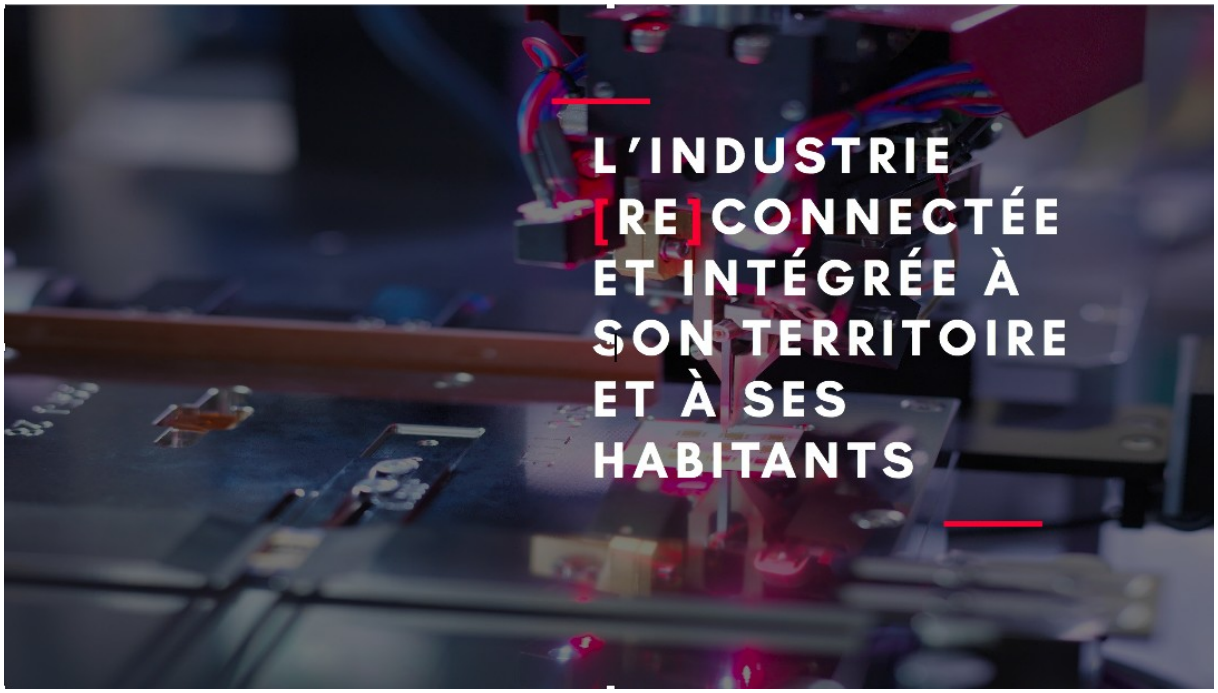


TIGA - Action 14

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.

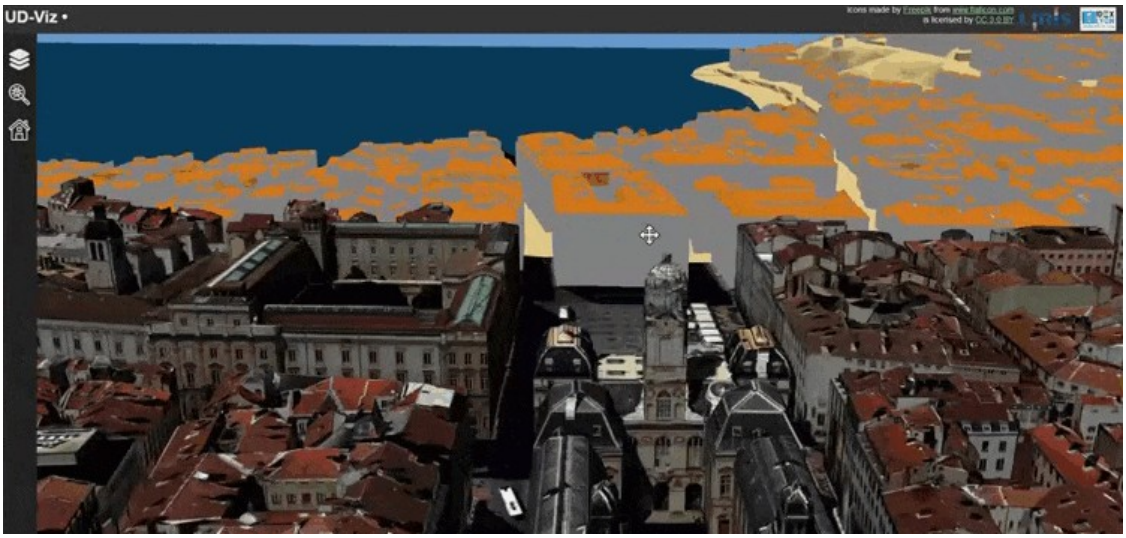


Figure 1 : UD-viz démonstration, application des textures sur 1er arrondissement de Lyon

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.

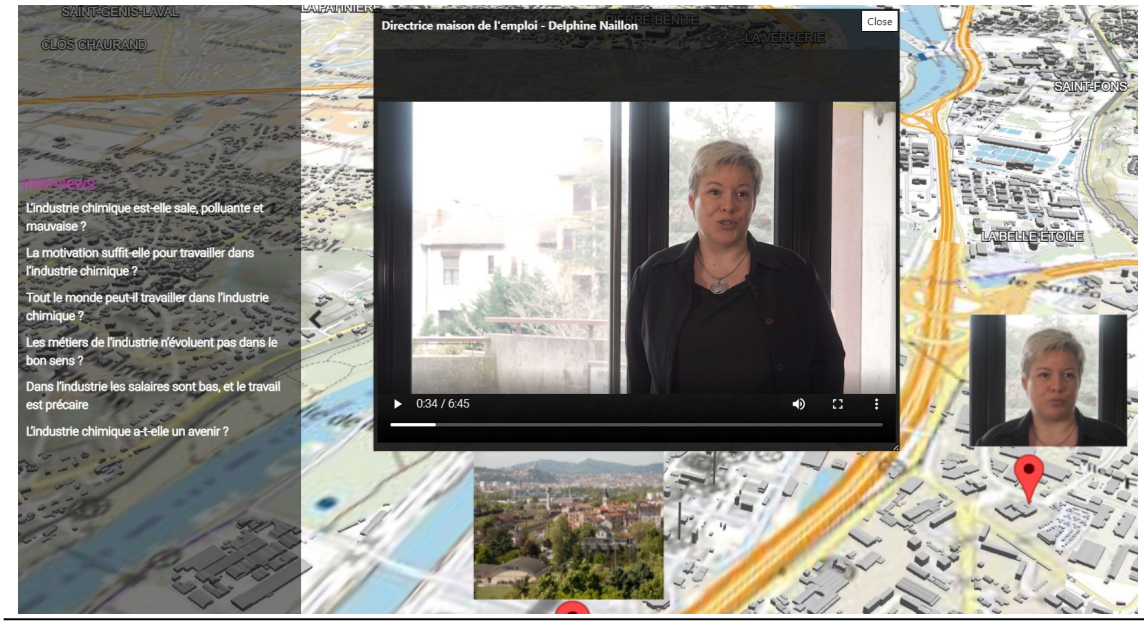


Figure 2 : "Derrière les fumées", intégration d'une interview de la directrice de la maison de l'emploi dans la représentation numérique de la vallée de la chimie

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.

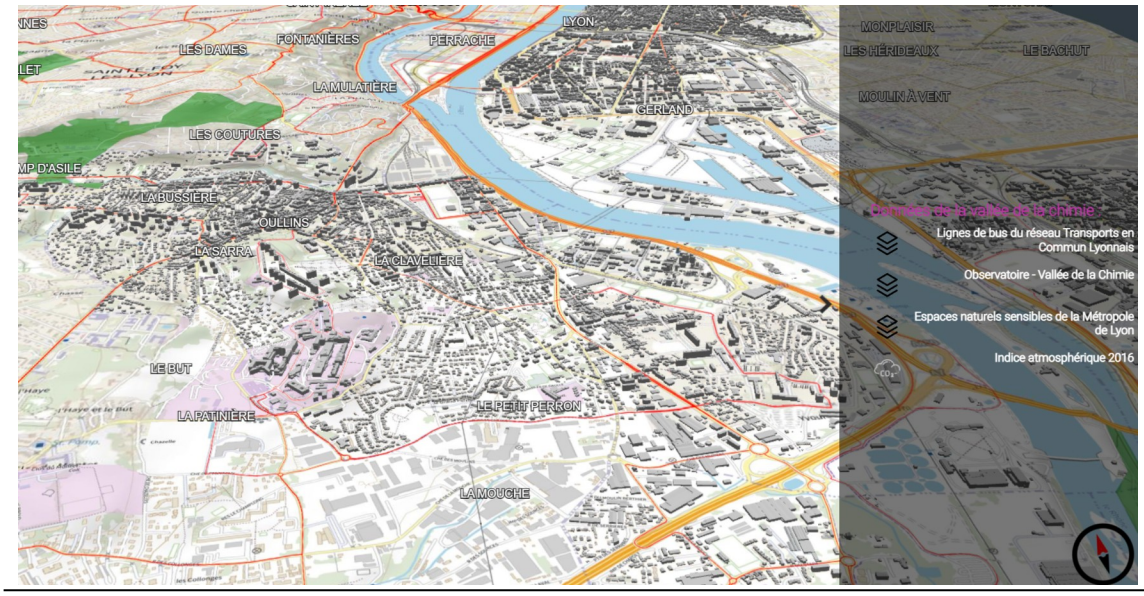


Figure 3 : "Derrière les fumées": intégration de données urbaines 2D tel que lignes de transports en communs (lignes rouges) et les espaces naturels sensibles de la métropole de Lyon (polygones verts) dans la représentation numérique de la vallée de la chimie

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.

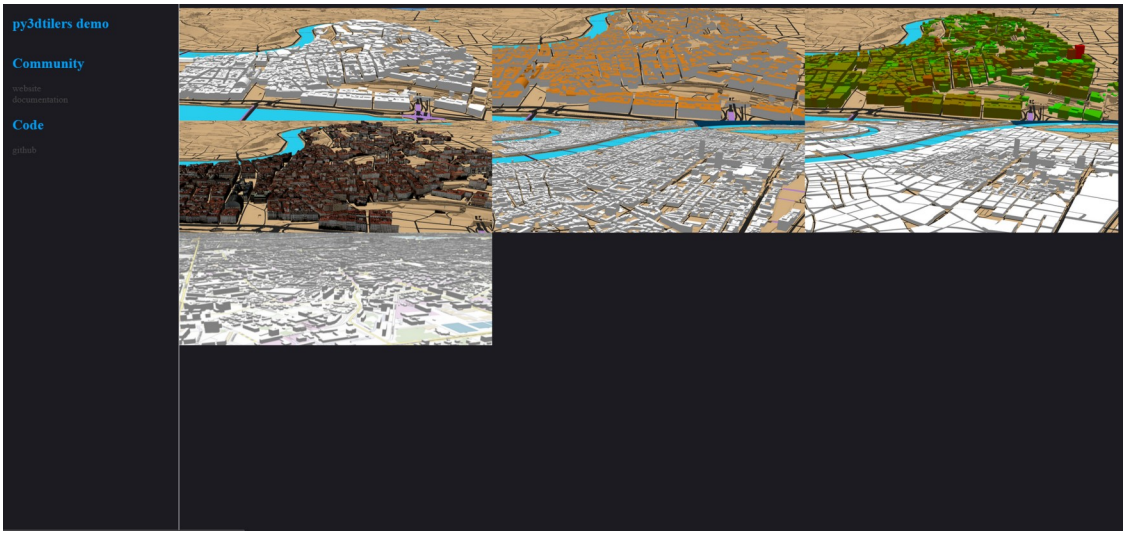


Figure 4 : Mosaïques des différentes démonstrations développées grâce à l'outil Py3tilers. Une démonstration de la ville texturée, une autre avec de la couleur, la ville avec différents niveaux de détail..

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.

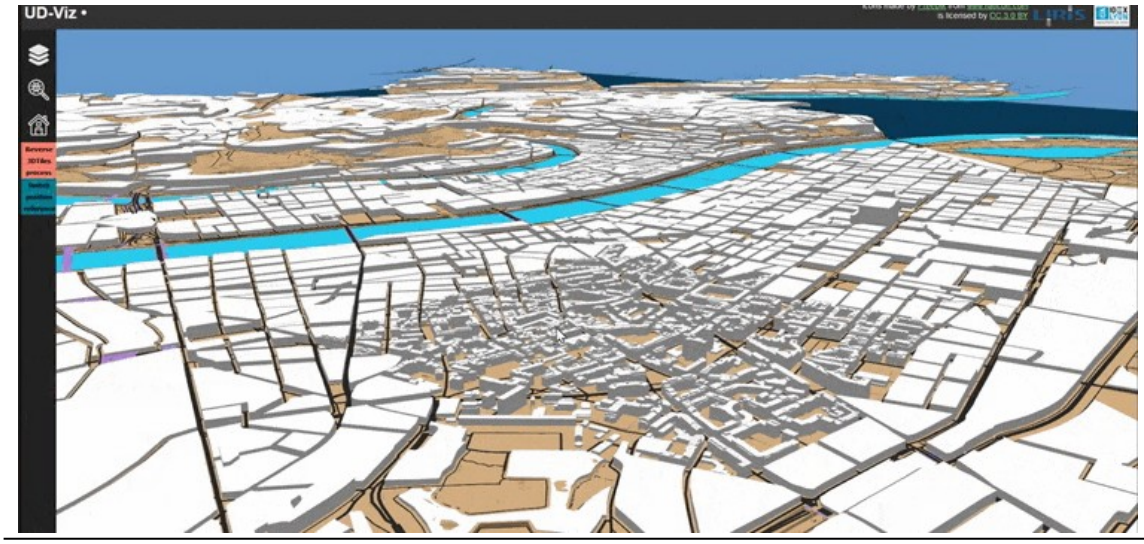


Figure 5 : Démonstration montrant le raffinement du niveau de détail des bâtiments de Lyon en fonction de la position de la souris

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.

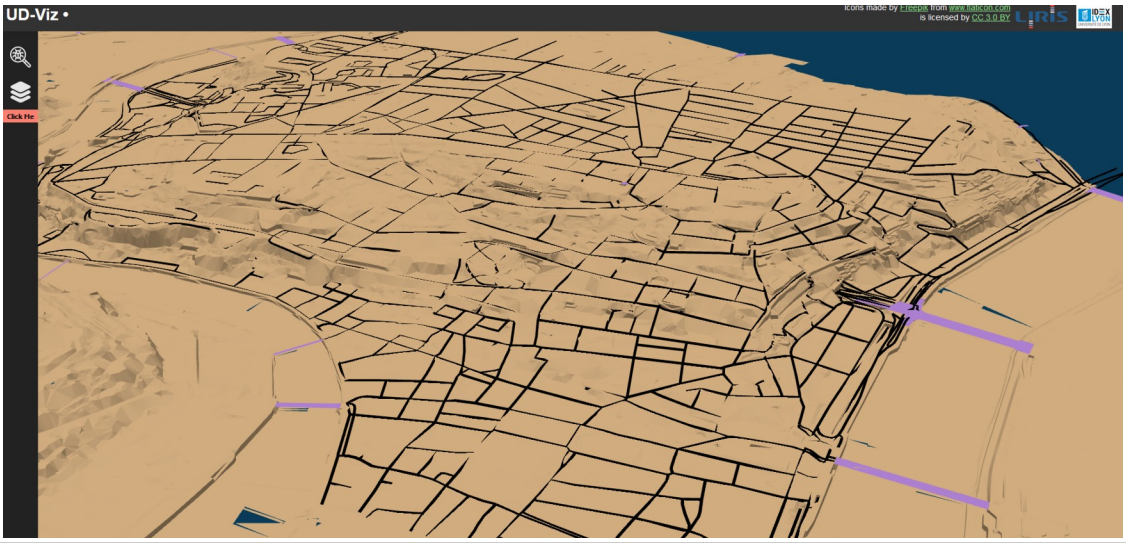


Figure 6 : Démo UD-Viz sur le calcul des hauteurs de routes contenu dans un fichier GeoJSON à l'aide d'un glisser/déposer dans l'application web UD-Viz

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.

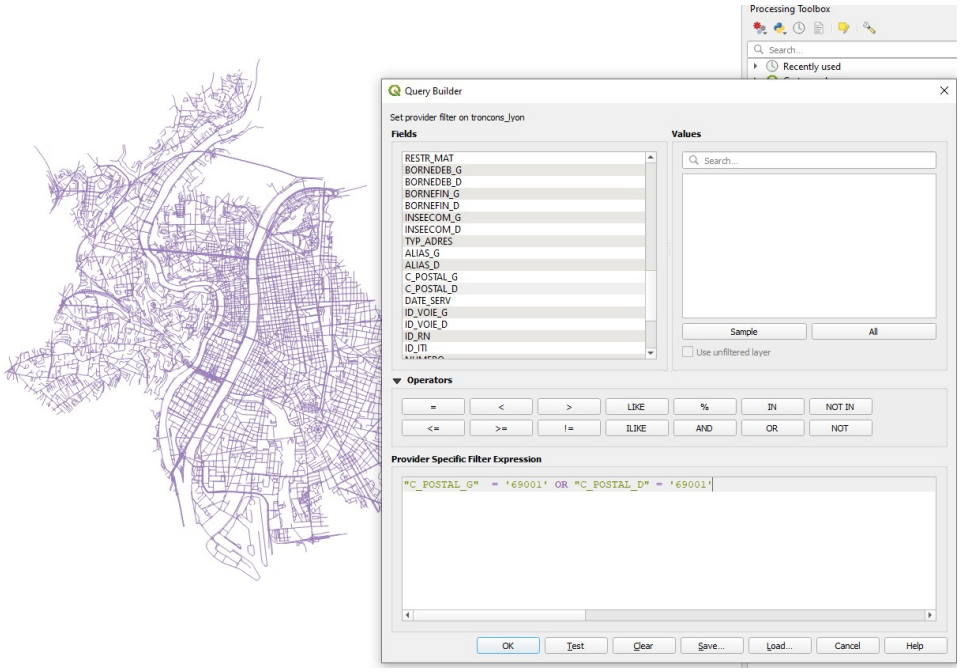
Geojson Tiler

Roads

Download the BD Topo data from IGN

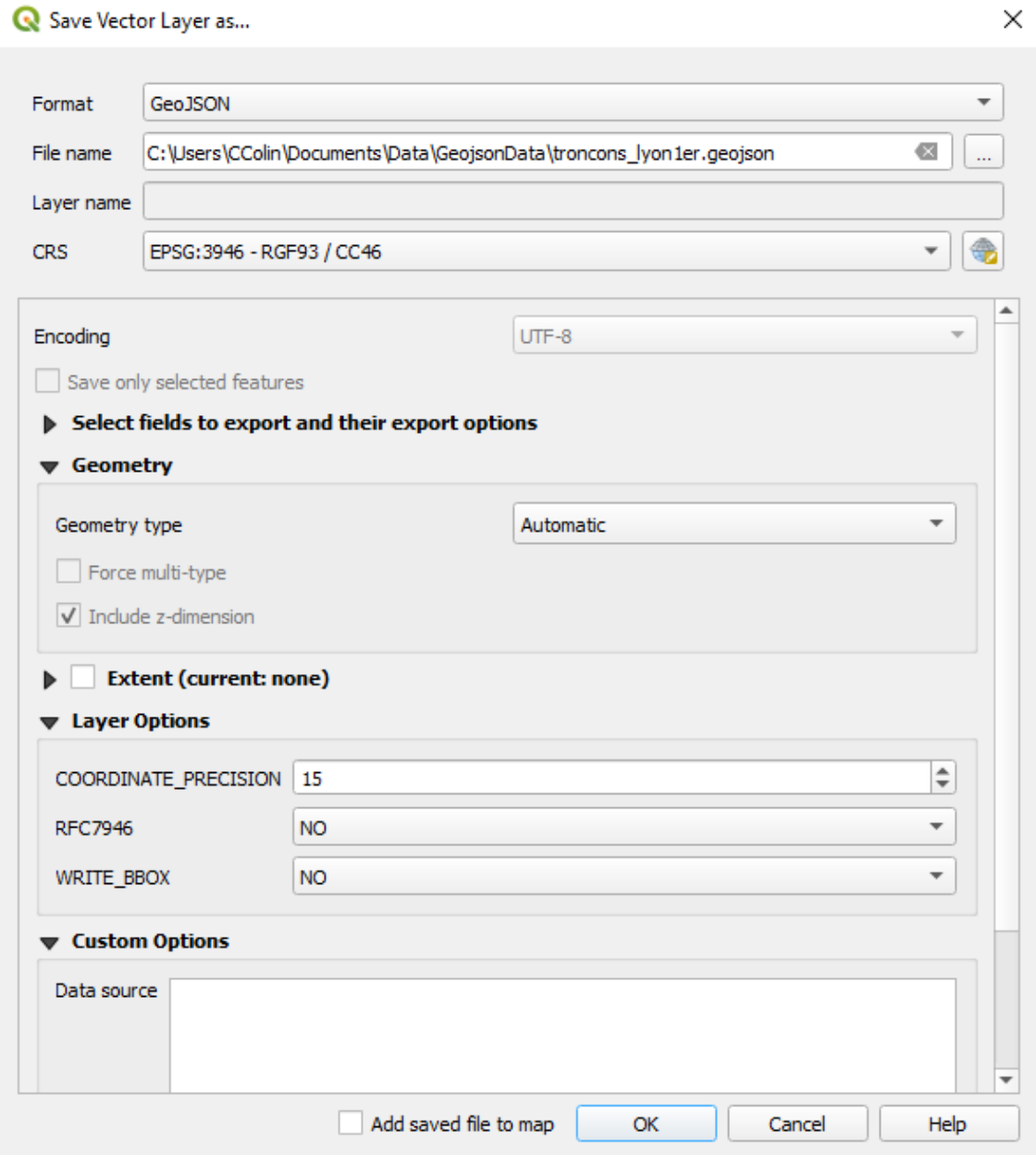
In QGIS, open the *BDTOPO/1_DONNEES_LIVRAISON/TRANSPORT/**TRONCON_DE_ROUTE.shp*** file.

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`.

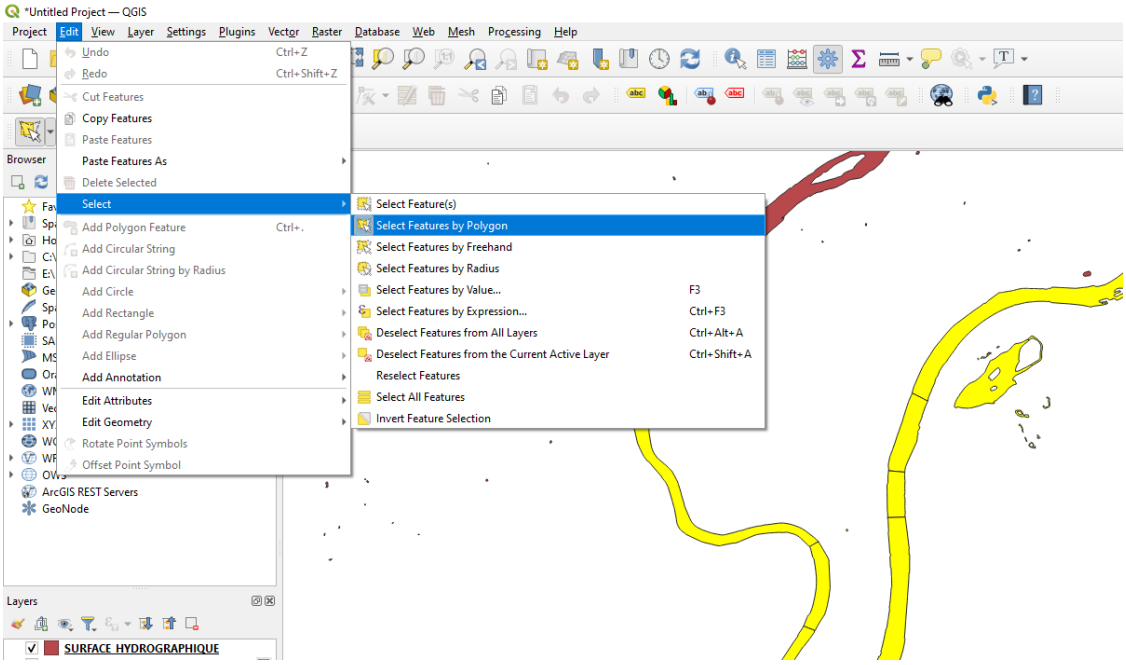
See the GeojsonTiler README for more information on usage.

Water bodies

Download the BD Topo data from IGN

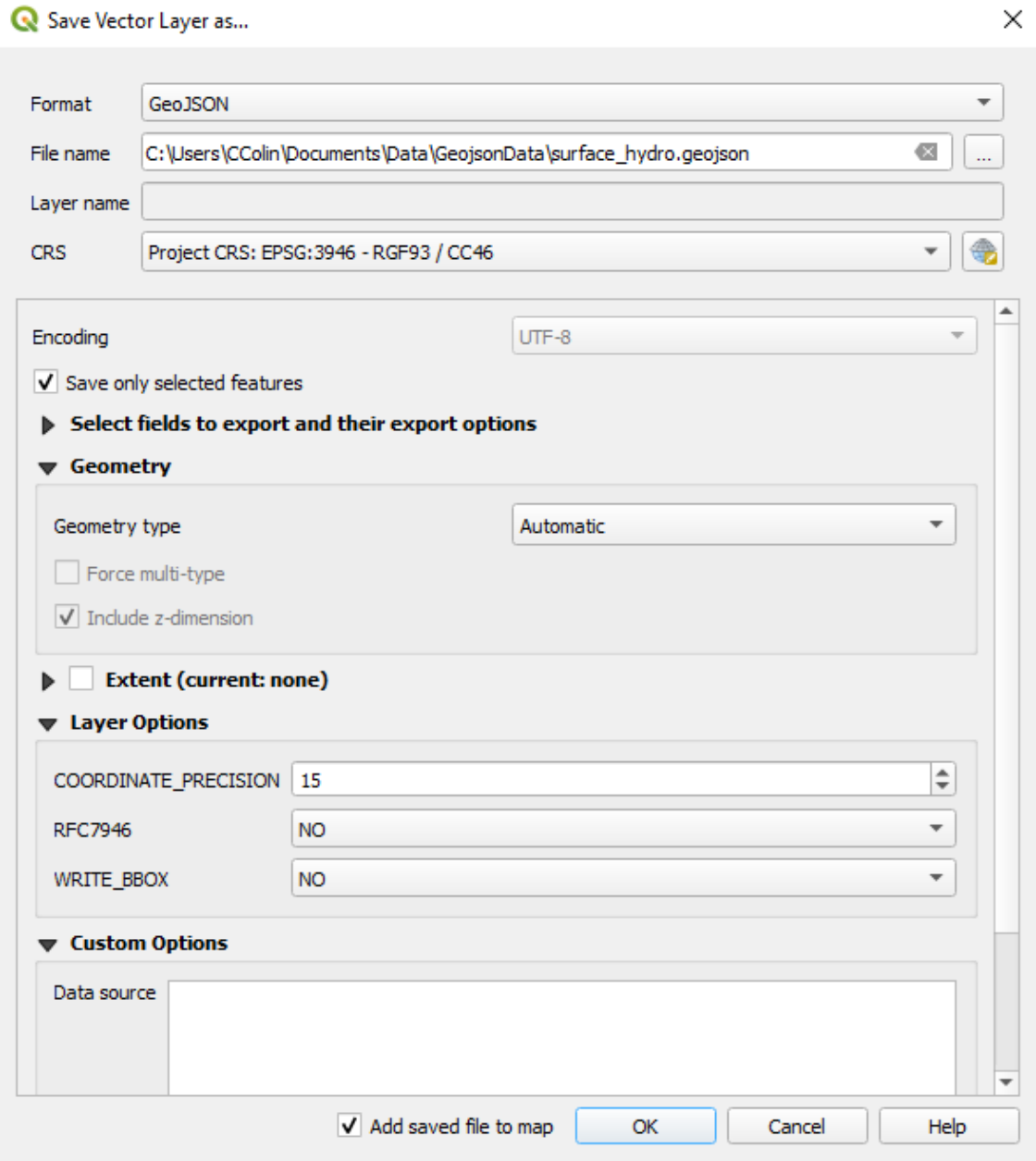
In QGIS, open the *BDTOPO/1_DONNEES_LIVRAISON/HYDROGRAPHIE/**SURFACE_HYDROGRAPHIQUE.shp*** file.

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.

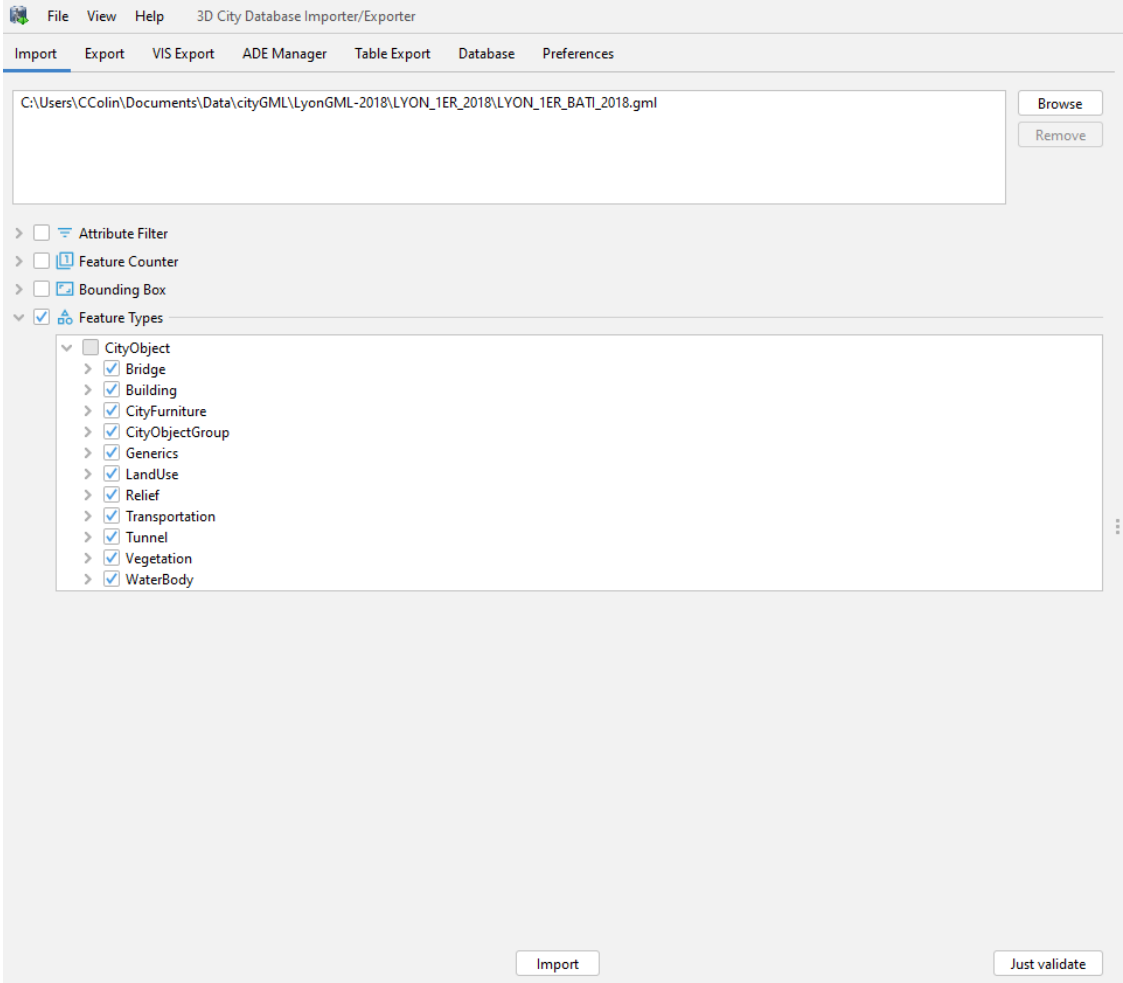
CityGML Tiler

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
```

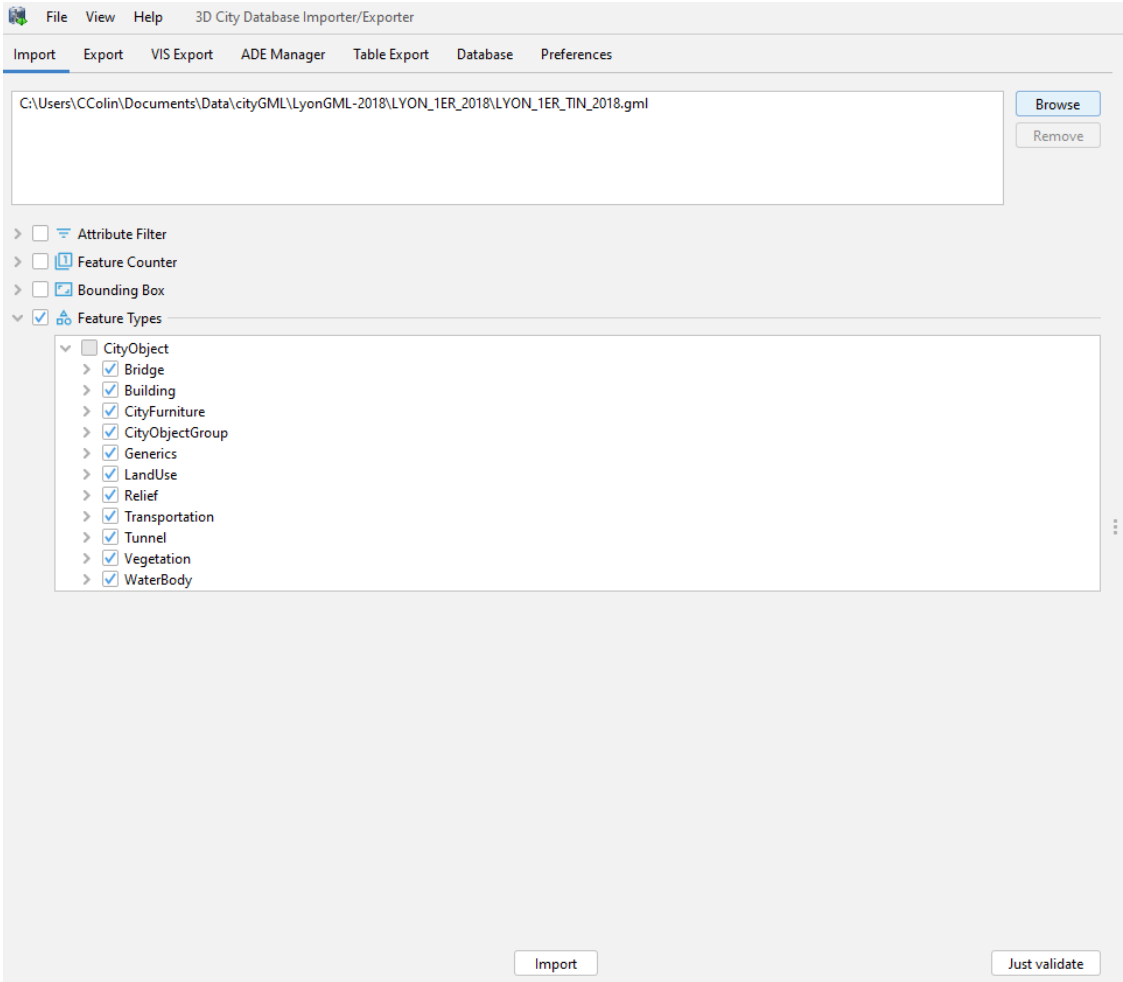
To create LOA, you can for example use *BDTOPO/1_DONNEES_LIVRAISON/ADMINISTRATIF/**ARRONDISSEMENT.shp*** from BD Topo (IGN). To be able to use it, export the .shp as GeoJson with QGIS (the projection must be the same as buildings, i.e EPSG:3946 most of the time for Lyon's data).

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

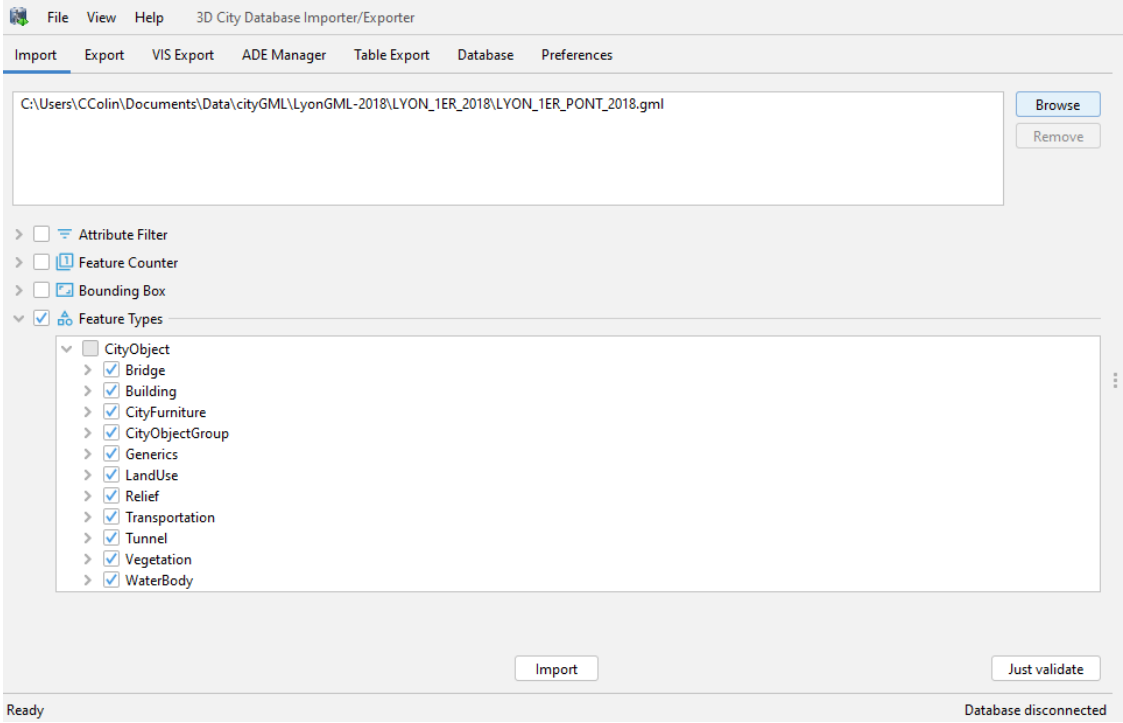
To use the Tiler, target your database config file and choose the type `relief` (see the CityTiler usage for more details):

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.