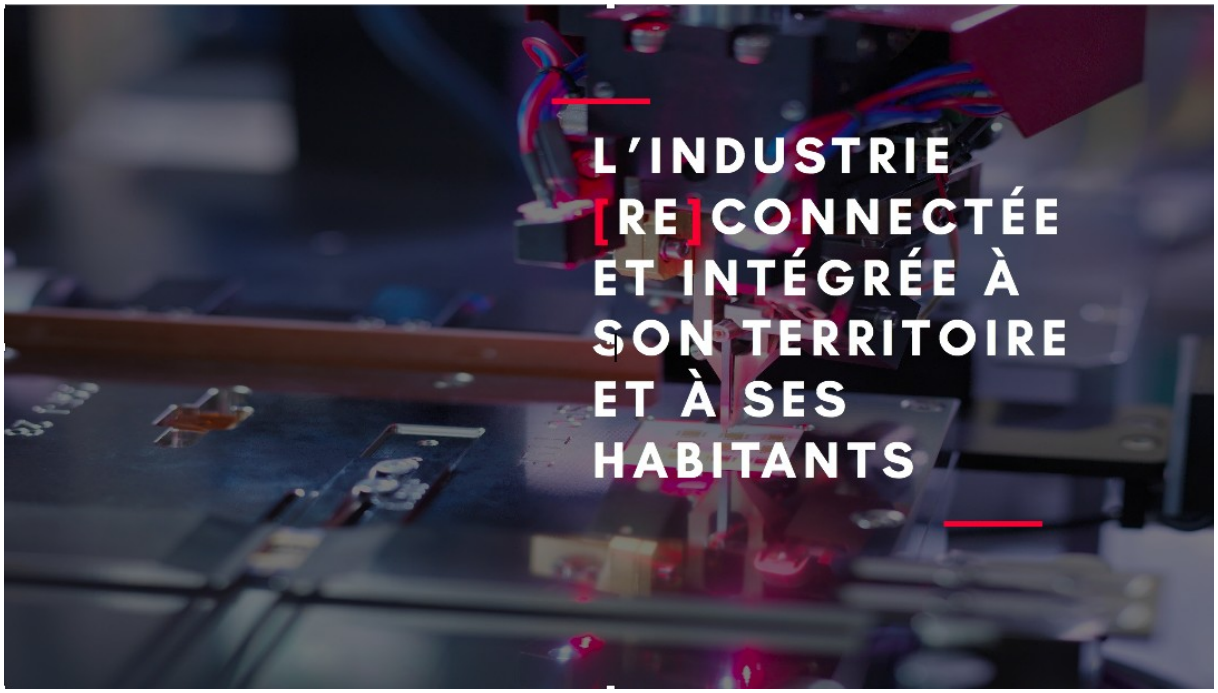



# TIGA - Action 14

## Think and Do Tank

## Sciences, Sociétés et Industrie

### Sous-action Médiation Augmentée

***« Développement des outils pour la médiation industrielle »***

Université de Lyon
Produit par Lorenzo Marnat, Gilles Gesquière et Corentin Gautier

Juin 2022

# Développement des outils pour la médiation industrielle

## Note aux lecteurs

Toutes les productions sont mises à disposition en libre accès. Le code a été libéré. Il est diffusé en Open Sources LGPL sur ces mêmes pages. Ce livrable est aussi proposé dans une version pdf contenant l'ensemble des informations. La première partie du document contient cette page décrivant le livrable 2022. Les contenus décrivant les composants les plus pertinents sont proposés en annexe du document pdf, mais aussi disponible sur github.

# Table des matières

## Introduction

La première étape de ce travail a consisté à proposer une veille des outils de médiation. Dans le premier livrable, des expérimentations inspirantes pour le projet TIGA ont été listées et analysées. Ce deuxième livrable est dédié à des propositions d'éléments de médiations liant numérique, territoire et jeu. Ces dispositifs sont pensés afin de favoriser l'interaction entre acteurs. Ils sont aussi conçus comme un ensemble de briques composables et réutilisables pour d'autres cas d'utilisations ou d'autres territoires.

Proposer de nouveaux outils innovants ne peut se faire que par une recherche "tirée par l'aval"; il s'agit de définir avec les partenaires un ensemble de besoins méthodologiques, scientifiques et techniques qui permettent de répondre à des cas d'utilisations présents dans TIGA. Les propositions faites ci-dessous permettent de nourrir en particulier deux besoins. Le premier besoin est de pouvoir mieux appréhender ce qui se passe dans une zone industrielle, en faisant ressortir la ville productive, mais aussi la vie des usagers du même territtoire. Ce travail a été mené avec la mission vallée de la chimie, le centre de formation Interfora AFAIP et TUBA : Tube à expérimentation urbaine Lyon. Le deuxième besoin porte sur la nécessaire mobilisation de données (en particulier 3D) dans un ensemble d'outils que nous mettons à disposition de la métropole de Lyon, en particulier de leur Laboratoire des Usages. Il s'agit de proposer des briques permettant une meilleure appropriation des données urbaines afin d'en faciliter l'utilisation par les usagers. Ces briques logicielles, issues du laboratoire Liris, sont présentées ci-dessous. Elle sont ensuite illustrées grâce à des cas d'utilisation.

Pour répondre à ce besoin d'une meilleure compréhension du secteur industriel et du territoire par le citoyen, nous nous sommes donc orientés vers des représentations 3D numériques de la ville. La visualisation 3D d'un environnement nous permet de mieux l'appréhender et ainsi mieux discerner son organisation, ses activités, etc. Cette représentation 3D de la ville peut être améliorée en allant au-delà de la géométrie des bâtiments. Nous voulons apporter plus de couleur et de réalisme à cette représentation pour rendre l'utilisateur plus à l'aise dans sa déambulation. De plus, afin de contextualiser un quartier ou un immeuble, nous cherchons à le lier à des connaissances, en particulier des contenus multi-médias (photo, vidéo, texte, etc.) et apporter une nouvelle dimension la 3D, allant au délà d'une simple déambulation dans l'espace 3D. Pour ce faire, nous avons conçus différents outils utiles pour la conception de jumeaux numériques avec comme point d'entrée des données urbaines. Dans un premier temps, nous développerons les socles techniques qui ont permis la construction de représentations 3D tel que la géométrie des bâtiments. Puis dans un second temps, nous détaillerons les démonstrations produites grâce aux outils développés et qui ont pu être mis en pratique. Ces outils ont été pensés de manière reproductible avec une documentation en continue afin que ceux-ci puissent être

utilisables de manière autonome et également sous différentes thématiques. Le type de Licence LGPL facilite aussi la réutilisation des composants et outils développés.

## Développement

### *Modèles 3D*

L'intéraction avec des données urbaines nécessite de résoudre un certain nombre de problématiques, telles que la visualisation massive d'objets 3D (bâtiments, ponts, végétation, etc) et la liaison de ces objets avec de la donnée sémantique (entendons par sémantique les données additionnelles qui viennent enrichir les données géométriques 3D; il peut s'agir d'attributs ou de façon plus général de corpus documentaires liés). L'approche que nous proposons consiste à n'appuyer au maximum sur des standards issus de L'isso TC/ 211 et de l'Open Geospatial Consortium. Cela facilite l'utilisation de composants déjà existants, mais aussi la réutilisation de nos propres composants. En ce sens, nous avons par exemple fait le choix d'utiliser le standard 3D Tiles, décrit par Cesium et l'OGC. 3D Tiles a été pensé pour aider à la visualisation massive de contenu géospatial 3D. Ce standard permet de décrire un *tileset* : un arbre de tuiles 3D. Chaque tuile contient des modèles 3D auxquels sont associées des données sémantiques. Un tileset permet une organisation spatiale des tuiles, optimisée pour le rendu d'objets 3D urbains, notamment grâce à la hiérarchisation spatiale des objets, mais aussi grâce aux niveaux de détails. Les niveaux de détail permettent d'alterner entre des géométries plus ou moins détaillées en fonction des besoins, par exemple en affichant des modèles très simplifiés de loin (distance par rapport au point de vue) et détaillés lorsqu'on est proche.

Les tuiles peuvent avoir différents formats :

- Batched 3D Model (B3DM) : Modèles 3D hétérogènes.
- Instanced 3D Model (I3DM) : Instances de modèles 3D.
- Point Cloud (PNTS) : Nombre massif de points colorés.
- Composite : Mélange de différents formats.

Dans les outils développés dans le cadre de TIGA, nous utilisons principalement le format B3DM. Ce format décrit les géométries à l'aide du format glTF, libre et facilitant le transfert et rendu de modèles 3D sur le web. Chaque tuile contient un ensemble de modèles 3D représentant par exemple des bâtiments ou des arbres. Chaque modèle 3D peut-être associé à un ensemble d'informations sémantiques.

La méthode utilisée pour créer les 3D Tiles peut avoir un impact direct sur la visualisation des objets. Il est donc nécessaire de disposer d'outils permettant de tester différentes méthodes de tuilage afin d'optimiser le rendu et la visualisation des modèles 3D. Il y a aussi le besoin d'offrir à l'utilisateur des outils lui permettant de créer des 3D Tiles depuis

différentes données, en y ajoutant de la couleur, des textures et des niveaux de détail. C'est avec cet objectif qu'ont été développés Py3DTiles et Py3DTilers.

### *Py3DTiles*

Py3DTiles est une librairie libre permettant de produire des 3D Tiles. Elle offre un ensemble de fonctionnalités pour écrire des 3D Tiles depuis de la donnée. Originellement développé par Oslandia, et le Liris, cette bibliothèque a été enrichie et robustifiée au cours du projet TIGA.

Le premier objectif des modifications et ajouts apportés à Py3DTiles est de faciliter la production de modèles 3D Tiles et leur personnalisation. Pour cela, nous avons notamment ajouté le système de création d'extension : l'utilisateur peut étendre le standard afin d'y ajouter ses propres fonctionnalités. Nous avons aussi introduit dans la librairie la création de matériaux. Ces matériaux permettent de changer l'apparence des modèles 3D en y appliquant des couleurs et des textures.
Le second objectif du travail sur Py3DTiles est de s'assurer que les 3D Tiles produits avec cette librairie puissent être visualisés avec différents outils. Dans ce but, des développements ont été effectués afin de vérifier que les 3D Tiles soient fidèles au standard. Cela permet d'être certain que les 3D Tiles créés avec Py3DIiles puissent être manipulés par tous les autres outils suivant aussi le standard.

### *Py3DTilers*

Py3DTilers est un outil libre développé dans le cadre du projet TIGA. Il permet de créer des 3D Tiles depuis différents formats de données: OBJ, GeoJSON, IFC et CityGML. Ces formats de données sont utilisés pour représenter des données géographiques et urbaines, 2D ou 3D. Le support de ces différents formats permet de créer des 3D Tiles depuis un grand nombre de données, où chaque donnée peut être spécialisée pour représenter des objets de la ville en particulier. Par exemple, le standard CityGML permet de représenter les géométries des bâtiments et du relief. Le standard IFC: Industry foundation classes décrit quant à lui plus en détails l'intérieur des bâtiments.

Py3DTilers se base sur la librairie Py3DTiles, décrite précédemment, pour produire des 3D Tiles. Cet outil vient rajouter des *Tilers*, permettant chacun de transformer un format précis de données en 3D Tiles. Py3DTilers intègre également un grand nombre d'options et de fonctionnalités pour personnaliser la création, parmi lesquelles :

- **La création de niveaux de détails**: permet d'ajouter un ou plusieurs niveaux de détail aux géométries. Cela permet de fluidifier le rendu en affichant des modèles 3D plus ou moins détaillés.

- **La reprojection des données**: permet de modifier le système de projection utilisé pour les coordonnées. Cela permet par exemple de visualiser dans un même contexte des données ayant originellement des systèmes de projection différents. Cela permet aussi de s'adapter aux contraintes pouvant être imposées par les outils de visualisation de 3D Tiles.
- **La translation et mise à l'échelle des modèles 3D**: permet de transformer la donnée en changeant l'échelle ou en déplaçant les géométries. Cela permet notamment de corriger les erreurs de placement ou de taille des objets dans la scène. Ces fonctionnalités permettent aussi de placer de la donnée non géo-référencée dans un contexte géospatial, ou inversement de placer de la donnée géo-référencée à des coordonnées centrées autour de (0, 0, 0).
- **L'export en modèle OBJ**: permet d'exporter les géométries au format OBJ. Ce format peut être visualisé dans la majorité des outils, ce qui permet de rapidement vérifier l'apparence des géométries.
- **L'ajout de textures**: si la donnée d'entrée est texturée, l'utilisateur peut choisir de conserver ou non les textures. Lorsque les textures sont conservées, elles sont stockées dans des atlas de textures sous forme de fichiers JPEG.
- **La création de 3D Tiles colorés**: permet d'ajouter des couleurs aux 3D Tiles. Les couleurs peuvent être choisies par l'utilisateur. Les couleurs peuvent être appliquées en fonction d'attributs des modèles (par exemple hauteur du bâtiment) ou en fonction du type des objets (toit, mur, sol, etc).

Toutes les options de Py3DTilers permettent d'obtenir un outil offrant une grande polyvalence. De plus, l'architecture de code permet de rapidement ajouter des nouvelles fonctionnalités ou de personnaliser celles qui existent déjà. Py3DTilers permet à l'utilisateur un contrôle total du processus de création de 3D Tiles, que ce soit via les options ou via la modification du code. L'outil permet de personnaliser les 3D Tiles produits, de tester de nouvelles modalités de répartition des tuiles ou de création de niveaux de détail. Py3DTilers se veut l'outil idéal pour innover ou expérimenter autour des 3D Tiles, et proposer des améliorations du standard. Cet outil est maintenant à disposition de la communauté TIGA et permet une utilisation facilitée des modèles 3D représentant les territoires cibles de TIGA.

En complément de la documentation présente sur le dépôt github de Py3dtilers un article scientifique a été rédigé sur Py3dTilers et ses fonctionnalités dans le cadre d'une conférence internationale (Smart Data Smart Cities & 3D GeoInfo) qui regroupe des experts de l'analyse des villes, des SIG, des jumeaux numériques, des villes intelligentes et de la science des données (Article Py3DTilers). Cet article a été accepté pour une présentation dans la conférence internationale 3DGeoinfo et sera publié dans le journal international ISPR Annals.

Les 3D Tiles générés avec l'outil Py3DTilers peuvent être visualisés avec différents logiciels. Néanmoins, nous utilisons dans la majorité des cas le visualisateur UD-Viz, un logiciel libre qui a été en partie développé au cours du projet TIGA.

### *UD-Viz*

UD-Viz est une librairie JavaScript permettant de visualiser de la donnée urbaine et d'intéragir avec cette donnée via navigateur Web. UD-Viz se base sur iTowns, développé par l'IGN, le LIRIS étant partenaire de ces développements. iTowns permet de charger et afficher des couches de données géographiques ainsi que des modèles 3D Tiles.

Dans le cadre du projet TIGA, UD-Viz a été enrichi afin d'offrir à l'utilisateur de nouvelles interactions avec la donnée ainsi qu'un meilleur contrôle sur l'apparence des modèles 3D. Pour cela, des développements ont été effectués afin de permettre l'intégration de contenus multi-médias et de couches de données 2D. Ces contenus permettent d'enrichir la visualisation et d'améliorer sa compréhension. De plus, le système de gestion du style des 3D Tiles a été revu afin de permettre une plus grande liberté sur l'affichage des couleurs et des textures des modèles 3D. Cela permet par exemple d'afficher des bâtiments colorés en fonction d'un attribut (hauteur, pollution, densité, etc) ou d'afficher alternativement des modèles colorés et des modèles texturés afin d'offrir plusieurs modalités de visualisation.

#### Couleurs et textures

UD-Viz intègre maintenant une gestion des styles permettant d'appliquer des couleurs aux modèles des 3D Tiles. Ces styles peuvent être appliqués modèle par modèle ou à un ensemble de modèles. Auparavant, un style arbitraire était appliqué par défaut à tous les 3D Tiles lors du chargement. Le problème avec ce style par défaut est qu'il détruit les styles déjà présents dans les modèles 3D, pour appliquer un style arbitraire à l'intégralité des modèles. En conséquence, il n'était pas possible de charger des 3D Tiles avec des textures ou des couleurs, car elles étaient détruites dès le chargement.

Des modifications ont été apportées à cette gestion des styles afin de laisser à l'utilisateur le choix entre:

- Appliquer un style par défaut aux 3D Tiles lors du chargement (comme auparavant).
- Conserver le style déjà présent dans les 3D Tiles et ne pas appliquer de style par défaut.

La deuxième option permet une plus grande diversité des styles, puisque ces derniers ne sont plus limités à une couleur unique. De plus, les différentes options de Py3DTilers, l'outil permettant de créer les 3D Tiles, offre beaucoup d'options pour ajouter des couleurs et des textures aux 3D Tiles. Ces couleurs et textures peuvent maintenant être chargées dans UD-Viz.

En plus de ce travail sur le style par défaut des 3D Tiles, des modifications ont été apportées à la gestion du style dans UD-Viz afin de corriger des erreurs et de robustifier le code.

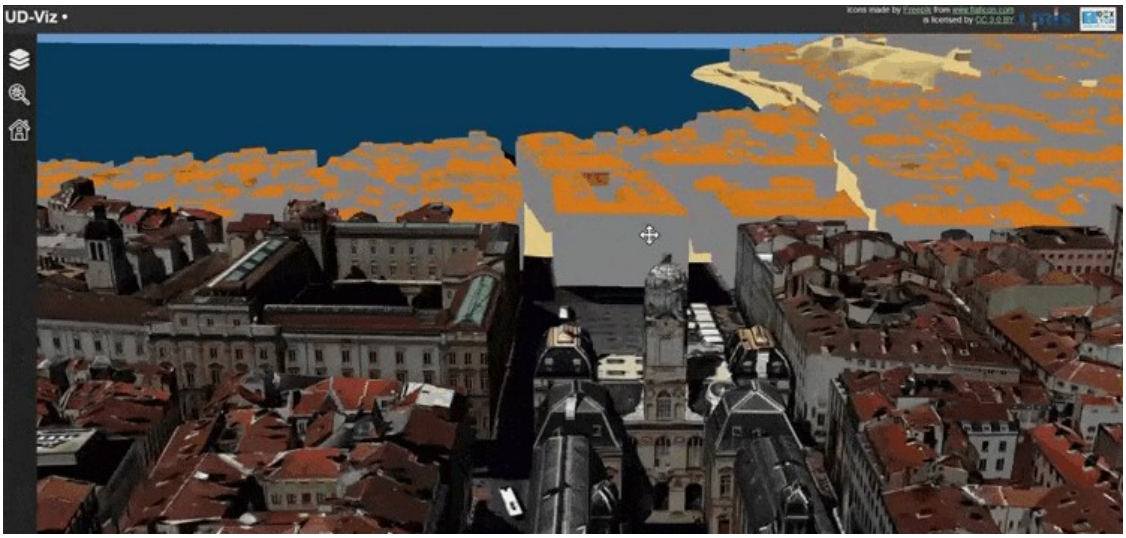



*Figure 1 : UD-viz démonstration, application des textures sur 1er arrondissement de Lyon*

### Intégration de contenus multi-médias

Afin d'améliorer la compréhension d'un territoire, nous voulions contextualiser les modèles 3D de bâtiments avec des données multi-médias additionnelles. Ces multi-médias peuvent être des images d'archives, des vidéos d'acteurs du territoire ou des photos d'un observatoire photographique(voir par exemple le travail sur l'Observatoire photographique des paysages de la Vallée de la chimie mené par la mission vallée de la chimie. Ce nouveau contenu apporte une autre dimension à la déambulation dans un environnement 3D et l'améliore avec plus d'informations sur celui-ci. Nous avons donc développé une méthode d'intégration de multi-medias qui se base sur la libraire UD-viz. (Annexe 5) L'intégration est découpée en deux parties :

- Configuration du fichier JSON : documentation sur la configuration du fichier JSON afin de lier contenus multi-medias et positions géospatiales.
- Episode visualizer object : programme d'intégration des multi-médias avec comme point d'entrée le fichier de configuration JSON.

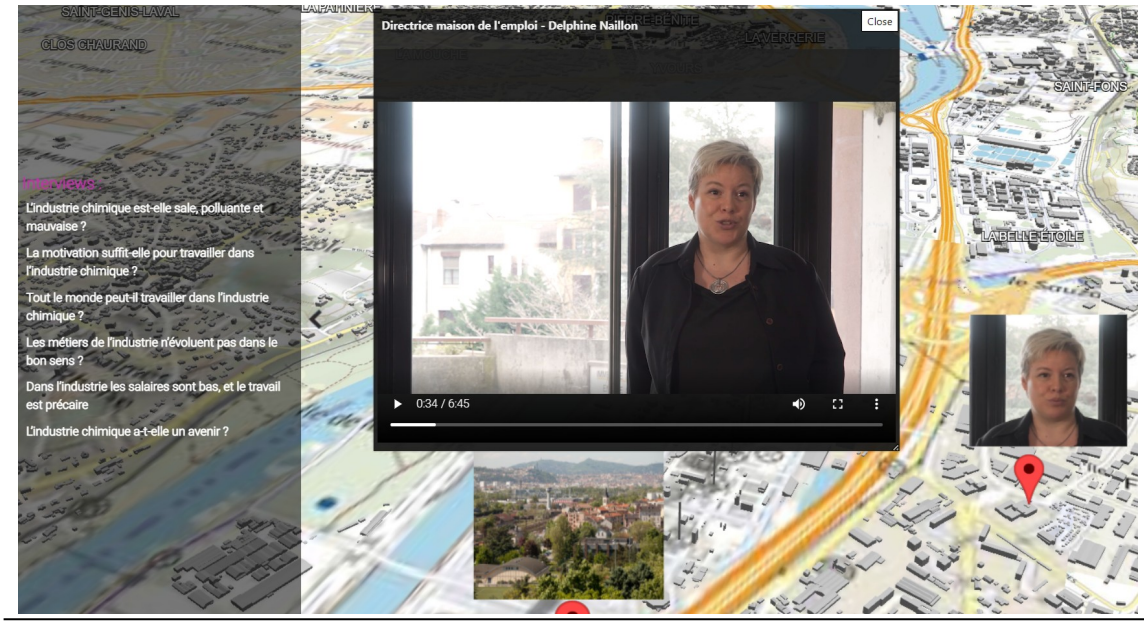



*Figure 2 : "Derrière les fumées", intégration d'une interview de la directrice de la maison de l'emploi dans la représentation numérique de la vallée de la chimie*

Cette approche a pu être utilisée dans le contexte du web-documentaire du projet "Derrière les fumées" (voir ci-après). Des interviews d'acteurs de la vallée de la chimie ont été disposées dans la représentation numérique de cette zone afin d'avoir plus d'informations sur l'activité de ce territoire.
Ce socle technique est documenté dans la librairie UD-Viz (Annexe 5) ainsi que dans un article scientifique sur l'intégration de multi-média dans une représentation 3D numérique soumis à la même conférence internationale que Py3dTilers, Smart Data Smart Cities & 3D GeoInfo (Article intégration de multimedia dans une représentation 3D numérique). Cet article a été accepté pour une présentation dans la conférence internationale 3DGeoinfo et sera publié dans le journal international ISPR Annals.

### Intégration de couches de données 2D

Dans cette idée d'une meilleure compréhension, au-delà d'une représentation 3D, nous nous sommes intéressés à une visualisation 2D du territoire. En effet, les données 2D urbaines apportent une nouvelle vision sur un territoire et donnent plus d'informations sur celui-ci, comme l'accessibilité de certains quartiers grâce au réseau de transport en commun ou bien les zones végétalisées d'un arrondissement.

L'intégration de couches de données se fait via des services WFS (Web Feature Service) et WMS (Web Map service) dans la bibliothèque UD-viz (Annexe 5). Ce type de données nous permet de représenter différentes géométries dans UD-Viz; cela peut être des polylignes,

des polygones ou des images PNG/JPEG. Cet ajout de couches de données apporte une nouvelle vision à la représentation numérique et permet de contextualiser les modèles 3D de bâtiments.

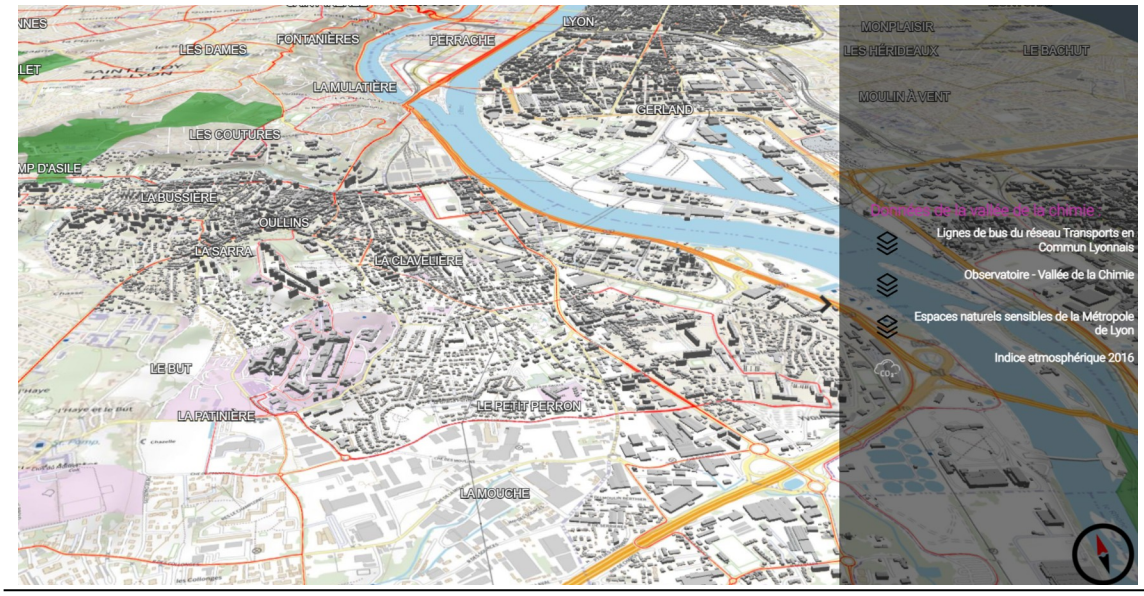



*Figure 3 : "Derrière les fumées": intégration de données urbaines 2D tel que lignes de transports en communs (lignes rouges) et les espaces naturels sensibles de la métropole de Lyon (polygones verts) dans la représentation numérique de la vallée de la chimie*

Un cas d'exemple de cette intégration peut être retrouvé dans la démo vallée de la chimie avec un jeu de données 2D disponible sur le site open data Grand Lyon. Nous avons intégré le réseau de transport en commun pour montrer l'accessibilité à certains sites industriels de la vallée de la chimie. Nous avons également intégré les espaces naturels sensibles pour donner une autre vision de ce territoire.

### *Démos UD-Viz*

UD-Viz-Template est un template d'application permettant de créer rapidement des démonstrations basées sur la librairie UD-Viz (Annexe 5). Ces démonstrations permettent d'illustrer des fonctionnalités et/ou des données particulières. Grâce à ce template, différentes démonstrations ont pu être réalisées afin d'alimenter le projet TIGA. En voici la liste présentée ci-dessous :

#### Démo Py3DTilers

- Code source de la démonstration ainsi que sa documentation.
- Docker pour reproduire l'application.

- Démo en ligne

Cette démo propose un ensemble de 3D Tiles créés avec les outils de Py3DTilers. On y retrouve des 3D Tiles de bâtiments, ponts, relief et cours d'eau, certains texturés et d'autres colorés.

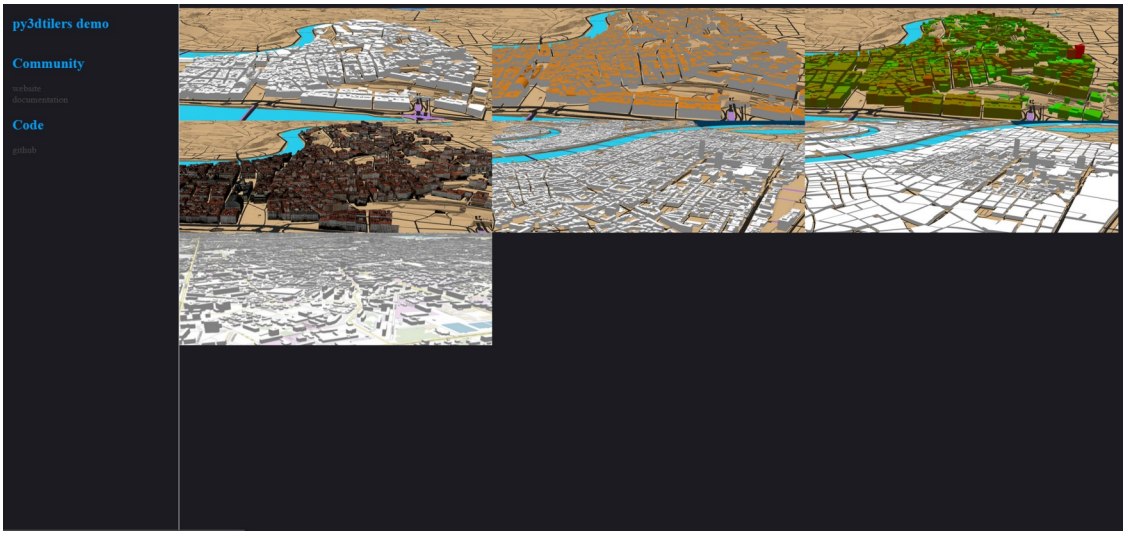



*Figure 4 : Mosaïques des différentes démonstrations développées grâce à l'outil Py3tilers. Une démonstration de la ville texturée, une autre avec de la couleur, la ville avec différents niveaux de détail..*

Les 3D Tiles ont été créés à partir de couches de données publiques issues du site du Grand Lyon et de l'IGN. Les modèles 3D des ponts et du relief sont créés à partir de la donnée CityGML du Grand Lyon, par l'intermédiaire d'une base de données 3DCityDB. Les fleuves et les routes sont créés à partir des données GeoJSON de l'IGN. Les bâtiments sont quant à eux créés soit à partir des fichiers CityGML soit à partir de fichiers GeoJSON. Une documentation a été créée pour expliquer plus en détails le processus de création des 3D Tiles depuis de la donnée publique.

La démo introduit aussi de nouvelles modalités de visualisation des 3D Tiles. Elle implémente notamment un nouveau fonctionnement des niveaux de détails des modèles 3D. Par défaut, le niveau de détails se raffine en fonction du zoom (position par rapport à la caméra) : plus la caméra est proche d'un modèle, plus ce dernier est détaillé. Ici, on offre la possibilité d'utiliser la souris comme une "loupe": les modèles proches de la souris de l'utilisateur sont raffinés afin d'obtenir des modèles plus détaillés sans avoir besoin de bouger la caméra.

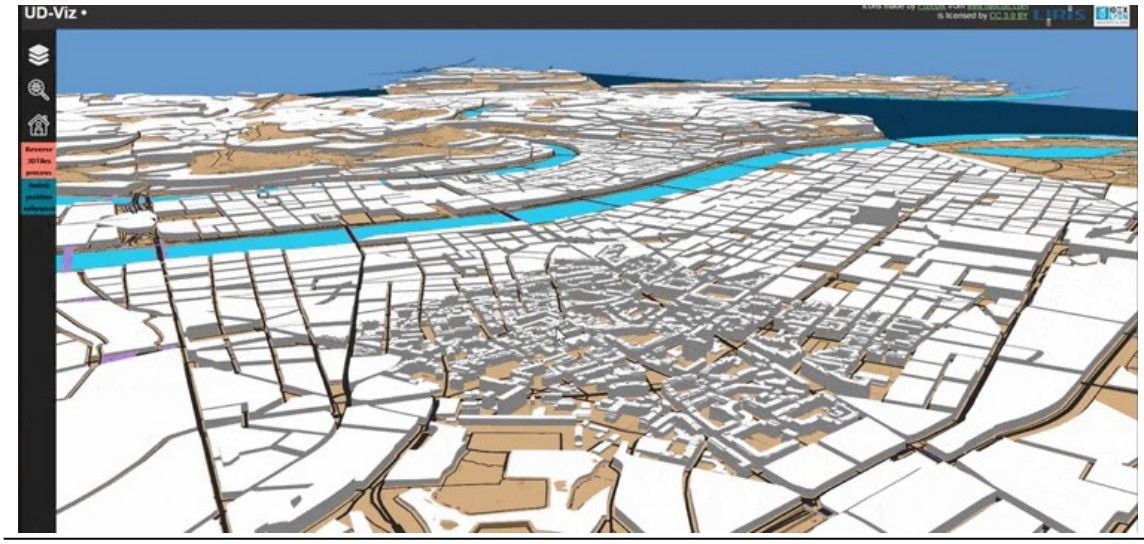



*Figure 5 : Démonstration montrant le raffinement du niveau de détail des bâtiments de Lyon en fonction de la position de la souris*

### Démo UI-driven

- Code source de la démonstration ainsi que sa documentation.
- Docker pour reproduire l'application.
- Démo en ligne

Cette démo permet de calculer la hauteur de routes en les plaçant sur le relief. Pour cela, les routes doivent être contenues dans des fichiers GeoJSON. Ces fichiers peuvent ensuite être glissés/déposés dans la démonstration. Les routes seront affichées au fur et à mesure du calcul. Une fois toutes les routes placées sur le relief, de nouveaux fichiers GeoJSON contenant les routes aux bonnes altitudes sont téléchargés. Le processus pour créer des routes 3D à partir de donnée de l'IGN est détaillé dans cette documentation.

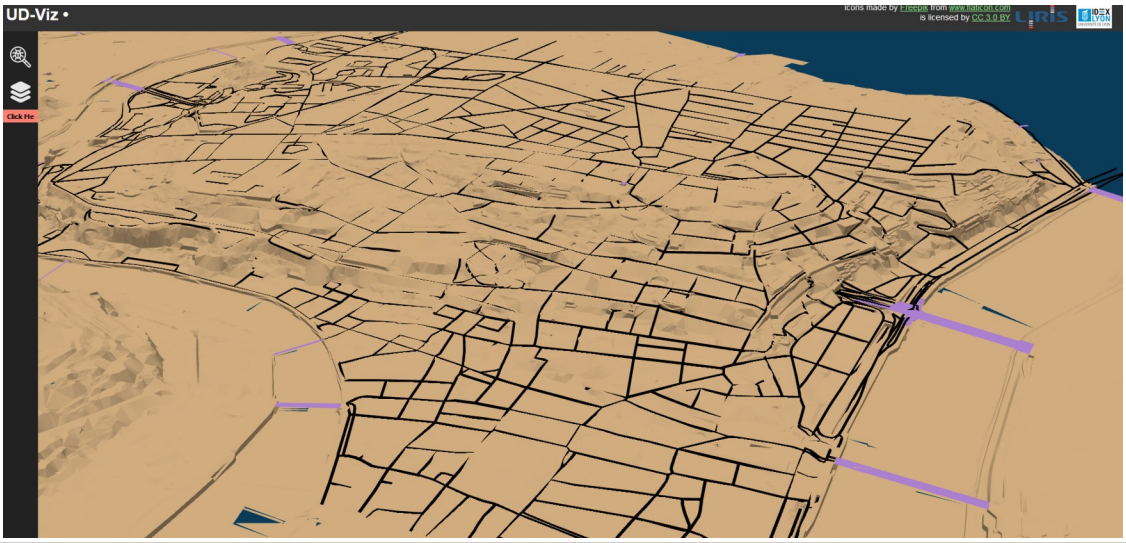



*Figure 6 : Démo UD-Viz sur le calcul des hauteurs de routes contenu dans un fichier GeoJSON à l'aide d'un glisser/déposer dans l'application web UD-Viz*

### *Démo Vallée de la chimie*

Une expérimentation de ces outils aidant à la représentation numérique d'un territoire et la démo Vallée de la chimie qui a été développé dans le cadre d'un web-documentaire en collaboration avec la missions vallée de la chimie, le centre de formation Interfora AFAIP et TUBA : Tube à expérimentation urbaine Lyon. Ce projet "Derrière les fumées" a pour but de déconstruire les idées préconçues que peuvent avoir les citoyens de la métropole de Lyon sur ce territoire. Dans cette problématique, nous nous sommes orientés vers une représentation numérique de la vallée de la chimie pour mieux la comprendre et y disposer du contenu multi-média afin d'apporter plus d'informations. Grâce à l'outil d'intégration de multi-média dans la bibliothèque UD-Viz (Annexe 5), nous avons pu disposer des d'interviews d'acteurs de ce territoire et des données urbaines dans la maquette numérique. Ces différents éléments sont disposés à des points d'intérêt en lien avec le thème de l'industrie.

- Code source de la démonstration ainsi que sa documentation.
- Docker pour reproduire l'application.
- Démo en ligne


*Figure 7 : Capture du web-documentaire "Derrière les fumées", avec les deux panneaux d'affichages, les différentes interviews d'acteurs de la vallée de la chimie et les données urbaines récupérées sur l'open data Grand Lyon*

Pour produire cette démonstration nous avons récupéré les données CityGML disponibles sur l'open data Grand Lyon pour ensuite les traiter avec Py3DTilers et générer les 3D Tiles (3D Tiles générés avec Py3DTilers : bâtiments, relief, routes, ponts, fleuves ) de la vallée de la chimie. En complément de ces modèles 3D, nous avons intégré les photos de l'observatoire de la vallée de la chimie produites par Florent PERROUD et les images d'archives de la catastrophe de Feyzin en libre accès sur le site la bibliothèque municipale de Lyon (lien des photos).

### *Docker*

Docker est un outil permettant de lancer des applications dans un contexte déterminé et isolé. Les applications ne sont ainsi pas exécutées directement sur la machine hôte, mais dans un contexte maîtrisé. Une application contenue dans un Docker sera toujours exécutée de la même manière, les versions des différents composants sont figées. Cela permet de s'assurer que l'application pourra être utilisée sur n'importe quelle machine, sans souci d'installation et sans erreur de version des logiciels. L'utilisation de Docker permet d'éviter qu'une application fonctionnelle ne devienne inutilisable après quelque temps à cause de mise à jour de la machine hôte ou de l'application elle-même.

C’est pourquoi toutes les applications développées lors du projet possèdent des versions contenues dans des Dockers. Ainsi, on s’assure de la pérennité dans le temps des applications en plus d’être certains qu’elles pourront être lancées sur toutes les machines.

- UD-Viz-Docker
- UD-Demo-TIGA-Webdoc-ChemistryValley-Docker
- UD-Demo-vcity-py3dtilers-lyon-Docker
- UD-Demo-VCity-UI-driven-data-computation-Lyon-Docker

Chaque docker listé est une application des différents outils développés dans le cadre du projet TIGA.

---

## Conclusion

Depuis le début du projet TIGA, l’action 14 “THINK & DO TANK Sciences, Sociétés et Industrie” permet de favoriser les échanges entre partenaires. Après un travail de veille qui a donné lieu au premier livrable en tout début de projet TIGA, et une phase d’échange au sein de cette action, il s’est agit de proposer un ensemble de composants permettant le développement de nouvelles modalités de médiation pour aider à la reconnexion entre Industrie, citoyen et territoire. La veille nous a permis de nous orienter vers ces premiers outils de médiation dans le but de rapprocher les différents acteurs du territoire de la métropole de Lyon.

Dans ce deuxième livrable, nous avons cherché à voir comment des représentations 3D pouvaient venir en interface entre le territoires et les usagers. Les différents outils ont permis la création de nouvelles représentations numériques 3D en y apportant une nouvelle dimension grâce aux multi-médias et ainsi de les éprouver sur des problématiques en lien avec l’industrie. Chaque application est développée grâce à un ensemble de composants atomiques mis à disposition des acteurs TIGA, mais aussi plus généralement d’une communauté plus large afin d’en assurer une meilleure dissémination. Au delà du code, il s’est agit de proposer une documentation liée, ainsi que des dockers afin que tout le monde puisse utiliser les composants sur son propre environnement de travail et répondre au besoin de reproductibilité. Les composants sont conçus de façon atomique afin de pouvoir les composer en fonction des besoins et assurer une meilleur réutilisabilité, mais aussi réplicabilité.

Ces composants ont été mis en place afin de proposer une nouvelle modalité d’accès aux contenus via le territoire dans le cadre du web doc “Derrière les fumées” mis en place en collaboration avec la missions vallée de la chimie, le centre de formation Interfora AFAIP et TUBA : Tube à expérimentation urbaine Lyon.

Une autre application de ces composants est faite sur les “maquettes tangibles”, élément d’hybridation entre maquette physique du territoire et contenu numérique. Elle sera décrite dans un prochain livrable. Grâce à l’intégration d’images d’archives dans un jumeau numérique de ville nous avons abordé l’évolution temporelle de la ville. Nous voulons aller plus loin et développer plus en profondeur ce côté évolutif pour mieux comprendre comment un territoire est devenu ce qu’il est actuellement. Ce projet Morphogenèse se veut focalisé sur l’évolution du travail de 1950 à aujourd’hui et est le prochain objectif du LIRIS dans le contexte du projet TIGA, en collaboration avec le laboratoire Environnement Ville et Société. Il viendra aussi compléter le prochain livrable que le LIRIS aura à produire dans les prochains mois dans le cadre du projet TIGA.

# Annexe 1

## Compute Lyon 3DTiles

This document explains how to create 3DTiles models of buildings, relief, roads, bridges and water bodies from Lyon's open data with py3dtilers.

To be able to use the Tilers from py3dtilers, follow the installation notes.

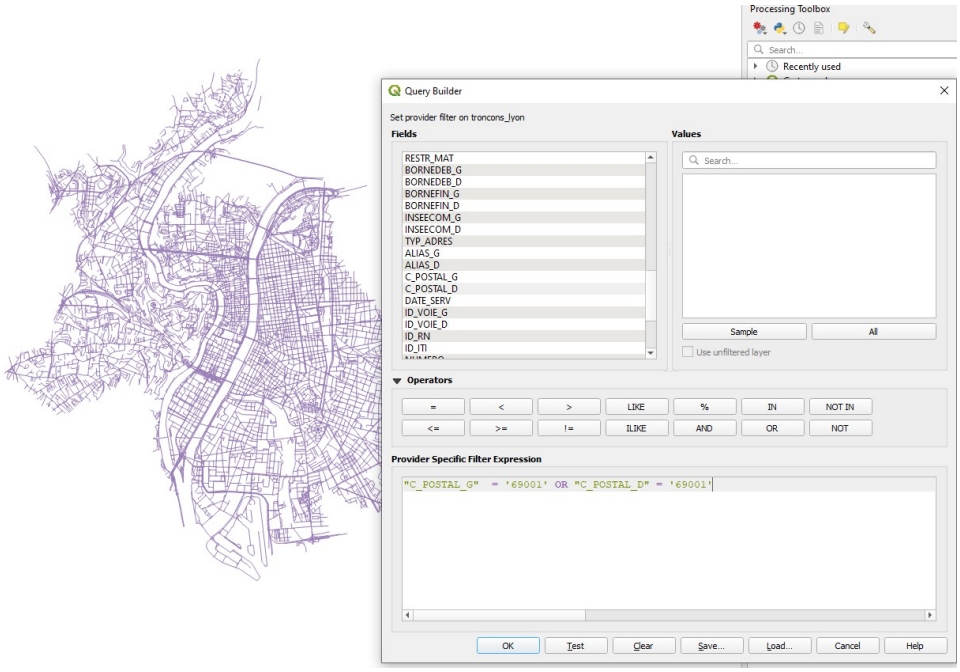
### Geojson Tiler

#### Roads

Download the BD Topo data from IGN

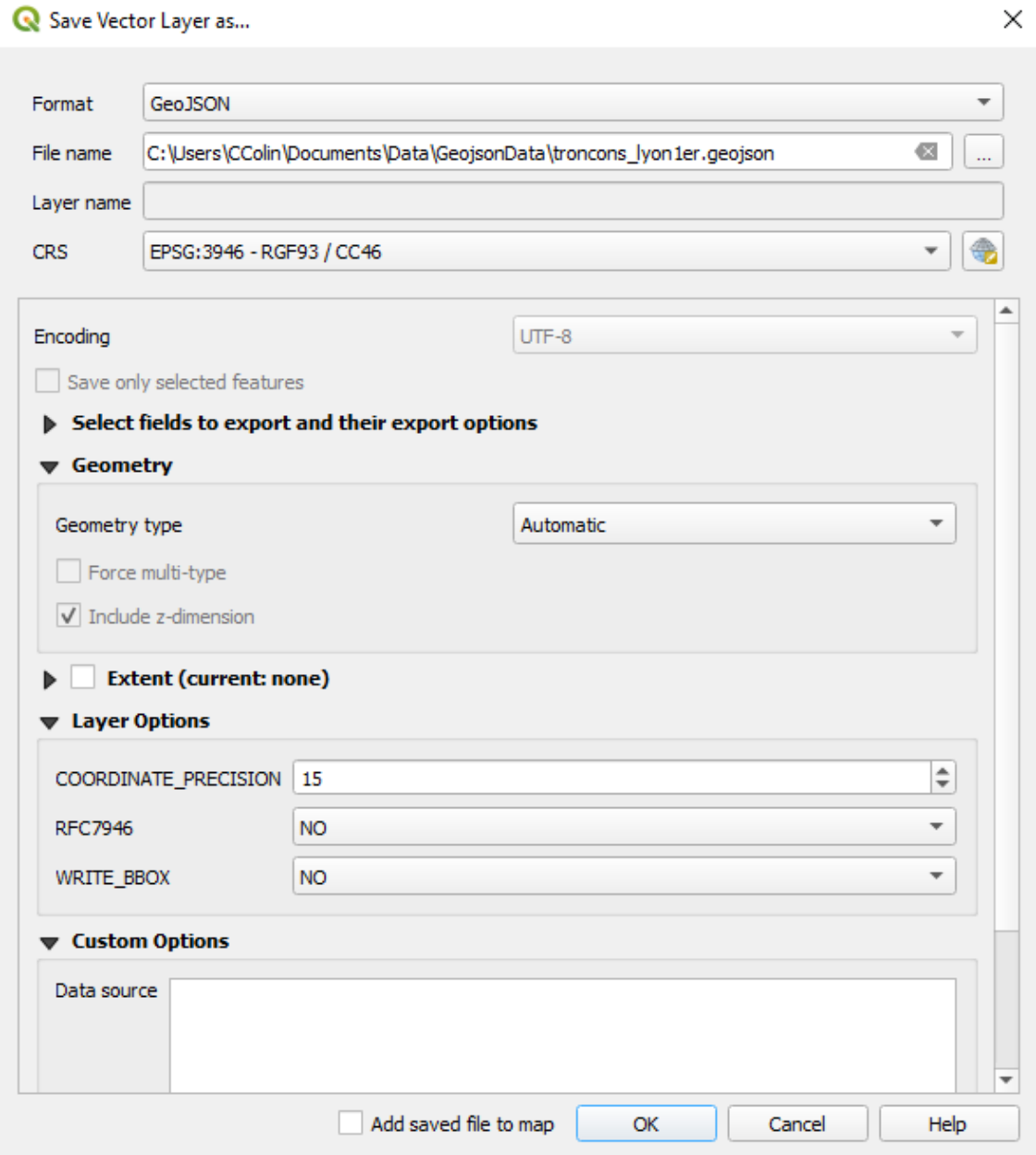
In QGIS, open the *BDTOPO/1_DONNEES_LIVRAISON/TRANSPORT/**TRONCON_DE_ROUTE.shp*** file.

You can filter roads, for example by keeping those starting or ending in Lyon 1er:




*qgis_filter_road*

Then, save the roads layer as a GeoJson file:




*qgis_save_layer*

To create roads 3DTiles with the GeoJsonTiler, run:

```
geojson-tiler --path path/to/troncons_lyon1er.geojson --height 0.5
```

The height argument set how thick are your roads (in meters). You can set it to an arbitrary value.
You can also set the width of your roads with `--width <float>`. By default, the GeoJsonTiler targets the property `LARGEUR` in geojson features to find the width; the property to target can be changed with `--width OTHER_PROP_NAME`.

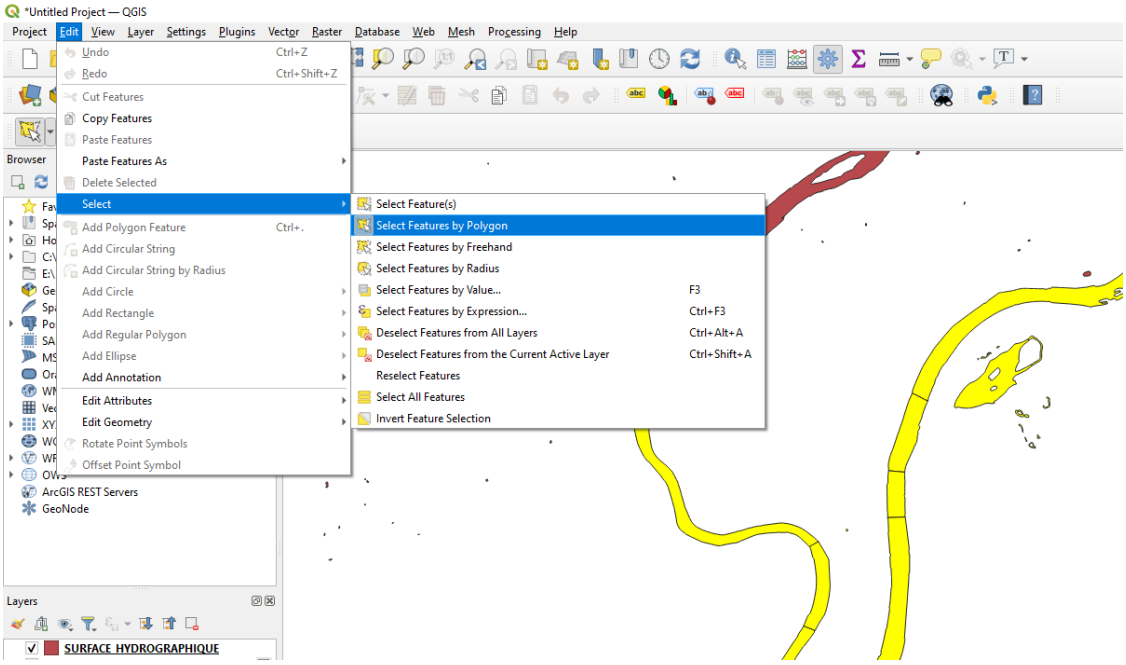
See the GeojsonTiler README for more information on usage.

### Water bodies

Download the BD Topo data from IGN

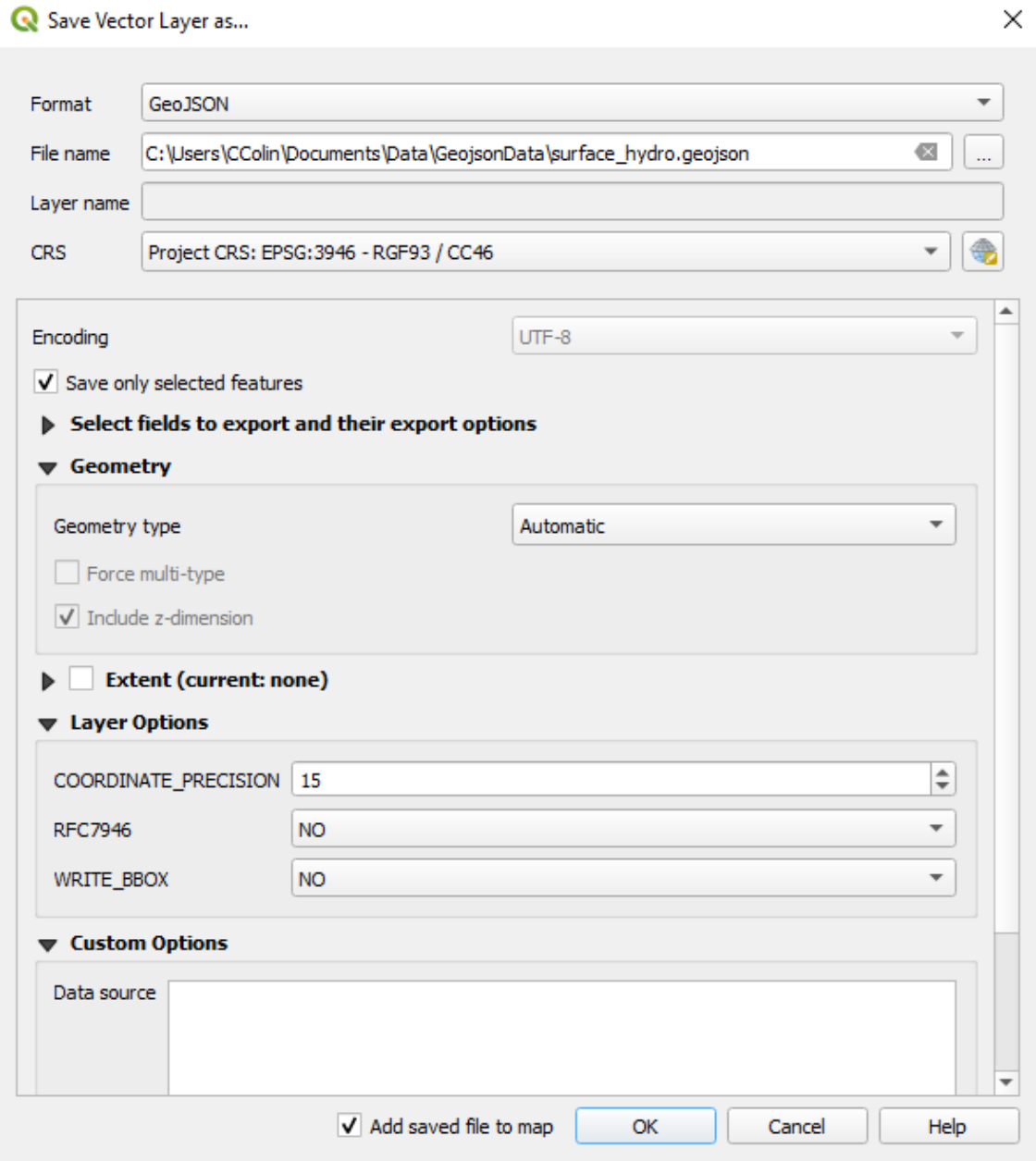
In QGIS, open the *BDTOPO/1_DONNEES_LIVRAISON/HYDROGRAPHIE/**SURFACE_HYDROGRAPHIQUE.shp*** file.

You can select only the parts you need:




*select_hydro_surface*

Then, save the roads layer as a GeoJson file:




*qgis_save_hydro*

To create 3DTiles with the GeoJsonTiler, run:

```
geojson-tiler --path path/to/surface_hydro.geojson --height 0.5
```

The height argument set how thick are your water bodies (in meters). You can set it to an arbitrary value.

See the GeojsonTiler README for more information on usage.

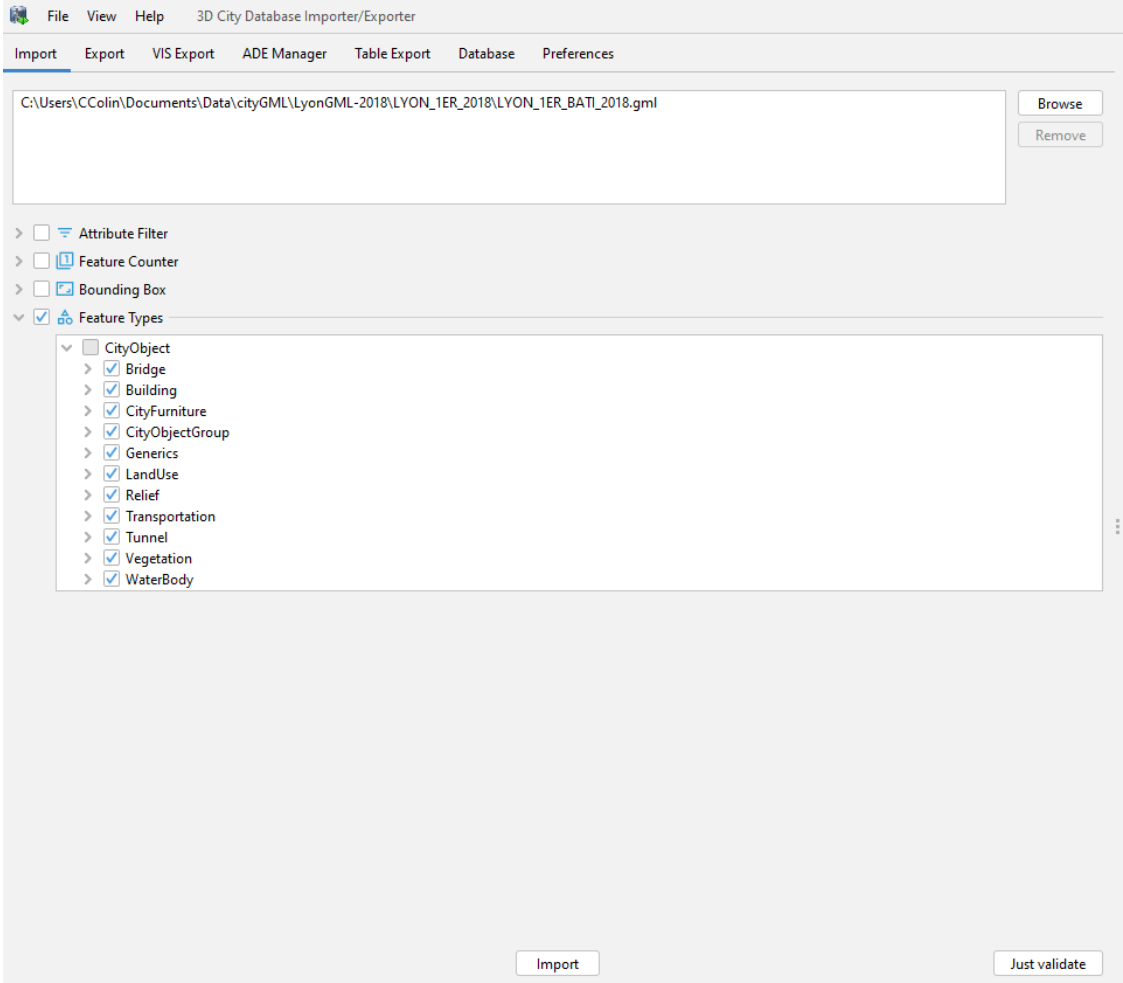
## CityGML Tiler

Creating 3DTiles with the CityGML Tiler require Postgres/PostGIS and 3DCityDB. The cityGML data must be hosted in a 3DCityDB database to be used by the CityGML Tiler. To host cityGML in database, you can follow **this tutorial** (recommended) or use the docker (may be outdated).

You should also copy the configuration file (for example `py3dtilers/CityTiler/CityTilerDBConfig.yml`) and add the details of your database in this new file.

### Buildings

Download the cityGML data from Data Grand Lyon (you can choose which districts of Lyon you want to download). Then, import the buildings into a 3DCityDB database:




*import_buildings*

To use the Tiler, target your database config file and choose the type `building` (see the CityTiler usage for more details):

```
citygml-tiler --db_config_path <path_to_file>/Config.yml --type building
```

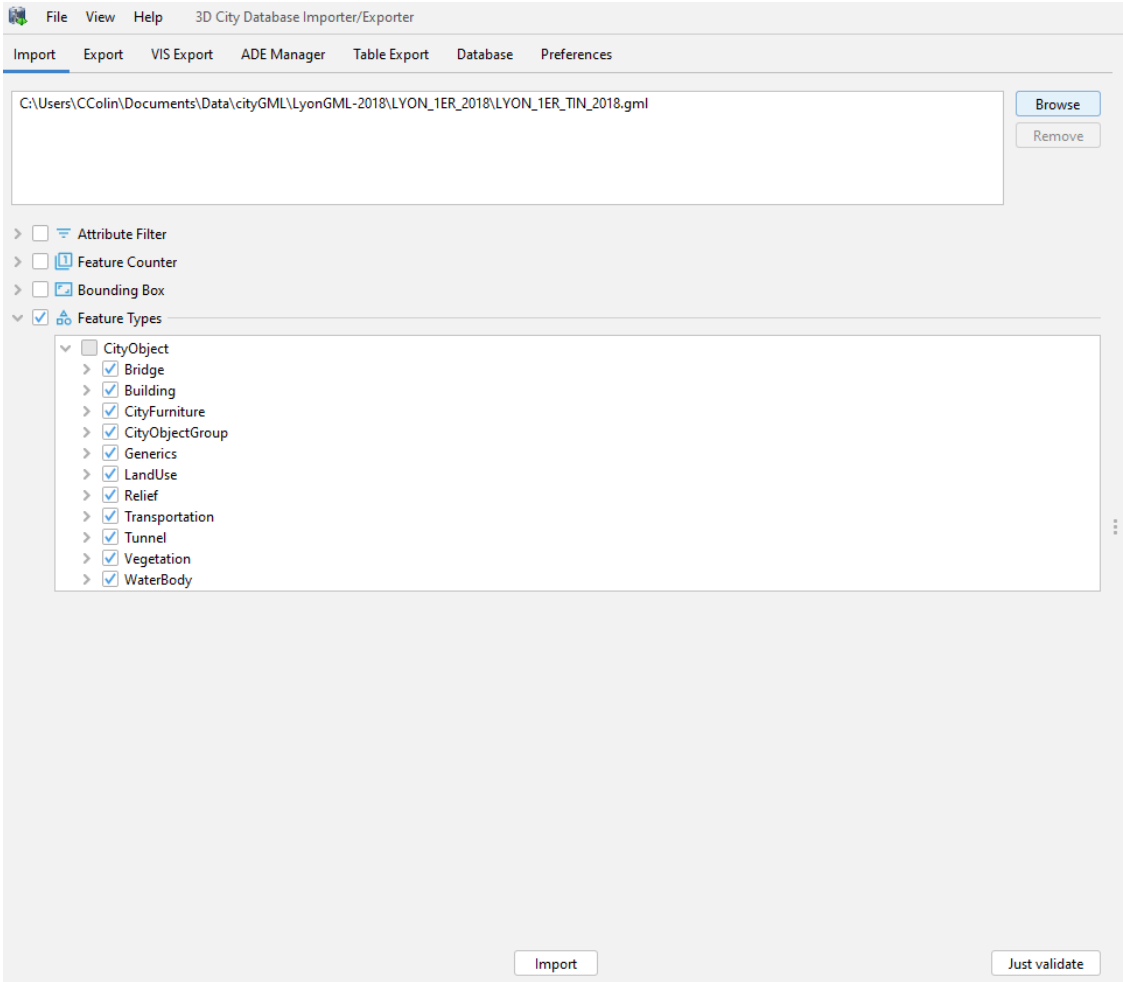
To create LOA, you can for example use *BDTOPO/1_DONNEES_LIVRAISON/ADMINISTRATIF/**ARRONDISSEMENT.shp*** from BD Topo (IGN). To be able to use it, export the .shp as GeoJson with QGIS (the projection must be the same as buildings, i.e EPSG:3946 most of the time for Lyon's data).

To create the 3DTiles with levels of detail, run:

```
citygml-tiler --db_config_path <path_to_file>/Config.yml --lod1 --loa
polygons.geojson
```

### Relief

Download the cityGML data from Data Grand Lyon (you can choose which districts of Lyon you want to download). Then, import the relief into a 3DCityDB database:




*import_relief*

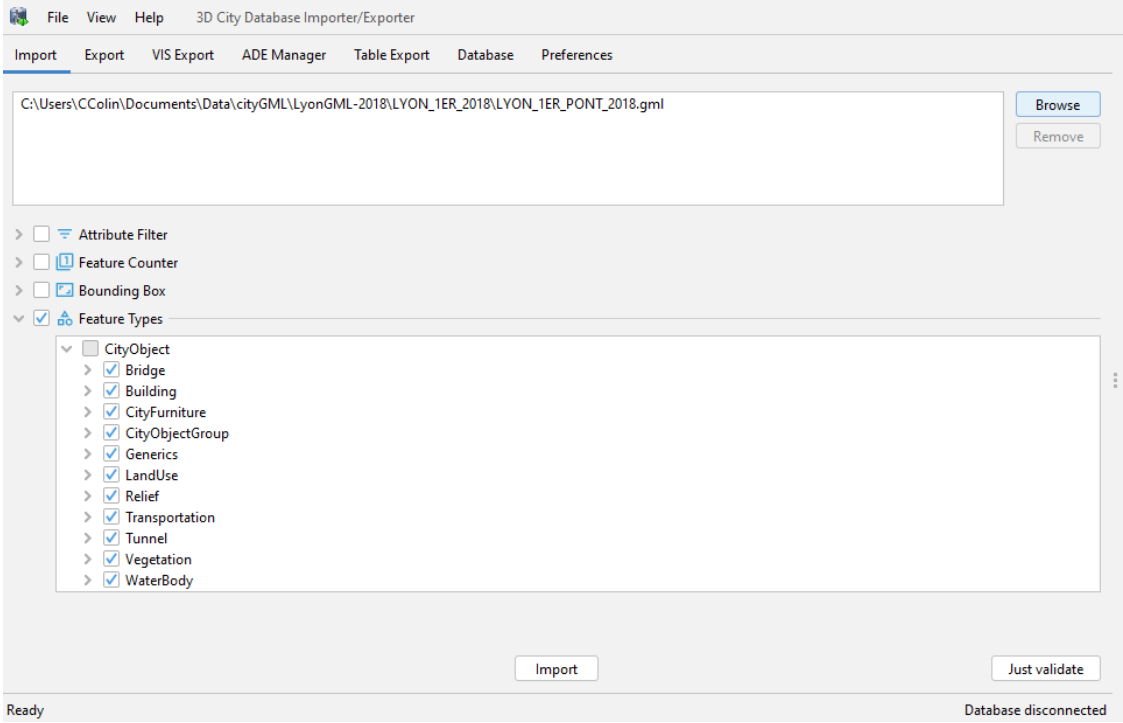
To use the Tiler, target your database config file and choose the type `relief` (see the CityTiler usage for more details):

To create the relief as 3DTiles, run:

```
citygml-tiler --db_config_path <path_to_file>/Config.yml --type relief
```

### Bridges

Download the cityGML data from Data Grand Lyon (you can choose which districts of Lyon you want to download). Then, import the bridges into a 3DCityDB database:




*import_bridges*

To use the Tiler, target your database config file and choose the type `bridge` (see the CityTiler usage for more details):

To create the bridges as 3DTiles, run:

```
citygml-tiler --db_config_path <path_to_file>/Config.yml --type bridge
```

# Annexes 2

## Pins documentation

Objet 3D interactif pointant sur des centres d'intérets dans la scène 3D de UD-Viz (Annexe 5). Ces éléments interactifs sont representés par une bulle d'image ainsi qu'une épingle ; l'image permet de donner un aperçu sur le contenu qui sera afficher si l'utilisateur intéragit avec en cliquant sur celui-ci.

Chacun de ces éléments est un contenu d'un épisode et un épisode comprends plusieurs contenus. Ce contenu permet de détailler une zone spécifique sur une maquette en y ajoutant du texte ou une image.

---

### Technical details

Un pin est un contenu d'un épisode qui est matérialisé par la classe `EpisodeContent.jset` contient toutes les informations sur un contenu d'un épisode. La classe `EpisodeVisualizer.js` comprend la liste des contenus de l'épisode et donc une liste d'objet `EpisodeContent` qui apparaissent dans la scène 3D.

---

*configEpisodes.json :*

Le configEpisodes.json est le fichier JSON qui permet de configurer tout le contenu de votre épisode. C'est ici que vous allez modifier les sources d'images, sa position etc… et doit suivre le template suivant :

- `lock` : L'élément 3D peut être vérrouillé ou dévérouillé en fonction de sa valeur dans le fichier de config. Cette valeur dépend d'ou se trouve l'utilisateur dans le visionnage d'un épisode. Le contenu pourra être accessible que si celui-ci a une valeur à **false**
- `position` : la position georéférence dans votre scène avec son type de projection (EPSG:3946).
- `imgUnLock` : le chemin en direction votre image dévérouillé.
- `imgLock` : le chemin en direction votre image vérouillé.

Une fois dévérouillé, l'épingle est accessible et intéractive pour afficher le contenu de ce point d'intéret plus en détail. Une fenêtre html apparait et donne une description de la zone que pointe l'élément.

---

*Code example*

Exemple de code pour créer 3 contenu d'un épisode et l'afficher dans la scène.

```
    let content_1 = new EpisodeContent(configEpisode['episode-1-data']
['content-1']);
    let content_2 = new EpisodeContent(configEpisode['episode-1-data']
['content-2']);
    let content_3 = new EpisodeContent(configEpisode['episode-1-data']
['content-3']);
    let listContents = [content_1,content_2,content_3];

    const episode_1 = new EpisodeVisualizer('episode_1', view3D,
listContents);
    episode_1.constructAllContent();
```

# Annexe 3

## Python 3DTiles Tilers

p3dtilers is a Python tool and library allowing to build `3D Tiles` tilesets out of various geometrical formats e.g. OBJ, GeoJSON, IFC or CityGML through 3dCityDB databases.

p3dtilers uses `py3dtiles` python library (forked from Oslandia's py3dtiles) for its in memory representation of tilesets.

py3dtilers can only produce `Batched 3D Models (B3DM)`. If you want to produce `Point Clouds (PNTS)`, see Oslandia's py3dtiles CLI.

### CLI Features

- ObjTiler: converts OBJ files to a 3D Tiles tileset
- GeojsonTiler: converts GeoJson files to a 3D Tiles tileset
- IfcTiler: converts IFC files to a 3D Tiles tileset
- CityTiler: converts CityGML features (e.g buildings, water bodies, terrain…) extracted from a 3dCityDB database to a 3D Tiles tileset
- TilesetReader: read, merge or transform 3DTiles tilesets

### Installation from sources

#### For Unix

Install binary sub-dependencies with your platform package installer e.g. for Ubuntu use

```
apt-get install -y libpq-dev       # required usage of psycopg2 within
py3dtilers
apt install python3 python3-pip    # Python3 version must be <=3.9
```

First create a safe python virtual environment (not mandatory yet quite recommended)

```
apt install virtualenv
virtualenv -p python3 venv
. venv/bin/activate
(venv)$
```

Then, depending on your use case, proceed with the installation of `py3dtilers` per se

- **py3dtilers usage use case**:
  Point pip directly to github (sources) repository with

- `(venv)$ pip install git+https://github.com/VCityTeam/py3dtilers.git`

- **py3dtilers developers use case**:
  Download py3dtilers sources on your host and install them with pip:

- ```
  apt install git
  git clone https://github.com/VCityTeam/py3dtilers
  cd py3dtilers
  (venv)$ pip install -e .
  ```

### For Windows

Install python3 (**<= 3.9**).

In order to install py3dtilers from sources use:

```
git clone https://github.com/VCityTeam/py3dtilers
cd py3dtilers
python3 -m venv venv
. venv/Scripts/activate
(venv)$ pip install -e .
```

### About IfcOpenShell dependency

**Caveat emptor**: make sure, that the IfcOpenShell dependency was properly installed with help of the `python -c 'import ifcopenshell'` command. In case of failure of the importation try re-installing but this time with the verbose flag, that is try

```
(venv)$ pip install -e . -v
```

and look for the lines concerning `IfcOpenShell`.

## Usage

In order to access to the different flavors of tilers, refer to the corresponding readmes to discover their respective usage and features:

- CityTiler readme
- GeojsonTiler readme

ObjTiler readme

- IfcTiler readme
```

- TilesetReader readme

Useful tutorials:

- CityTiler usage example
- GeojsonTiler usage example
- Visualize 3DTiles in Cesium, iTowns or UD-Viz (Annexe 5)
- Create 3DTiles from OpenStreetMap data
- Host CityGML data in 3DCityDB

## Develop with py3dtilers

### Running the tests (optional)

After the installation, if you additionally wish to run unit tests, use

```
(venv)$ pip install -e .[dev,prod]
(venv)$ pytest
```

To run CityTiler's tests, you need to install PostgreSQL and Postgis.

To setup PostgreSQL with Postgis on Windows, follow the first step (1. Download PostgreSQL/PostGIS) of 3DCityDB tutorial.
For Ubuntu, follow this tutorial.

### Coding style

First, install the additional dev requirements

```
(venv)$ pip install -e .[dev]
```

To check if the code follows the coding style, run `flake8`

```
(venv)$ flake8 .
```

You can fix most of the coding style errors with `autopep8`

```
(venv)$ autopep8 --in-place --recursive py3dtilers/
```

If you want to apply `autopep8` from root directory, exclude the *venv* directory

```
(venv)$ autopep8 --in-place --exclude='venv*' --recursive .
```

### Developing py3dtilers together with py3dtiles

By default, the py3dtilers' `setup.py` build stage uses github's version of py3dtiles (as opposed to using Oslandia's version on Pypi. When developing one might need/wish to use a local version of py3dtiles (located on host in another directory e.g. by cloning the original repository) it is possible

- to first install py3dtiles by following the installation notes
- then within the py3dtilers (cloned) directory, comment out (or delete) the line reference to py3dtiles.

This boils down to :

```
$ git clone https://github.com/VCityTeam/py3dtiles
$ cd py3dtiles
$ ...
$ source venv/bin/activate
(venv)$ cd ..
(venv)$ git clone https://github.com/VCityTeam/py3dtilers
(venv)$ cd py3dtilers
(venv)$ # Edit setup.py and comment out py3dtiles reference
(venv)$ pip install -e .
(venv)$ pytest
```

### Concerning CityTiler

- For developers, some design notes
- Credentials: CityTiler original code is due to Jeremy Gaillard (when working at LIRIS, University of Lyon, France)

### Configuring your IDE

When configuring your IDE to run a specific tiler, you must indicate the module you want to run (e.g. py3dtilers.CityTiler.CityTiler) and not the path to the file (i.e. not ${workspace_root}/py3dtilers/CityTiler/CityTiler.py), otherwise python will not be able to resolve the relative import of the Tilers to the Common package of py3dtilers. An example of launch configuration in VSCode:

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "<launch_config_name>", // e.g. "CityTiler" or "bozo"
            "type": "python",
            "request": "launch",
```

```
            "module": "<tiler_module>", // e.g.
py3dtilers.CityTiler.CityTiler
            "args": ["--db_config_path",
"${workspaceRoot}/py3dtilers/CityTiler/<my_config_file.yml>"],
            "console": "integratedTerminal"
        }
    ]
}
```

### Profiling

Python standard module cProfile allows to profile Python code.

#### *In code*

Import modules:

```
import cProfile
import pstats
```

Profile the code between `enable()` and `disable()`:

```
cp = cProfile.Profile()
cp.enable()  # Start profiling

# code here

cp.disable()  # Stop profiling
p = pstats.Stats(cp)
p.sort_stats('tottime').print_stats()  # Sort stats by time and print them
```

#### *In command line*

cProfile can be run in the shell with:

```
python -m cProfile script.py
```

# Annexe 4

## py3dtilers demo

This demo illustrate 3D Tiles tilesets created with py3dtilers. The demo is based on UD-Viz (Annex 5) which is using iTowns to visualize 3D models.

*Note: the code in 3dtilesProcessing.js is widely inspired by the iTowns's 3D Tiles processing.*

**See online demo.**

### Installation

You need Node.js to run the demo.

Clone the repo then install it:

```
git clone https://github.com/VCityTeam/UD-Demo-vcity-py3dtilers-lyon.git
cd UD-Demo-vcity-py3dtilers-lyon
npm install
npm run build
```

### Usage

Run the demo:

```
pyton3 -m http.server
```

The demo is now hosted on `localhost:8000`.

### Refinement

This demo introduces 2 new buttons allowing to change the way iTowns is processing the 3D Tiles.

#### Reverse 3DTiles process

This button reverse the refinement of the tiles.
By default, the parents tiles are displayed before the children. The children are displayed when we focus on a tile.
With the reversed processing, the children are displayed before their parents. The parents are displayed when we focus on a tile.

### Switch position reference

This button change the reference position to refine 3D Tiles. The reference position is either the camera or the mouse position. The tiles will be refined depending on their distance with the reference position.

## Docker

A Docker version of the demo also exists in this repo.

# Annexe 5

## UD-Viz : Urban Data Vizualisation

UD-Viz is a JavaScript library based on iTowns, using npm and published on the npm package repository, allowing to visualize, analyse and interact with urban data.

A tutorial of the game engine can be found here

### Install node/npm

For the npm installation refer here

UD-Viz has been reported to work with versions:

- node version 16 (16.13.2)
- npm version: 6.X, 7.X. and 8.X

### Installing the UD-Viz library per se

```
git clone https://github.com/VCityTeam/UD-Viz.git
cd UD-Viz
npm install
```

### Running the examples

```
cd PATH_TO_UD-Viz
npm run build
cd /
git clone https://github.com/VCityTeam/UD-SimpleServer
cd UD-SimpleServer
npm install
node index.js PATH_TO_UD-Viz 8000
```

- UD-Viz-Template (demonstration) application,
- online demos are available here

## Developers

### Pre-requisite

Developing UD-Viz requires some knowledge about JS, node.js, npm and three.js.

### Developers documentation

After generation, the browsable documentation is stored within this repository.

Refer to this README to re-generate it.

### Coding style

The JavaScript filees coding style is defined with eslint through the .eslintrc.js configuration file. It can be checked (e.g. prior to a commit) with the `npm run eslint` command. Notice that UD-Viz coding style uses a unix `linebreak-style` (aka `LF` as newline character).

### Tips for Windows developers

As configured, the coding style requires a Linux style newline characters which might be overwritten in Windows environments (both by `git` and/or your editor) to become `CRLF`. When such changes happen eslint will warn about "incorrect" newline characters (which can always be fixed with `npm run eslint -- --fix` but this process quickly gets painfull). In order to avoid such difficulties, the recommended pratice consists in 1. setting git's `core.autocrlf` to `false` (e.g. with `git config --global core.autocrlf false`) 2. configure your editor/IDE to use Unix-style endings

#### *Notes for VSCode users*

When using Visual Studio Code, you can install the eslint extension allows you e.g. to automatically fix the coding style e.g. when saving a file .

### Prior to PR-submission 1: assert coding style and build

Before pushing (`git push`) to the origin repository please make sure to run

```
npm run travis
```

(or equivalently `npm eslint` and `npm run build`) in order to assert that the coding style is correct (`eslint`) and that bundle (production) build (`webpack`) is still effective. When failing to do so the CI won't check.

Note that when commiting (`git commit`) you should make sure to provide representative messages because commit messages end-up collected in the PR message and eventually release explanations.

### Prior to PR-submission 2: functionnal testing

Before submitting a pull request, and because UD-Viz still misses some tests, **non regession testing must be done manually**. A developlper must thus at leas) check that all the demo examples (refer to their online deployment) are still effective.

### Submitting a Pull Request

When creating a PR (Pull Request) make sure to provide a correct description

## Sources directory layout (organizational principles)

Definitions: - Component: everything thats is necessary to execute only one aspect of a desired functionality (see also module). - Extension: a component depending on a web service in order to be functionnal. - Widget (web widget): an embedded element of a host web page but which is substantially independent of the host page (having limited or no interaction with the host) - Template: a class build on sibling sub-directories (Game, Widgets, Views) components and proposing an application model - View: decorated/enhanced iTowns Views

```
UD-Viz (repo)
├── src                         # All the js sources of UD-Viz JS library
|    ├── Components             # A set of components used by sub-directories
at this level
|    ├── Templates              # Classes builded with other sub-directory
(Game, Widgets, Views) to propose application model
|    ├── Views                  # Classes of 3D views encapsulating the
itowns view
|    ├── Game                   # A sub-directory offering game engine
functionnality
|    |    ├── Shared            # code that can be executed both and client
and server side to simulate a world
|    |
|    └── Widgets                # A sub-directory gathering a set web web
widgets (UI)
|         ├── Widget_1
|         ├── Widget_2
|         ├── ...
|         └── Extensions        # Widgets depending on an external web
service
├── ...
└── webpack.js
```

Notes: * The position of a specific component in the sub-folder hierarchy reflects how it is shared/re-used by sub-directories. For example if a given component is only used by a single widget, then it gets defined within that widget folder. But when another component usage is shared by two widgets then its definition directory gets promoted at the level of the two widgets.

# Annexe 6

## A template application of the UD-Viz package

This repository holds a template (demonstration) application of the UD-Viz JavaScript package. The goal of this template application is to

- illustrate the main features of UD-Viz,
- provide code that demonstrates how such features can be configured/extended/embeded and eventually combined/integrated within a full autonomous application,
- illustrate the JavaScript ecosystem required for building and running it,
- be used as a template for creating/declining your own application.

Because this template application is fully functional maybe the simplest way to understand what it does is to build it and run it.

### Install npm

For the npm installation refer here

Required npm version: UD-Viz-Template has been reported to work with npm versions npm 6.X.

Reminder: `npm install -g npm@6.14.15` enables to switch npm version

### Installing and running the template application

The template application can be locally (on your desktop) started in the following way

```
npm install
npm run debug      # integrates building
```

and then use your favorite (web) browser to open `http://localhost:8000/`.

Note that technically the `npm run debug` command will use the webpack-dev-server npm package that - runs node application that in turn launched a vanilla http sever in local (on your desktop) - launches a watcher (surveying changes in sources) - in case of change that repacks an updated bundle - that triggers a client (hot) reload

## Technical notes concerning the template application

Some modules used by the DemoFull require some server-side components to be installed on some server (possibly your desktop). For example * the 3D objects (buildings) are (by default) serverd by a LIRIS server and thus require no specific configuratione there is nothing more to do * handling of documents will require you to install the API_enhanced_city. * you can also modify the application configuration file

## Making your own UD-Viz based application

The present `UD-Viz-Template` repository holds all the required elements constituting an independent JavaScript application (using the UD-Viz package among others) as well as the technical means to build and run (and debug) that application. In order to realize your own UD-Viz based application it thus suffice to duplicate this repository and start adjusting, modifying, extending and deriving the code of your duplicate.

First create a new repository, e.g. `https://github.com/exampleuser/MyApp.git` (the git repository does not need to be hosted at github) to host your new application.

Then replicate this git repository which can be done with e.g. the following commands :

```
# Create a scratch directory
mkdir foo; cd foo

# Make a bare clone of this repository
git clone --bare https://github.com/VCityTeam/UD-Viz-Template.git

# Mirror-push to the new repository
cd UD-Viz-Template.git
git push --mirror https://github.com/exampleuser/MyApp.git

# Remove the temporary scratch directory
cd ../..
rm -rf foo

# Cleanly clone your new repository
git clone https://github.com/exampleuser/MyApp.git
```

You can then proceed with using your `MyApp` with exactly the same instructions as for this `UD-Viz-Template` application that is * `npm install` (install the dependencies) * `npm run debug` (building and running the application) * optionnaly you can lint your code with eslint by running the `npm run eslint` command. This new repository now holds a buildable (`npm`

`install`) and runnable (`npm run debug`) application (just follow the `Readme.md` as you did for UD-Viz-Template), that you can start adapting to suit your needs.

The main entry point in order to customization your new `MyApp` application is the src/bootstrap.js file that is centered on UD-Viz's Template.Allwidgets class.

Then you can also adapt the `assets/config/config.json` configuration file that defines e.g. * links to the used `assets` for the icons, logos of your application, * the `extents` i.e. the geographical portion of the territory that will be displayed, * some default data streams used e.g. - the `background_image_layer` that define the terrain (through a `WMS` (Web Mapping Service) stream), - some 3d buildings (based on 3DTiles) refer e.g. to the `3DTilesLayer` entry, - the default `camera` position within the scene, - ...

---

FIXME for all the bottom of this page

Ensuite trois cas d'utilisation:

- on veut créer une demo a partir de brique existante mais on veut configuré lesquelles, dans ce cas on peut utilisé un template et lui filé la config adapté. comme pour le AllWidget template avec une config qui va décrire quel widget est utilisé. EBO: on va conserver la config ?
- on veut créer une demo mais le code n'existe pas, dans ce cas la meilleure méthode est rajouter son code dans ud-viz et de dev avec les deux repo side by side. on peut aussi rajouter son code dans son projet et se demerder avec l'api proposé par les template pour y incorporer son code. typiquement on pourrait dev son propre widget (grace a du code + bas niveau si necessaire, dans ce cas widget) de ud-viz et ensuite l'ajouter via l'api de allwidget template.
- le besoin n'est pas couvert par un template existant. pareil meilleure méthode créer le template dans ud-viz a partir de code + bas niveau de ud-viz (game, widget, views) et de dev side by side sinon dans son projet créer les classes manquantes a partir du code ud-viz plus bas (toujours widget, game, view)

quand je dis meilleure méthode c'est mieux car le projet ud-viz beneficie directement de features réutilisable par les autres dev, et ca évite une étape d'intégration si jamais on désirait l'intégrer plus tard.

### When working with a docker container: the `diff` alternative strategy

If you demo is defined within a docker container then an alternative strategy (to the complete replication of the DemoFull directory) consists in (within your `Dockerfile`) -

cloning the UD-Viz-demo repository, - placing yourself (with `WORKDIR`) inside the `DemoFull` directory, - overwriting the `DemoFull` code with your partial customizations (e.g. just overwriting `BaseDemo.js` and the `config.json` files).

A example of this docker container based strategy can be found in the DatAgora_PartDieu demo as illustrated by the Dockerfile commands.

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

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

[1]Université Lumière Lyon 2, LIRIS UMR-CNRS 5205, Université de Lyon, Lyon, France
[2]Berger-Levrault, Limonest, France

**Commission XX, WG XX/YY**

**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. Nevertheless, the amount of data implies experimenting with this format in order to improve it, for example, the organization of objects in tiles. This format is used by a few products on the market, but with limited functionality to allow experimentation. 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, 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 with the possibility of improvements thanks to the following functionalities:

- 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 keep additional thematic information related to geometries, to visualize them in the same context on the web in real time. 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. It also allows to add semantic data linked to the geometry, if necessary.

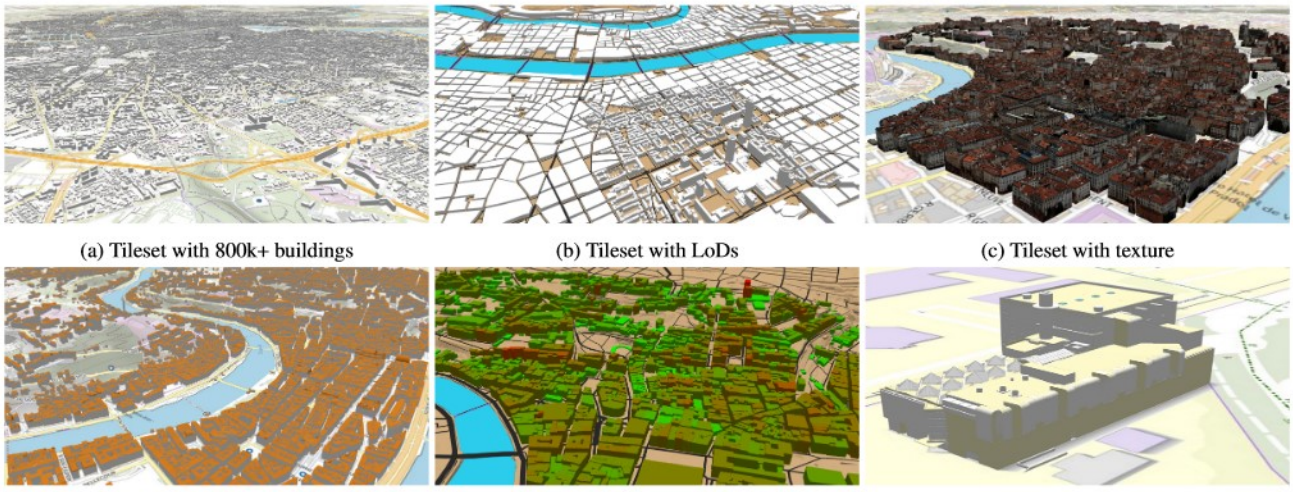
We present in Section 2 the 3D Tiles format and its uses in various works on the visualization of geospatial data. The architecture of Py3DTilers, its functionalities and *Tilers* are presented in Section 3. Some examples of 3D Tiles created with Py3DTilers will be presented in Section 4, followed by 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 OFC. It has been designed to support the massive

[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



(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

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*.

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

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

### 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 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

[8] https://www.safe.com/

[9] https://www.rhinoterrain.com/en/rhinocity.html

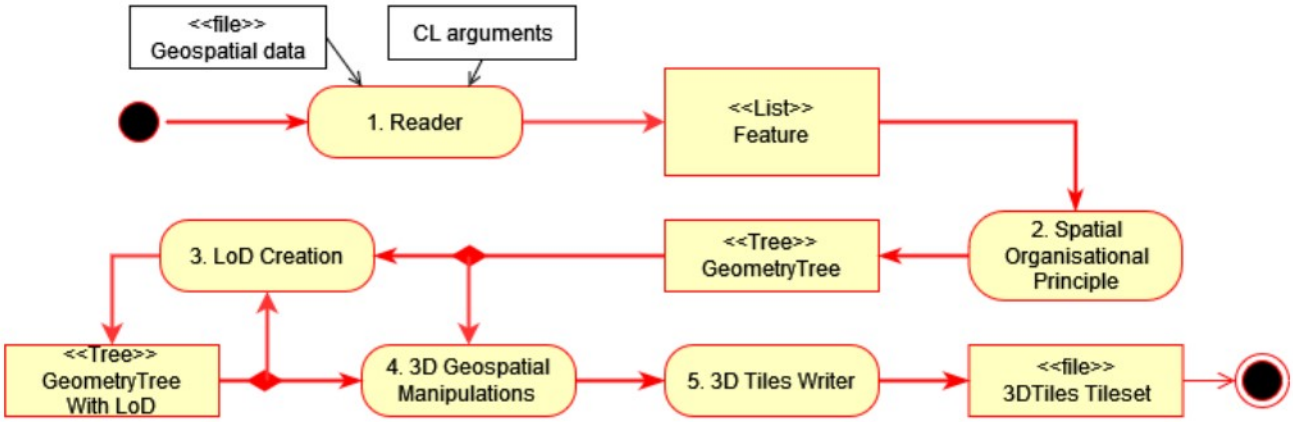



Figure 2. Activity diagram of the 3D Tiles creation process

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. Fon 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.

## 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

[10] https://github.com/Gaia3D/F4DConverter
[11] http://www.mago3d.com/
[12] https://gitlab.com/Oslandia/Py3DTiles
[13] https://github.com/Geodan/pg2b3dm

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.

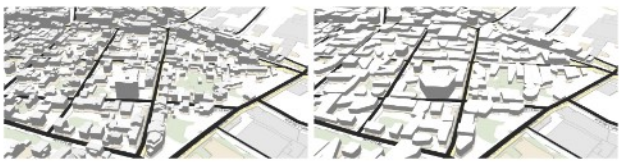
### 3.3 Possible operation on features

**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.

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

The first LoD creation method consists of an extrusion of the footprint of the *features* in order to obtain a simplified model for each *feature* (figure 3).



(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*. For example, these blocks can be defined by the network of roads of the city, where all the buildings in the same block will be represented by a single model (figure 4).

(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, etc.

**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 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 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.

[15] http://ifcopenshell.org/

[16] https://github.com/pywavefront/PyWavefront

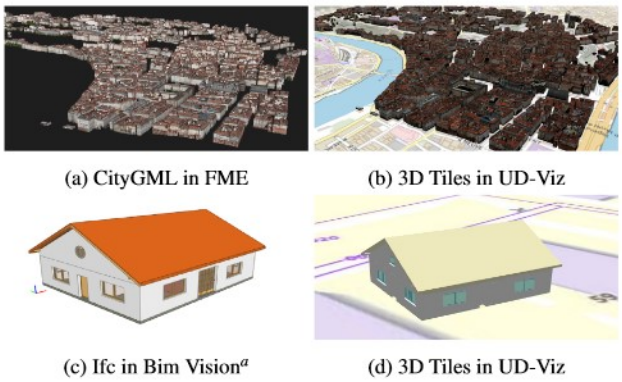
**3.4.5 3D Tiles tileset:** Py3DTilers is able to read and load 3D Tiles (in B3DM format only) into memory. Once loaded, the geometry of each 3D model and its associated Batch Table data are stored as *features*. The data related to the tiles (bounding volumes, geometric errors, etc.) are also stored 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. RESULTS

### 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.



(a) CityGML in FME (b) 3D Tiles in UD-Viz

(c) Ifc in Bim Vision[a] (d) 3D Tiles in UD-Viz

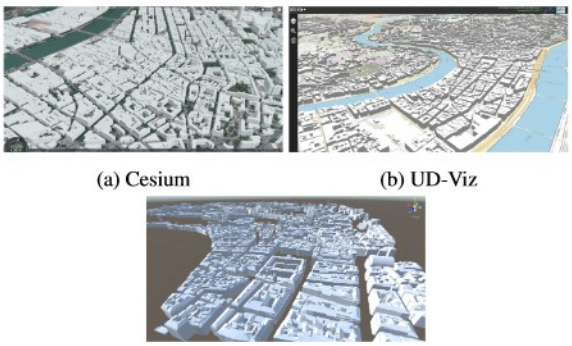
[a] https://bimvision.eu/

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

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.

### 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.

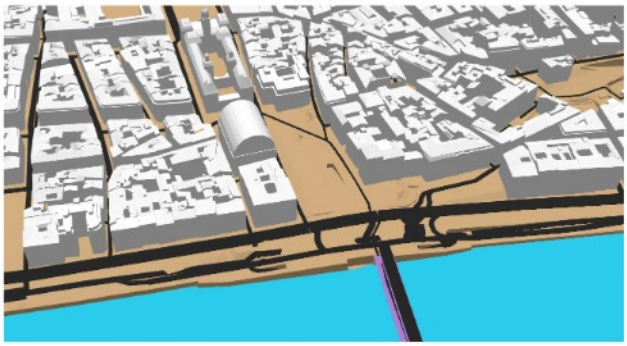


Figure 8. 3D Tiles of buildings, relief, roads, bridges and water bodies

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 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.

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

[18] https://www.itowns-project.org/

[19] https://github.com/VCityTeam/UD-Viz

[20] https://unity.com/

[21] https://github.com/NASA-AMMOS/Unity3DTiles

### 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.

## 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 table, 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.

A current 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.

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[26]. 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[27] 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://www.ogc.org/standards/se
[27] https://cesium.com/blog/2021/11/10/introducing-3d-tiles-next/

## ACKNOWLEDGEMENTS

The authors would like to thank the TIGA project[28], led by the metropolis of Lyon, and the Berger-Levrault[29] 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.

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

## 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.

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.

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 WebGIS 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.

# INTEGRATING MULTIMEDIA DOCUMENTS IN 3D CITY MODELS FOR A BETTER UNDERSTANDING OF TERRITORIES

Corentin Gautier[1], Johanna Delanoy[2], Gilles Gesquières[3]

[1] Université de Lyon, LIRIS UMR-CNRS 5205, Université de Lyon, Lyon, France - corentin.gautier@universite-lyon.fr
[2] INSA Lyon, LIRIS UMR-CNRS 5205, Université de Lyon, Lyon, France - johanna.delanoy@insa-lyon.fr
[3] Université Lumière Lyon 2, LIRIS UMR-CNRS 5205, Université de Lyon, Lyon, France - gilles.gesquiere@univ-lyon2.fr

**Commission IV WG IV/9**

**KEY WORDS:** Multimedia, Urban 3D model, Billboards, Documents, Story telling

**ABSTRACT:**

Digital 3D representations of urban areas, through their growing availability, are a helpful tool to better understand a territory. However, they lack contextual information about, for example, the history or functionality of buildings. On another side, multimedia documents like images, videos or texts usually contain such information. Crossing these two types of data can therefore help in the analysis and understanding of the organization of our cities. This could also be used to develop document search based on spatial navigation, instead of the classical textual query. In this paper, we propose four approaches to integrate multimedia documents in a 3D urban scene, allowing to contextualize the scene with any type of media. We combine these integration approaches with user guidance modes that allows to guide the user through the consumption of these media and support its understanding of the territory. We demonstrate the usefulness of these techniques in the context of different projects within the Lyon area (France). The use of multimedia documents integrated into a digital tour allows, for example, the iconic buildings to be contextualised or to understand the evolution of a territory through time.

## 1. INTRODUCTION

Digital 3D representations of urban areas are becoming widely available. Multiple tools exist to visualize them such as Google Earth[1] or Cesium[2]. By allowing to wander into virtual cities, these tools allow a better understanding of a territory. However, most of them are limited to a 3D representation of the buildings, without providing more information about their functionality, their evolution or other elements of context. Aside from 3D representations, multimedia (e.g. images, texts or videos) also plays an important role in the understanding of the urban landscape. Archival images, videos on the history of a district, job descriptions of certain industries or interviews in a district can provide important additional information about a territory. While these media have the potential to enrich virtual 3D cities, the link between those multimedia and digital 3D representations is rarely made.

Online mapping systems (e.g. Google Maps) allow to wander in a 2D or 3D view and pick information through labels or pins that simulate points of interest. By linking multimedia to a spatial position, users can find information more easily in a desired area. This can be seen as another form of search, rather than formulating a query on a web browser. As such, these tools lack ways to fully integrate the media into a 3D view of the city, and usually lack of explanatory media on the organisation of a district or a whole city, on its history, etc.

We propose a model that allows to combine a 3D view of the city in which the user can navigate with a collection of multimedia that provides additional information about the underlying 3D scene. We present new ways to integrate each multimedia document in the 3D scene as well as ways to guide the user through a collection of different documents. More precisely, we introduce four modalities of multimedia integration in a scene (shown in blue in Figure 1) that are presented in Section 3:

- The **3D geo-pinned multimedia** uses the principle of pins that target a point of interest and are linked to a multimedia document. Our extended pins gives an overview of the linked document.
- The **3D geo-web renderer** shows documents at given positions in the scene, always facing the camera such that the user can get the information from any place in the scene.
- The **extended document** superimposes images on a 3D model and complete them with textual data. The extended document is linked to a precise camera position.
- The **slideshow** allows to make a presentation in a view by placing documents in a 3D space.

A large number of documents linked in a scene do not necessarily guarantees a better understanding of a territory, as users might need to be guided through them in order to understand them. We thus present ways to guide the user through its consumption of documents in the scene. Our three user guidance modes range from a fully sequential order of consumption, to a free mode where the user can wander in the scene and pick the documents as he wishes. These modes and the way they can can be mixed with the multimedia integration modalities are presented in Section 4.

The combinations of the integration modalities and user guidance modes open the door to a large variety of choices in the way to use multimedia documents in combination with 3D virtual cities. We demonstrate the potentialities of several of these

[1] https://www.google.com/intl/fr/earth/
[2] https://cesium.com/

combinations in the context of multiple projects developed in the city of Lyon (France) in Section 5. Such projects include a web-documentary that aims to deconstruct preconceived ideas about a territory or presentations about the future of a neighbourhood using a tangible model.

## 2. RELATED WORKS

Many tools exist to navigate in a virtual urban environment (Bleisch and Dykes, 2015, Boutsi et al., 2019, Jaillot et al., 2021). Thanks to the navigation in the scene, users can visualize a city or any 3D geometry under different angles. This can greatly help to better understand an urban space by virtually navigating in it.

Other tools, such as web-documentaries, also allow to walk in an urban space and better understand it without making use of a 3D scene. These tools can target a wider audience as they are web-based and do not require any specific hardware. Their 2D representation of the digital territory is enhanced by additional information such as video ("Correspondances" [3]) or images ("Balade au merlan" [4]). However, these tools remain limited in their interaction because of the 2D representation of the world that they carry. We would like to take up this spatial contextualization of documents to the third dimension and integrate such documents in 3D walking tours.

Enhancing 3D scenes with multimedia documents have been used to show the evolution of a city through time (Jaillot et al., 2021, Chagnaud et al., 2016). Such documents can bring information about the state of a city at a certain date through images of buildings at different times or their chronology. These works have introduced several methods to visualize historical photo documents in a 3D scene, allowing to show the urban past but also the future planning of cities. However, they lack the possibility to provide additional information on these pictures to help reach a better understanding. They also do not allow to use any other type of media, such as videos of web pages, or to interact with them.

Other methods have tried to contextualise a digital model with quantitative data (Bleisch and Dykes, 2015) or text and images (Boutsi et al., 2019) in a scene. This gives another perspective on the data and a better understanding of a territory by linking information to a geospatial position. These methods are good examples on how a 3D representation and multimedia or data can complement each other. However, they only integrate one type of content and do not give an overview of the different documents present in the scene, making the search for a specific document or data harder. We want to provide a new method of document search through a digital representation of a territory by allowing an explicit overview of the different multimedia available.

Rather than images or data, web pages can also be integrated in a 3D context to contextualise a 3D scene (Wijnants et al., 2015). This allows giving the user access to more content, such as external websites, while not losing the focus on the digital representation of a territory. The interest of integrating the web page directly into the 3D scene is also to support the relationship between the target and its related annotation objects (Seo et al., 2015). While we follow the same goal, we aim to provide a model that allow to integrate any type of multimedia in the scene and to bring information on a specific geometries like building or district (Samuel et al., 2016).

Multimedia documents can also have interesting metadata like their title, source, publication date, key content, tags. (Gan et al., 2014) gives an overview of the existing techniques used to visualize a multitude of documents both in 2D and 3D environment. However, they do not target geographic applications and the integration in 3D urban scenes.

While integrating multimedia document in the 3D digital representation of the city can greatly improve the understanding of such area, the user can quickly be lost in this multitude of information and not know where to start searching. Some works have shown the interest in guiding the user by making him follow a logical sequence of information. It also helps to engage the user and allow him to be more effective in its reading of the information (Othman et al., 2011). In our work, we combine methods to integrate multimedia into 3D urban scenes with ways to interact with a collection of documents, providing different levels of guidance to the user.

Existing methods of integrating multimedia do not satisfy our need to manage multiple types of document or to provide interaction with them. We provide several ways to integrate such multimedia in the 3D scene, depending on the type of such documents but also on their goal and content. These integration modes can be combined with user guidance modes that help reach better user experience and understanding of territory.

## 3. MULTIMEDIA INTEGRATION APPROACHES

Our objective is to integrate different types of multimedia documents into a 3D urban scene in order to make the representation more informative and interactive. The integrated documents can thus be videos (eventually 360 videos) as well as images, textual documents or web pages, as shown in Figure 1. They are all georeferenced and have a position in space. Although this position is a natural link with the 3D scene describing the city, integrating the media visually in a simple interface such that the user can mentally connect them to the 3D representation of the city is not easy.

We propose four methods to integrate such documents in the 3D scene, explained thereafter and shown in blue in Figure 1. The choice of one of them depends on the type of media but also on their meaning or their number. This integration makes it possible to have all the information on a district on the same web interface, easing the understanding of such areas. But it can also help to find documents or information by wandering in the digital representation of the city and browsing the multimedia that are available. Each of these integration modalities can be used in the 3D scene with different user guidance modes (free access, conditional and sequential) that will be described in Section 4 and are shown in yellow in Figure 1.

The first two approaches (**3D geo-pinned multimedia** and **3D geo-web renderer**) integrate a media into a 3D scene such that it is linked to a precise spatial location and always shown facing the camera (similar to the billboard technology). The **extended document** is linked to a camera position and superimposes the media to the 3D scene, while the **slideshow** projects a series of media onto a 3D plane in the scene. All of these methods aim to be integrated in a web-based library for 3D visualization of geographical data.

[3] https://correspondances.tv5monde.com/
[4] http://unebaladeaumerlan.fr/

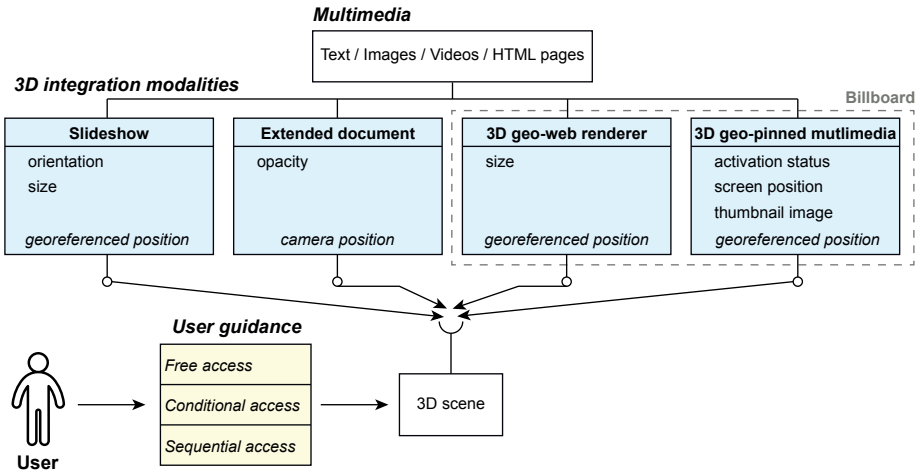



Figure 1. Architecture diagram of our proposed model. The multimedia can be integrate in the 3D scene using one of the multimedia integration approaches (in blue). Each approach is characterized by attributes that allow to parameterize the integration. The media are linked to the scene by either a georeferenced position or a camera position (shown in italic). The user can consume the integrated media in the scene through different guidance modes (in yellow).

### 3.1 Billboard inspired approaches

These approaches consist in an arrangement of thumbnails directly shown in the 3D scene and targeting a precise spatial point. The thumbnail is always facing the camera such the user can access the information from any position in the scene. The two approaches differ in the way the document is displayed through the thumbnail: either directly or summarized in an interactive pin. In either cases, the integration is characterized by a georeference position (GPS coordinates).

The **3D geo-pinned multimedia** is based on the principle of pins that target a point of interest. We enrich the traditional pin by adding a thumbnail above it that gives a visual feedback on the nature and content of the multimedia. This pin is meant to be interactive: when selected, the linked document appears in 2D on the screen in front of the 3D scene (see example in Figure 2). The pin and its thumbnail are always facing the camera while the multimedia document is displayed as a 2D element that will not move with the camera. This approach allow to give the user a global vision of all the contents available in a scene. The user can then pick from this data and decide to get more detailed information about a place or building pointed out by the pins. The interactivity of the pin can be deactivated, allowing to control what the user can access as content at a given moment. In this case, the user can still see the pin and its thumbnail, thus knowing that there is a document accessible, but will not be able to consume the linked document. In addition to its spatial position, the 3D geo-pinned multimedia is thus also characterized by its activation status, the position where to display the document on the screen and its thumbnail image (see Figure 1).

The **3D geo-web renderer** also points to a precise location in the 3D scene but the linked document will be directly shown to the user when he approaches it. Again, the document will be displayed always facing the camera so that the user can see it from any position in the scene. This way, the user can continue walking around in the 3D scene and mentally put this document in context. An example of the integration of a web page with this approach is shown in Figure 3. The size the media appears in the 3D scene is an additional parameter in this integration method.

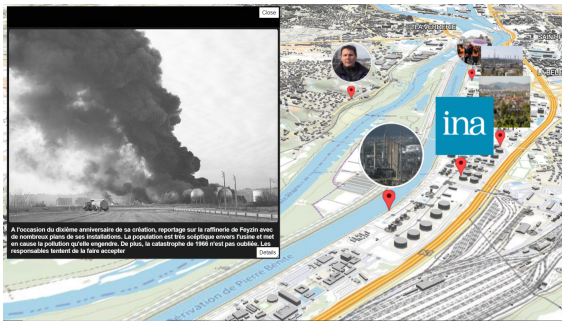



Figure 2. An example of the 3D geo-pinned multimedia approach. The pins show the available multimedia on the 3D scene while the selected document is shown on 2D at the left of the screen.

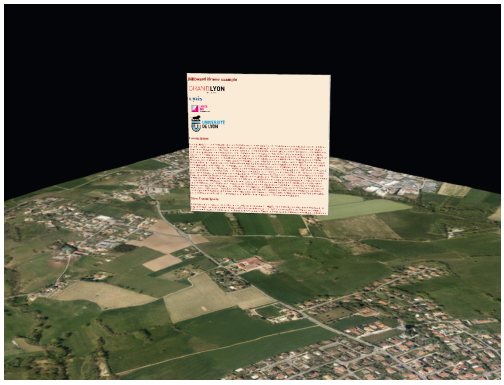


Figure 3. An example of the 3D geo-web renderer approach. An html page is shown over the 3D scene.

These two billboard approaches are similar in the way they are integrated into a scene, they both target a precise location in space and allow to quickly see which content is accessible while moving into the scene. However, by summarizing the information into small pins instead of directly showing it, the 3D geo-pinned multimedia allows to display more document without cluttering the space. It could thus be a preferred solution when

a large number of documents need to be shown in a small space.

### 3.2 Extended Document

The **extended document** approach displays a multimedia document in a 2D context on top of a 3D scene. The integration of the document in the 3D scene is done by modifying the position and orientation of the camera, rather than positioning the document in the 3D space (see Figure 4). In order to access different documents in the same scene, the user has to choose the documents he is interested in in a list. When selected, the camera will move to the right position and overlay the document on top of the 3D scene. In order to keep the link with the 3D scene, the camera will move smoothly toward the target position, allowing the user to locate more easily its position in 3D. Documents in this approach are thus characterized by the position and orientation of the camera, as well as the opacity of the element to be overlaid.

The extended document is composed of three distinct windows:

- The **navigator windows** (Figure 4, left-most window) which allows to navigate between the different documents with their titles and reference dates. This window also has a filter system for a better navigation among the documents .
- The **inspector windows** (Figure 4, right-most window) gives a preview of the document that will be superimposed in the 3D scene, as well as all additional information about it such as its description, source, and publication date. It allows the user to have a first understanding of the document and verify that it is the media that he wants to consume.
- The **visualization windows** (Figure 4, middle window) is activated when the user decides to engage the visualization in the inspector window. It will smoothly move the camera to the correct position and superimpose the document on the 3D view. It also allows to change the opacity of the document in order to compare it with the underlying 3D.


Figure 4. An example extended document with its three windows: the navigator window showing the list of available documents (left); the inspector window showing the metadata of the selected document (right); the visualization window that superimposes the document on the 3D scene (middle).

This approach will be preferably used with images, or animated images, as it allows to directly compare it with the 3D view of the scene. It can help users to realize the evolution of a district or a building, or observe it under another format.

### 3.3 Slideshow

The slideshow is anchored at a 3D position in the 3D scene and can display the documents on a 3D plane (see Figure 5). This plane is traditionally parallel to the ground but can be also set vertically to contextualize face buildings. The documents will thus not follow the camera as the user move into the scene. The slideshow is characterized by its center position (georeferenced position), its size and its orientation. This method overlays the 3D model of buildings on top of the documents, that can be images or videos of data layers on a district, or different states of a territory. For example, one could drop the image of the transport network of a city and see how it fits with the morphology of that city. The slideshow is made to integrate a series of documents: it simulates a presentation in the 3D world in which one can scroll through different documents, located at the same place, in order to illustrate problems or ideas on a district. The user can easily drag and drop different images or videos into the scene, allowing to quickly illustrate different ideas, but also to work in team and experiment different solutions and illustrations. A geographic mask is made available to prepare data directly in dedicated GIS information systems tools.

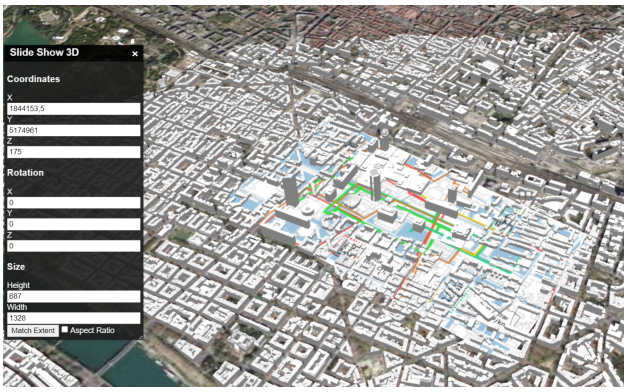



Figure 5. An example of the slideshow. The configuration window (on the left) allows to set its parameters. One image of the slideshow is shown on a ground plane in the 3D scene.

## 4. USER GUIDANCE

Our different multimedia integration approaches allow to integrate a large variety of multimedia in a scene, from an archive image to a web page. When becoming too numerous in one scene, the user could get lost among them thus breaking the goal of our approach to bring a better understanding of a territory. We thus design several ways to consume a collection of documents, by possibly guiding the user through their consumption:

- **Sequential access**: The documents are provided in a given order. As in a guided tour, this allows to build a narrative and guide very precisely the user through the documents. This mode limits the order of freedom of the user, it can only act on the time to spend on each document, but helps to build a better understanding of the documents.
- **Conditional access**: Going through some documents might be needed to access others, but the order is not completely constrained. Several documents are available at one moment in which the user can chose freely, but viewing some or all of them is needed to access the next ones.

- **Free access**: the user can consume the different multimedia in the scene as he wishes.

Most of our integration approaches can be used with any of these user experiences, although some are more naturally suited for some specific user guidance. The only exception is the slideshow, that is essentially made for sequential access. It allows a linear succession of images or videos on a 2D map and can be used in the case of a presentation to contextualise a district.

The extended document, because it has smooth transitions between documents and is not anchored at a specific position in the 3D scene, is well suited to be used in a sequential approach. It can be used to build a *guided tour* as a list of extended document where each of them with be contextualized in the tour with additional text. It can, for example, consist of a sequence of documents that show the evolution of a district with photos from different eras.

Both the 3D geo-pinned multimedia and the 3D geo-web renderer can be used naturally with any of the guidance. However the 3D geo-pinned multimedia is well suited for conditional access thanks to its activation status. It allows the user to see all the contents that is available in the scene through the pins, but the activation of some of the documents can be conditioned to the viewing of other ones.

## 5. IMPLEMENTATIONS

We experimented our different approaches on various use cases around the city of Lyon (France). Each of these use cases mobilizes one of our multimedia integration approaches, combined with a user guidance mode, allowing a wide variety of user experiences. These implementations are implemented within the UD-Viz library[5], a JavaScript library allowing to visualize, analyse and interact with urban data.

### 5.1 Chemistry Valley: 3D geo-pinned multimedia

We demonstrate the use to the 3D geo-pinned multimedia in the context of a web-documentary, developed as part of a collaboration between a training center for the Chemistry industry Interfora[6] and the Lyon metropolis. This web documentary aims to make the citizens rediscover the area of the Chemistry Valley (South of Lyon, France) through a 3D digital wandering where the user can interact with documents arranged in the 3D scene.

A large number of media, from different types were to be set in the scene. As examples:

- A photographic observatory [7] of the Chemistry valley, consisting of a set of pictures of the chemistry valley from different angles.
- Additional layers as for instance bus lines, the atmospheric index, etc. can be used thanks to numerous data available via WFS (Web Feature Service) and WMS (Web Map Service) in the open data data.grandLyon.com[8].

[5] https://github.com/VCityTeam/UD-SV
[6] https://www.interfora-ifaip.fr/
[7] https://www.caue69.fr/1/page/10622/page
[8] https://data.grandlyon.com/

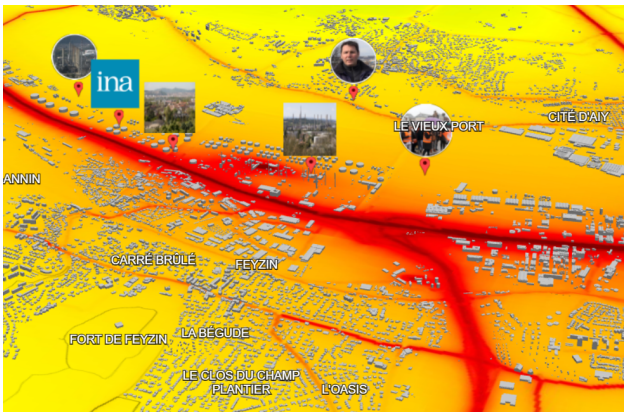



Figure 6. The 3D geo-pinned multimedia modality used in the context of the Chemistry valley. The pins show an overview of the different media available in the scene.

- Jobs descriptions located in the chemistry area. These pdf files provide information on workers in the area.
- Videos of stakeholder interviews in the Chemistry valley on the theme of employment and training.

The 3D geo-pinned multimedia is the best choice in this case since it allows to easily navigate between a large number of media, without cluttering the scene. Additionally, some of these media must be seen before others in the project specifications, naturally leading to using a conditional access mode. This also claims in favor of using the 3D geo-pinned multimedia.

```
"episode-1-data":{
 "content-1" :{
  "lock" : false, // document activation
  "position" :{ // coordinates in the scene
   "x" : "1843554.77",
   "y" : "5165405.73",
   "z" : "220"
   },
   "imgUnlock":"./assets/img/Observatoire/
       cheminee.jpg",
   "imgLock":"./assets/img/Observatoire/
       cheminee.jpg",
   "text": "Vallee de la chimie -
       Observatoire photographique",
   "src":"https://umap.openstreetmap.fr/fr/
       map/vallee-de-la-chimie"
 }
```

Figure 7. An example of a JSON configuration file used for the 3D geo-pinned multimedia.

The conditional access allows to prioritise some contents, especially for the interviews that were created specifically for this experience. These videos were made to be seen in a given order in order to understand the whole issue. As a reminder, in the 3D geo-pinned multimedia modality, documents can be naturally deactivated.. Their pin and thumbnail are still shown on the 3D scene but the user can not interact with them to access the document. In order to better guide the user towards the accessible content, we symbolize the deactivation of the media by a lock image superimposed to the thumbnail. An example of a visualization of the Chemistry valley in this project can be seen in Figure 6: the atmospheric index is used as a color map on the

ground of the 3D model while several pins show the available content.

Each interactive element in this scene can be configured through a JSON file that contains the geospatial position, the display thumbnail, its activation status, the text content and the document link. An example of such configuration file can be seen in Figure 7.

The integration of multimedia in the 3D scene makes it possible to show the different professions that exist in the Chemistry valley, and help the users to know more about the Chemistry industry in this territory. To further this understanding, we have also integrated urban data into the scene by using geoservices[9], spatial web services that make geodata available in a structured form. Finally, in order to show the valley differently, we have integrated WMS (Web map service) and WFS (Web feature service) which allow the display of georeferenced geometries such as building cadastres.

Thanks to the 3D geo-pinned multimedia method, we have brought another form of information search through the wandering in the 3D scene. The user can pick up interviews with the stakeholders of this territory and get information on where they are located and which company they belong to before even starting to watch the video. This makes it easier for him to choose the content that interests him.

### 5.2 Flying campus: 3D geo-web renderer

We use the 3D geo-web renderer method in the context of the project "IMUV/Flying campus"[10]. The aim of this project is to create a virtual space in which photographic exhibitions, conferences, or other events can be organised. It is an interactive space, floating above the city, where a variety of elements can be arranged and where several users can gather. Elements to be displayed can be of any types, including web pages. The use of the 3D geo-web renderer allow to integrate any type of media into the 3D space, such that the user can still be in immersion while consuming the content. One of the use case is a conference room in which a 3D geo-web renderer is placed to broadcast a video, the users can gather in the room and freely interact with the video. Figure 8 shows a user observing the city in 3D while looking at a map showing the organisation of the city.

We implemented the 3D geo-web renderer using the CSS3Drenderer technology (a web rendering method from the javascript library ThreeJS[11]) that allows to include web pages, with which the user can interact, in the 3D scene. Our implementation consists in an overlay of two types of rendering: a CSS3Drenderer which shows the document to be displayed and the WebGLRenderer which is the 3D scene rendering of the walkthrough (Seo et al., 2015). We make a part of the 3D scene rendering transparent and render the CSS3DRenderer context behind the WebGLRenderer context. This allow the content of the document in the CSS3DRenderer to appear in the 3D scene.

By placing these billboards in the 3D scene of the flying campus, the user does not loose focus and has all the information in one web application. If the document to integrate is a web page, the user can freely interact with it as if it was facing the window of a web browser.

[9] https://geoservices.github.io/
[10] https://demo.liris.cnrs.fr/vcity/flyingcampus/
[11] https://threejs.org/

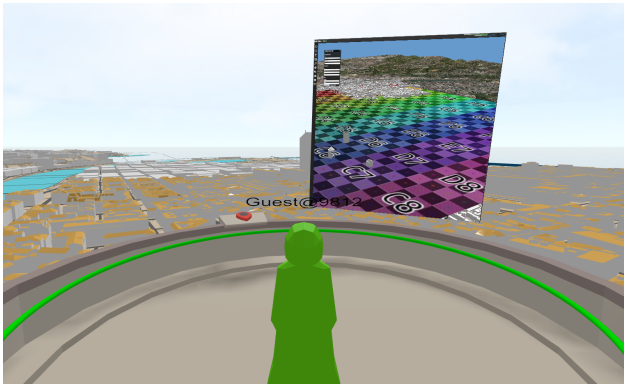



Figure 8. An example of the use of the 3D geo-web renderer in "Flying campus", a virtual gathering place. The user can still observe the 3D scene while the integrated multimedia is shown to the user.

In this context, the user is free to consume the content of these multimedia as he wishes and is not limited by an order to follow (free access).

Flying campus demonstrates the value of this multimedia integration as a way to virtually share multimedia experience and build participative experiences.

### 5.3 Historical guided tour: Extended document

In this project, we use the extended document as a way to show the evolution of a district over time. Our goal was to integrate a collection of archival photos from 1760 to 2017 in the 3D scene. Each picture shows how the district looked like at a specific date along with additional information. By allowing to superimpose the picture on top of the 3D scene, at its correct position, the extended document is naturally suited for this kind of use cases. By playing on the opacity slider, the user can more easily observe the morphological evolution of a building or a district between two periods. An example showing the historical organisation of a district with some additional information is shown in Figure 9

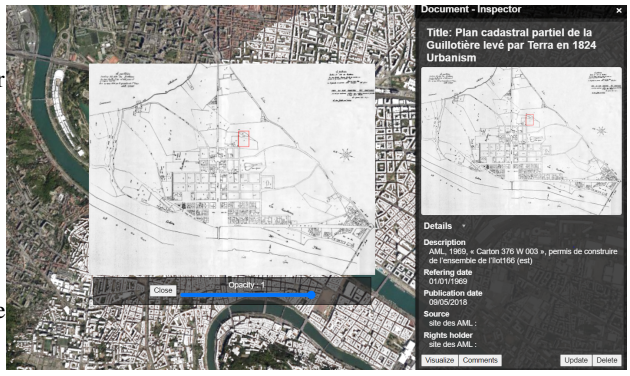



Figure 9. An historical cadastral plan is shown superimposed on top of the 3D scene by using the extended document modality.

To bring more coherence between these documents, we used the sequential access mode as a user guidance. We thus developed a **guided tour** in which the user can follow the chronological evolution of the district and better understand how it has evolved through the ages.

This use case allowed us to provide information about the evolution of a district thanks to archive images and the integration in the 3D scene.

### 5.4 Gratte-Ciel project: Slideshow

We use the slideshow approach in the context of an event organized during the Anthropocene week[12]. Several multidisciplinary students (Geomatician, urban planner, information- communication) gathered on the issue of using a tangible or digital model as a technological tool for the intelligibility of territories. The students produced images or videos as an information layer over a district of Lyon, that they could contextualize with the 3D model of the same district. Three examples of such layers are shown in Figure 10. The images were produced such that they fit the underlying 3D geometry and can thus be superimposed with it. The slideshow method allows to see the geometry of building on top of these layers, allowing to better understand the interaction between those buildings and the data represented in the different layers.

The students could drop their vizualisations onto the 3D map and animate a presentation by going through the different maps. They have proposed still image of geographic data or animated simulation with multi-agent model.

The aim of this project was also to show how tangible models could help in understanding urban projects. In addition to the 3D scene, we also provided the students with a tangible model of the neighbourhood that was built using white Legos. The white color allows to easily project information into the model. Rather than only seeing their maps in the digital 3D scene, students could project them onto the tangible model. Figure 11 shows an example projection on the model. We calibrated a video-projector such that the 3D scene in UD-Viz was matching with the tangible model, and we projected the slideshow module on the model. Using the slideshow on a tangible model allows to more easily understand and discuss about the interaction between the building and other data.

In these examples, we could observe the interest in adding an image to a 3D scene and having a way to present information linked to the 3D world. The mix of 3D modelling of buildings and videos or map backgrounds allowed to better understand the problematic of the district and the possibilities of the new urban project.

### 5.5 Building an home made application

All the described components are delivered as open source content available at https://github.com/VCityTeam/UD-Viz. Each of the demonstration project can be tested using the following links:

- Chemistry valley project using the 3D geo-pinned multimedia: https://github.com/VCityTeam/UD-Demo-TIGA-Webdoc-ChemistryValley
- Flying campus using the 3D geo-web renderer: https://github.com/VCityTeam/UD-Viz/blob/master/examples/Billboard.html
- Historical guided tour using the extended document approach: https://github.com/VCityTeam/UD-Demo-VCity-Spatial-multimedia-db-Lyon. The docker can be found at https://github.com/VCityTeam/UD-Demo-VCity-Spatial-multimedia-db-Lyon
- Gratte-ciel project with the slideshow approach: https://github.com/VCityTeam/UD-Demo-Anthropocene-GratteCiel-. The docker can be found at https://github.com/VCityTeam/UD-Demo-Anthropocene-GratteCiel-docker

Some demos are provided as dockers to ease their deployment. All the source code is accessible such that the different integration methods can be exploited in other contexts.

[12] https://ecoleanthropocene.universite-lyon.fr/a-quoi-revent-les-maquettes–247850.kjsp?RH=1633680335198

## 6. CONCLUSION

Multimedia documents play an important role in understanding the urban landscape. We have presented several ways to integrate multimedia in a 3D scene in order to bring a better understanding of a territory. We also introduced different user guidance modes, going from a fully guided sequence up to a free consumption of media, than can be combined with our multimedia integration modalities. Different use-cases around real project allowed us to demonstrate the variety of user experiences that can be created with these approaches. The addition of multimedia in the urban digital model brings another dimension to it, and greatly improve the way user can understand and discover a territory.

We currently demonstrated use cases making use of a unique integration approach for all the media in the scene, but the different approaches can be combined in the experience.

While our integration approaches can cover various scenario and multimedia types, it might be complicated to find the right integration method for a given use case. It is also not clear yet what are the exact effects on the user experience for each of these modes. Additionally, while our user guidance approach helps the user to find its way in presence of a lot of content, a filtering or grouping system would also prove useful. Being able to assign themes or time period to the different contents would allow the user to filter the different media, but also to group them in the scene and develop new modalities to interact with them. For example, linking the multimedia with a time period would add a fourth dimension to the scene, allowing to travel through time in the scene. The user could then observe the evolution of a district or a building by navigating through documents of different ages.

## REFERENCES


Bleisch, S., Dykes, J., 2015. Quantitative data graphics in 3D desktop-based virtual environments – an evaluation. *International Journal of Digital Earth*, 8(8), 623–639.

Boutsi, A.-M., Ioannidis, C., Soile, S., 2019. Intercative online visualization of complex 3D geometries. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W9, 173–180.

Chagnaud, C., Samuel, J. S., Servigne, S., Gesquière, G., 2016. Visualization of Documented 3D Cities. *The Eurographics Workshop on Urban Data Modelling and Visualisation, UDMV 2016*, Proceedings, Liège, Belgium, 87–93.

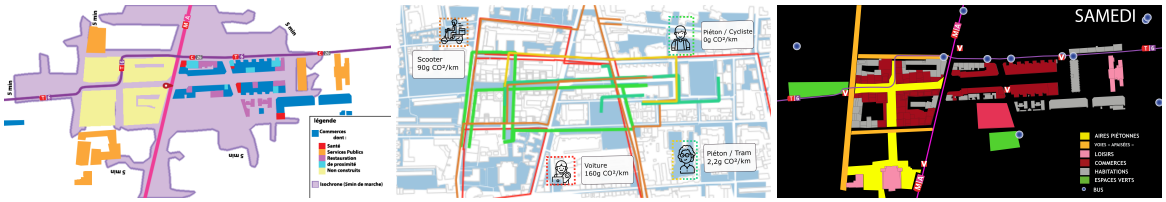



Figure 10. Three examples of image layers created by students for the slideshow. These layers can be used one after another in a presentation.

Figure 11. One image of the slideshow projected onto the tangible model.

Gan, Q., Zhu, M., Li, M., Liang, T., Cao, Y., Zhou, B., 2014. Document visualization: An overview of current research. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6.

Jaillot, V., Rigolle, V., Servigne, S., Samuel, J. S., Gesquière, G., 2021. Integrating multimedia documents and time-evolving 3D city models for web visualization and navigation. *Transactions in GIS*, 25(3), 1419–1438. Publisher: Wiley.

Othman, M. K., Petrie, H., Power, C., 2011. Engaging Visitors in Museums with Technology: Scales for the Measurement of Visitor and Multimedia Guide Experience. *Human-Computer Interaction – INTERACT 2011*, 6949, Springer Berlin Heidelberg, Berlin, Heidelberg, 92–99. Series Title: Lecture Notes in Computer Science.

Samuel, J., Périnaud, C., Gay, G., Servigne, S., Gesquière, G., 2016. Representation and Visualization of Urban Fabric through Historical Documents. E. D. L. ISBN 978-3-03868-011-6 (ed.), *14th Eurographics Workshop on Graphics and Cultural Heritage*, Proceedings of 2016 - Eurographics Workshop on Graphics and Cultural Heritage, Genova, Italy, 157–166.

Seo, D., Yoo, B., Ko, H., 2015. *Webized 3D experience by HTML5 annotation in 3D web*.

Wijnants, M., Erum, K., Quax, P., Lamotte, W., 2015. *Web-mediated Augmentation and Interactivity Enhancement of Omni-directional Video in Both 2D and 3D*.