IFC data, with several levels of detail (figure 1b). Roads and rivers can be created from polygons or lines from GeoJSON files.

The Py3DTilers options offer the possibility to choose the size of the tiles as well as their geometric errors and levels of detail. This control over the creation of 3D Tiles allows to easily customize the visualization according to the user's needs. For example, it is possible to create tileset that is refined at a

[18] https://www.itowns-project.org/
[19] https://github.com/VCityTeam/UD-Viz
[20] https://unity.com/
[21] https://github.com/NASA-AMMOS/Unity3DTiles

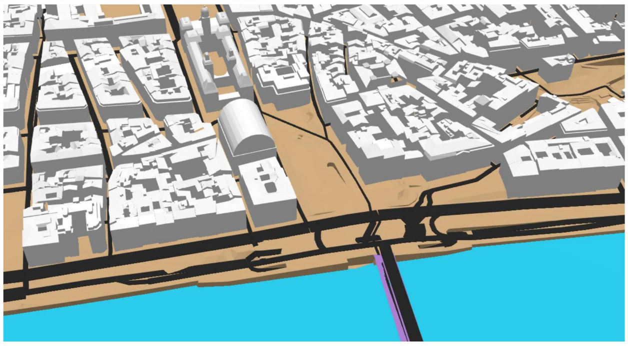


Figure 8. 3D Tiles of buildings, relief, roads, bridges and water bodies

greater distance from the camera or to create a tileset with a very large number of tiles, each containing very few different objects. Py3DTilers allows to experiment by creating tiles and tilesets according to specific needs.

4.3 Reproducibility

To allow easy use of the tool, a docker[22] is available for the community, with a documentation consisting of the technical architecture of the tool, and tutorials for use according to the different input formats and the possible options to customize the creation of 3D Tiles.

In order to reproduce the 3D Tiles of the figures 1f and 1e, a tutorial and dockers are available[23]. In addition, the data used are made available. A solution to visualize the 3D Tiles produced from these data with UD-Viz is also described.
Finally, some examples of 3D Tiles produced with Py3DTilers are available[24]. Online demonstrations[25] allow to visualize some tilesets produced with Py3DTilers.

4.4 Limitations

3D Tiles standard provides a good opportunity to manage and visualize a large amount of 3D data at different scales. But even if the data is loaded in a progressive way, it is currently difficult to visualize a large dataset, especially if the 3D models are textured. A first attempt is to use the LoD process to reduce the resolution level of irrelevant geometries, but the textures are still too heavy in RAM. Using LoDs on texture images may induce a new way of texturing 3D objects in a progressive manner, which would allow textures to change over time. This problem is not yet solved: Py3DTilers can't compute progressive textures or create meshes with more than one textured material.

An other limitation of 3D Tiles, and by extension of Py3DTilers, is the recency of the format. It is still evolving: a new version 1.1 is being considerated by the OGC. Thus, existing tools will need to evolve to follow the standard. Furthermore, the 3D Tiles format is not yet supported by many visualization softwares. It can be difficult to use the 3D Tiles produced elsewhere than in the few specialized software programs. Moreover, although the 3D Tiles produced by Py3DTilers are aligned with the standard, some tools do not support all the features or extensions. Hence, work is being done in parallel on iTowns and UD-Viz in order to propose open source tools for visualizing 3D Tiles and their extensions.

A last problem is the management of additional semantic information. Although information can be contained in the Batch and Feature tables, the current 3D Tiles standard limits the storage of semantic data. Py3DTilers supports the *Batch Table Hierarchy* extension, which improves the management of semantic data, but the information can still only be associated with geometries and stored in tiles. It would be interesting to have a semantic associated with tilesets or groups of tiles, by supporting the *3DTiles_metadata* extension[26] or 3D Tiles Next[27].

5. CONCLUSION

Py3DTilers is a robust tool for creating 3D Tiles that conform to the specification. This ensures that the tilesets produced can be used by any 3D Tiles visualization or manipulation software. Py3DTilers differs from other 3D Tiles production tools by its flexibility: it offers a large number of transformation and data distribution options. Moreover, Py3DTilers is able to create 3D models from several different data formats, while abstracting from the specificities of each format. This allows to manipulate 3D models from different sources in the same context. Also, by using Batch and Feature tables, it allows to keep semantic data of each model. Finally, Py3DTilers being an open source tool, it is possible for everyone to enrich it. The code architecture allows support for new data formats by developing only the reading and triangulation of the geometry of the source data. The transformation into 3D Tiles is common and can be used without code modifications. It is also possible to easily integrate extensions to specialize the produced 3D Tiles. All of these makes it a tool that offers great versatility. It allows the user to have total control over the 3D Tiles creation process, either through options or through code modification. Py3DTilers allows the user to customize the tilesets produced, to test new ways of distributing tiles or creating levels of detail. Using this tool may help to innovate or experiment around 3D Tiles, and propose improvements to the standard.

Future work is planned to improve the rendering of 3D Tiles for city-scale tilesets. First of all, the compression of geometries and textures would drastically reduce RAM consumption. The integration of progressive levels of detail and textures would make it possible to refine the 3D Tiles much more fluidly, for example, as the camera approaches the 3D models or according to a context set by the user. It should also be possible to generate geometric errors that automatically adapt to the extent of the tile and its number of levels of detail. In addition, work is needed to balance the memory weight of the tiles and to provide more options for the distribution of *features* in the tiles. Furthermore, the next evolution of Py3DTilers is the use of style sheets as proposed by the OGC[28]. This would allow to apply a style to the *features* of a tile via separate files, exploiting the properties of each of the *features*. Finally, parallel work is underway to provide correction and validation of geometries. This will ensure that the 3D models are correct and that their normals are well oriented. The project also follows closely the evolution towards 3D Tiles Next announced in 2021.

[22] https://github.com/VCityTeam/Py3DTilers-docker
[23] https://github.com/VCityTeam/UD-Reproducibility/tree/master/Articles/2022_Py3DTilers
[24] https://github.com/VCityTeam/UD-Sample-data/tree/master/3DTiles
[25] https://py3dtilers-demo.vcityliris.data.alpha.grandlyon.com/
[26] https://github.com/CesiumGS/3d-tiles/tree/main/extensions/3DTILES_metadata
[27] https://cesium.com/blog/2021/11/10/introducing-3d-tiles-next/
[28] https://www.ogc.org/standards/se

ACKNOWLEDGEMENTS

The authors would like to thank the TIGA project[29], funded by La Banque des Territoires and led by the metropolis of Lyon, and the Berger-Levrault[30] company who allowed the development of this open source tool in order to experiment around the 3D Tiles format.

This work was conducted as part of the VCity project of Liris. The authors would like to thanks the member of this project for the gratefull help in this work.

REFERENCES

Beetz, J., van Berlo, L., de Laat, R., 2010. Bimserver.org - an Open Source IFC model server. 9.

Chen, Y., Shooraj, E., Rajabifard, A., Sabri, S., 2018. From IFC to 3D Tiles: An Integrated Open-Source Solution for Visualising BIMs on Cesium. *ISPRS International Journal of Geo-Information*, 7(10), 393. http://www.mdpi.com/2220-9964/7/10/393.

Colin, C., Samuel, J., Servigne, S., Bortolaso, C., Gesquière, G., 2022. Creating contextual view of CMMS assets using geospatial 2D-3D data.

Gaillard, J., Peytavie, A., Gesquière, G., 2018. Visualisation and personalisation of multi-representations city models. *International Journal of Digital Earth*, 1-18. https://hal.archives-ouvertes.fr/hal-01946770.

Gautier, C., Delanoy, J., Gesquière, G., 2022. Integrating multimedia documents in 3D city models for a better understanding of territories.

Hijazi, I. H., Krauth, T., Donaubauer, A., Kolbe, T., 2020. 3DCITYDB4BIM: a System Architecture for Linking Bim Server and 3d Citydb for Bim-Gis. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, V-4-2020, 195–202. https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/V-4-2020/195/2020/.

Hor, A.-H., Sohn, G., Claudio, P., Jadidi, M., Afnan, A., 2018. A semantic graph database for bim-gis integrated information model for an intelligent urban mobility web application. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-4, Copernicus GmbH, 89–96. ISSN: 2194-9042.

Jaillot, V., Pedrinis, F., Servigne, S., Gesquière, G., 2017. A generic approach for sunlight and shadow impact computation on large city models. *25th International Conference on Computer Graphics, Visualization and Computer Vision 2017*, Proceedings of WSCG2017, 25th International Conference on Computer Graphics, Visualization and Computer Vision 2017, Pilsen, Czech Republic, 10 pages.

Jaillot, V., Servigne, S., Gesquière, G., 2020. Delivering time-evolving 3D city models for web visualization. *International Journal of Geographical Information Science*, 34(10), 2030–2052. https://www.tandfonline.com/doi/full/10.1080/13658816.2020.1749637.

Kolbe, T., Gröger, G., Plümer, L., 2005. CityGML - Interoperable access to 3D city models. *Geo-information for Disaster Management*.

Kulawiak, M., Kulawiak, M., 2017. Application of Web-GIS for Dissemination and 3D Visualization of Large-Volume LiDAR Data. I. Ivan, A. Singleton, J. Horák, T. Inspektor (eds), *The Rise of Big Spatial Data*, Lecture Notes in Geoinformation and Cartography, Springer International Publishing, Cham, 1–12.

Mao, B., Ban, Y., Laumert, B., 2020. Dynamic Online 3D Visualization Framework for Real-Time Energy Simulation Based on 3D Tiles. *ISPRS International Journal of Geo-Information*, 9(3), 166. https://www.mdpi.com/2220-9964/9/3/166.

Schilling, A., Bolling, J., Nagel, C., 2016. Using glTF for streaming CityGML 3D city models. *Proceedings of the 21st International Conference on Web3D Technology*, ACM, Anaheim California, 109–116.

Xu, Z., Zhang, L., Li, H., Lin, Y.-H., Yin, S., 2020. Combining IFC and 3D tiles to create 3D visualization for building information modeling. *Automation in Construction*, 109, 102995. https://linkinghub.elsevier.com/retrieve/pii/S0926580519304285.

Yao, Z., Nagel, C., Kunde, F., Hudra, G., Willkomm, P., Donaubauer, A., Adolphi, T., Kolbe, T. H., 2018. 3DCityDB - a 3D geodatabase solution for the management, analysis, and visualization of semantic 3D city models based on CityGML. *Open Geospatial Data, Software and Standards*, 3(1), 5. https://opengeospatialdata.springeropen.com/articles/10.1186/s40965-018-0046-7.

Zhan, W., Chen, Y., Chen, J., 2021. 3D Tiles-Based High-Efficiency Visualization Method for Complex BIM Models on the Web. *ISPRS International Journal of Geo-Information*, 10(7), 476. https://www.mdpi.com/2220-9964/10/7/476.

[29] https://www.tuba-lyon.com/projet/tiga-mediation-industrielle/
[30] https://www.research-bl.com/