



ÉCOLE NATIONALE
DES SCIENCES
GÉOGRAPHIQUES

École Nationale des
Sciences Géographiques



Université Gustave Eiffel



INSTITUT NATIONAL
DE L'INFORMATION
GÉOGRAPHIQUE
ET FORESTIÈRE

Institut Géographique
National

Rapport d'analyse - Lecture de CityGML pour iTowns

Axel Dumont

Avril 2024

Sommaire

Glossaire et sigles utiles	4
Introduction	5
1 Reformulation du besoin	6
1.1 Les objectifs de l'étude	6
1.2 Les acteurs	6
1.3 Les contraintes	6
2 Analyse fonctionnelle - solution proposée	7
2.1 Détail des grandes fonctionnalités	7
2.2 Analyse UML	8
3 Étude technique	10
3.1 Langages et outils utilisés	10
3.2 Données : quantité, qualité, formats, erreurs	10
4 Réalisation et suivi du projet	12
4.1 Les risques	12
4.2 Phasage du projet	12
4.3 Travail réalisé	12
4.4 Évolutions techniques	13
4.5 Résultat	13
Conclusion	16

Table des figures

2.1	Diagramme de cas d'utilisation de la pipe	7
2.2	Diagramme de séquence de l'application	8
2.3	Diagramme de composants de la pipeline	9
3.1	Diagramme de classe lié à la végétation issus de la documentation de 3DCityDB	10
3.2	Exemple des données utiles dans le traitement des géométries implicites (Point d'an-crage + Matrice 4x4)	11
3.3	Extrait de la table implicit geometry du cas d'études	11
4.1	Résultat finaux de la visualisations sur iTowns	14
4.2	Résultat finaux de la visualisations sur iTowns (sans géométrie implicite)	14
4.3	Résultat finaux de la visualisations sur iTowns (avec filtre sur les reliefs)	15

Glossaire et sigles utiles

.b3dm Batch 3D Model

.obj Wavefront 3D Objects

3DCityDB Base de Donnée 3DCity

ENSG École Nationale des Sciences Géographiques

IGN Institut Géographique National

RDI Recherche Développement Imagerie et LIDAR

TSI Technologies des Systèmes d'Information

WKB Well-Known Binary

Introduction

Le **projet de fin d'étude** du Master Technologies des Systèmes d'Information (TSI) s'inscrit dans un contexte d'application des connaissances et compétences acquises au cours de la formation. Dans le cadre de ce projet j'ai intégré l'équipe du département Recherche Développement Imagerie et LIDAR (RDI) de l'Institut Géographique National (IGN) sous la tutelle d'Antoine Lavenant et Quentin Bouillaguet. Ce projet se déroule sur 5 semaines, du 21 Mars au 30 Avril 2024. Ce dernier consiste en la mise en place d'une pipelines permettant la lecture de fichier au format CityGML dans l'outil **iTowns**. iTowns est une librairie de développement open source permettant de concevoir des applications de visualisation et d'interaction avec des données géographiques 2D et 3D à travers le web (cf [site de l'IGN](#)). Actuellement iTowns ne lis que des fichiers au formats 3DTiles or certaines données 3D de l'IGN étant produites sous le format CityGML cette pipeline permettra ainsi aux futurs utilisateurs de visualiser leurs données directement par le biais de leur navigateur web.

Ce rapport d'analyse vous fournira une analyse détailler du fonctionnement de la pipelines ainsi que des pistes d'optimisation pour la suite.

1. Reformulation du besoin

1.1 Les objectifs de l'étude

À terme la pipeline devra permettre à l'utilisateur de lire ses données CityGML en utilisant iTowns de manière simple et intuitive. Ainsi la pipeline devra effectuée de manière automatique les différentes étapes nécessaires à la lecture de ce format de données.

Afin de pouvoir réaliser cette tâche dans les délais imposés on s'appuiera sur [Py3dTilers](#). Ce repository servira donc de base au développement du projet. Ce dernier permet ainsi la conversion de donnée CityGML en 3DTiles par le biais d'une Base de Donnée 3DCity (3DCityDB) que l'on hébergera sur une base PostgreSQL incluant l'extension PostGIS.

On ne notera ici qu'un seul cas d'usage qui est celui d'un utilisateur voulant visualiser ses données CityGML par le biais de l'outil iTowns.

1.2 Les acteurs

Les différent·e·s acteur·ice·s de ce projet sont :

- les commanditaires du projet : Antoine LAVENANT et Quentin BOUILLAGUET qui sont les interlocuteurs principaux de ce projet et qui définiront de manière AGILE les différentes tâches à effectuer au fur et à mesure de l'avancement du projet et qui définissent les besoins de l'utilisateur.
- les différents acteurs du RDI : apporteront des réponses à certaines mes interrogations notamment techniques vis à vis de l'environnement de l'IGN.
- moi : mon rôle est de réaliser l'analyse technique du projet et de l'implémenter.

1.3 Les contraintes

La principale contrainte est intrinsèquement liée aux délais imposé. En effet, au vue des délais il m'est impossible de réaliser un script de conversion de toute pièce, ainsi je dois analyser l'existant, ici Py3dTilers, afin de pouvoir y ajouter les fonctionnalités nécessaires au projet.

De plus le format de donnée de sortie afin d'utilisé iTowns doit impérativement être du 3DTiles.

Enfin, Py3dTilers ne prend pas en charge la conversion des géométries implicites contenues dans les données CityGML. En effet, les géométries implicites ont pour particularité de ne pas avoir leur géométrie intégrée à la donnée, ce sont des géométries communes à plusieurs objets 3D présents sur une scène. Ces géométries sont stockées à part dans des fichier 3D tel que .3ds pour notre cas d'étude. Ainsi il sera nécessaire de géoréférencer ces objets 3D à partir des données présentes dans le CityGML tel que leur point d'ancrage ainsi qu'une matrice de transformation 4x4.

2. Analyse fonctionnelle - solution proposée

2.1 Détail des grandes fonctionnalités

Les grandes fonctionnalités ont été définies en fonction du cas d'usage. Voici un recensement des grandes fonctionnalités que nous pouvons envisager. Bien sûr, chaque utilisateur·ice peut y voir un intérêt différent.

Fonctionnalité	Détail
Déposer un fichier CityGML	L'utilisateur doit pouvoir placer un fichier CityGML dans la 3DCityDB
Convertir un fichier CityGML en 3DTiles	La pipeline permet de convertir totalelement (géométries implicites comprises) un fichier CityGML en 3DTiles
Visualiser ses données CityGML	La pipeline doit permettre au final à l'utilisateur de visualiser directement ses données sur iTowns

Afin de représenter l'orchestration entre les grandes fonctionnalités et acteur·ice·s de la pipeline, il est possible d'utiliser un diagramme de cas d'utilisation, comme sur la figure 2.1

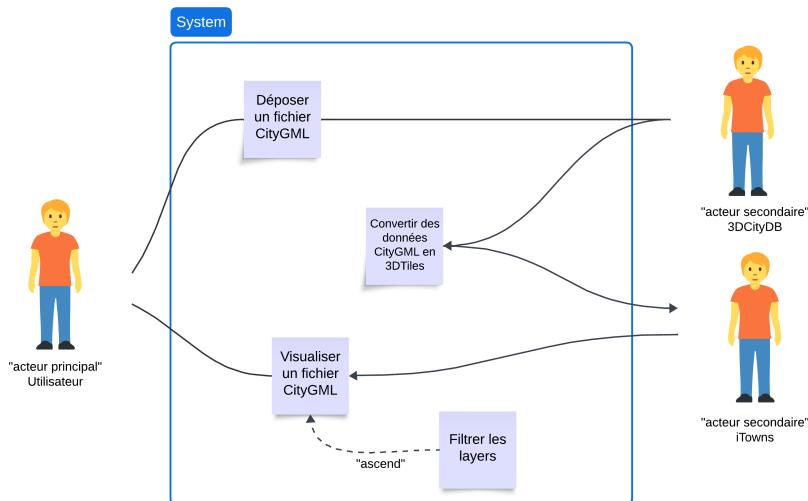


FIGURE 2.1 : Diagramme de cas d'utilisation de la pipe

Nous avons aussi effectué un diagramme modélisant la vie de l'application, de façon chronologique, il s'agit du diagramme de séquence ci-dessous.

Cinq "objets" interagissant ensemble ont été identifiés dans le fonctionnement de la pipeline. On notera que pour l'utilisateur la manipulation reste la même, en effet, les données implicites contenues dans les données de l'utilisateur sont traités différemment des données explicites mais cette dissociation ne devra pas affecter la manipulation de l'utilisateur qui veut simplement visualiser ces données.

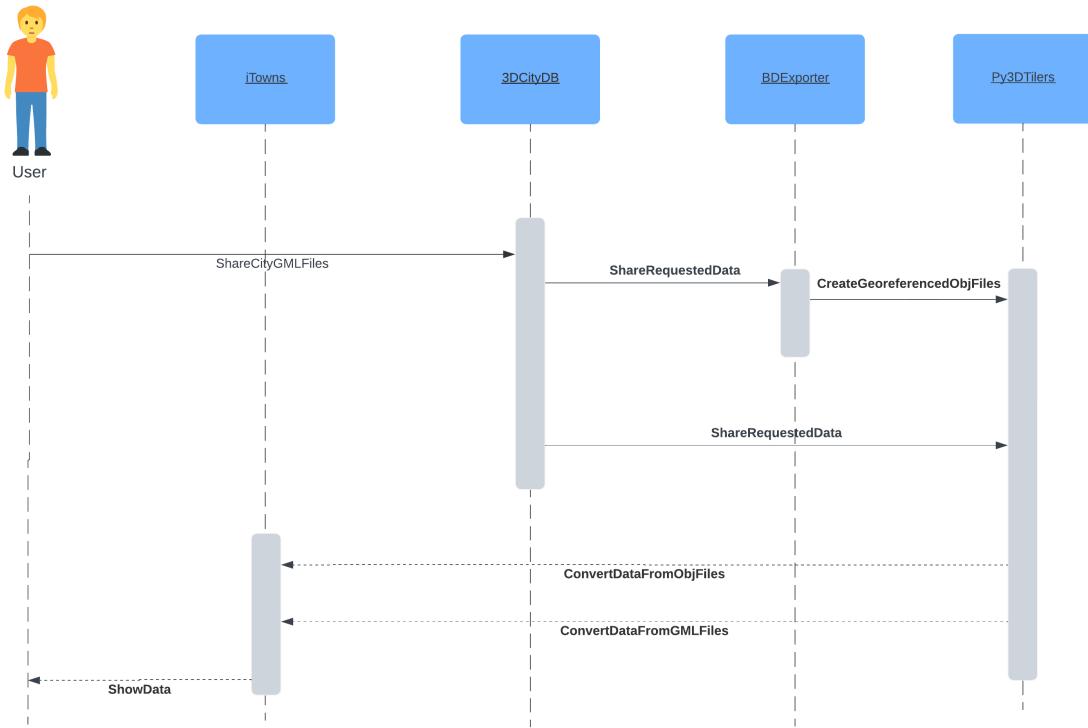


FIGURE 2.2 : Diagramme de séquence de l'application

2.2 Analyse UML

Grâce au diagramme des cas utilisations on peut visualiser l'orchestration des fonctionnalités et des acteur·ice·s ainsi que clarifier les tâches devant être effectuées par l'application.

Il est possible de représenter les différents composants de la pipeline ainsi que les liens entre eux par un diagramme de composants, représenté sur la figure 2.3.

2. Analyse fonctionnelle - solution proposée

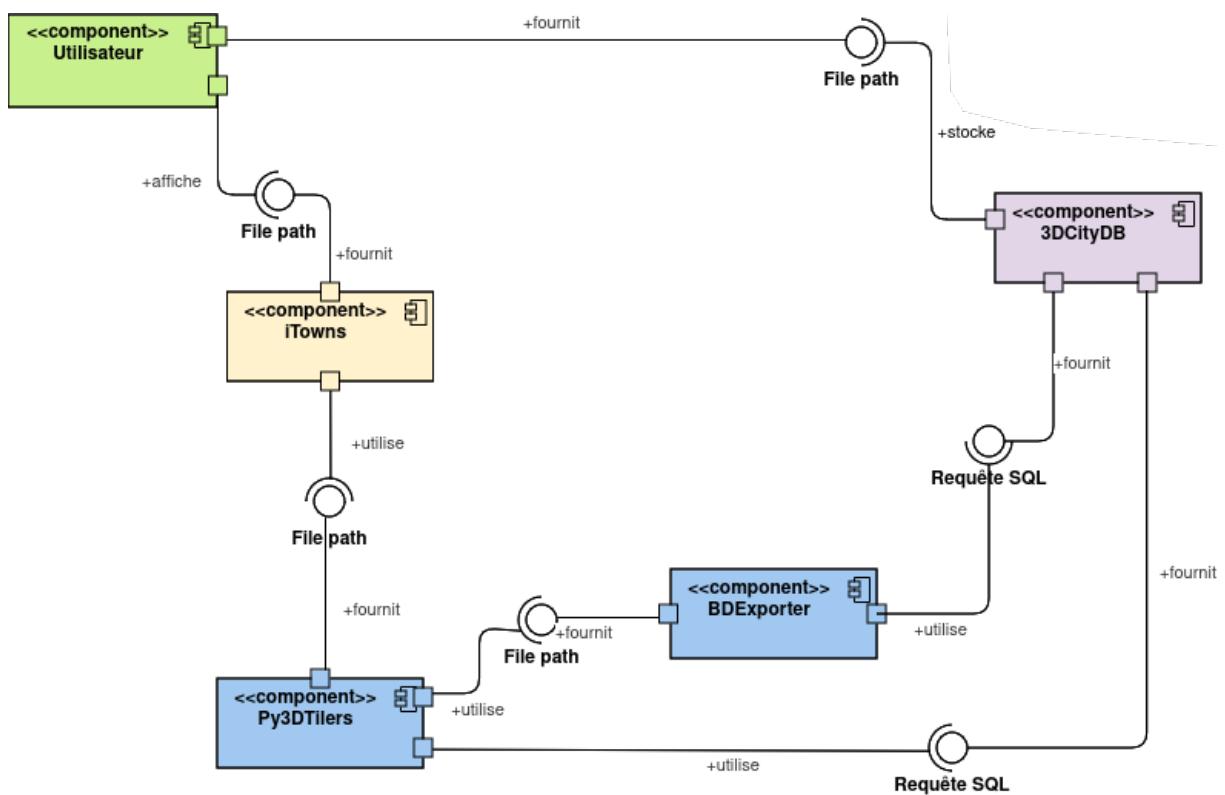


FIGURE 2.3 : Diagramme de composants de la pipeline

3. Étude technique

3.1 Langages et outils utilisés

Le choix des langages de programmation utilisés durant le projet s'est fait en fonction des outils utilisés.

Tout d'abord, l'outil central de cette pipeline est [Py3dTilers](#). Cette outil, même si incomplet à la base au vue des besoins, permet de convertir des données 3D contenues dans un fichier au format Wavefront 3D Objects (.obj) ainsi que des données au format CityGML contenues dans une 3DCityDB en 3DTiles (format de donnée contenant un fichier tileset.json ainsi que plusieurs fichier Batch 3D Model (.b3dm)). Afin de répondre aux besoins du projet il est donc nécessaire d'apporter des ajouts à [Py3dTilers](#) étant donné qu'il ne prend en compte que certains thèmes du CityGML et ne prend pas en charge les géométries implicites.

Ainsi, de part l'utilisation de [Py3dTilers](#) il est obligatoire de mettre en place un 3DCityDB. D'après la documentation de 3DCityDB il est conseillé de mettre en place une base de données **PostgreSQL** muni de l'extension **PostGIS** afin d'y installer 3DCityDB.

Ensuite, pour ce qui est des langages de programmation utilisés j'ai utilisé des langages liés aux outils utilisés ainsi que des langages facilitant l'automatisation. Ainsi, pour ajouter des éléments à [Py3dTilers](#) j'ai utilisé du **python3**, pour les transitions entre les différentes étapes et exécutions des scripts python le langage **bash** a été utilisé (en raccord avec les commandes mises en places par Py3dTilers) et enfin du javascript afin de configurer iTowns.

3.2 Données : quantité, qualité, formats, erreurs

Pour ce projet la donnée d'entrée est au format CityGML et sera stockée dans la 3DCityDB et donc sous un certain format. Afin de ne pas trop surcharger ce rapport en expliquant comment chaque type de donnée est formater dans la base de donnée voici le diagramme des classes issus de la [documentation de 3DCityDB](#) lié à la végétation (cf figure 3.1). Ce cas est intéressant car simple et permettant de mettre en évidence la notion de géométrie implicite.

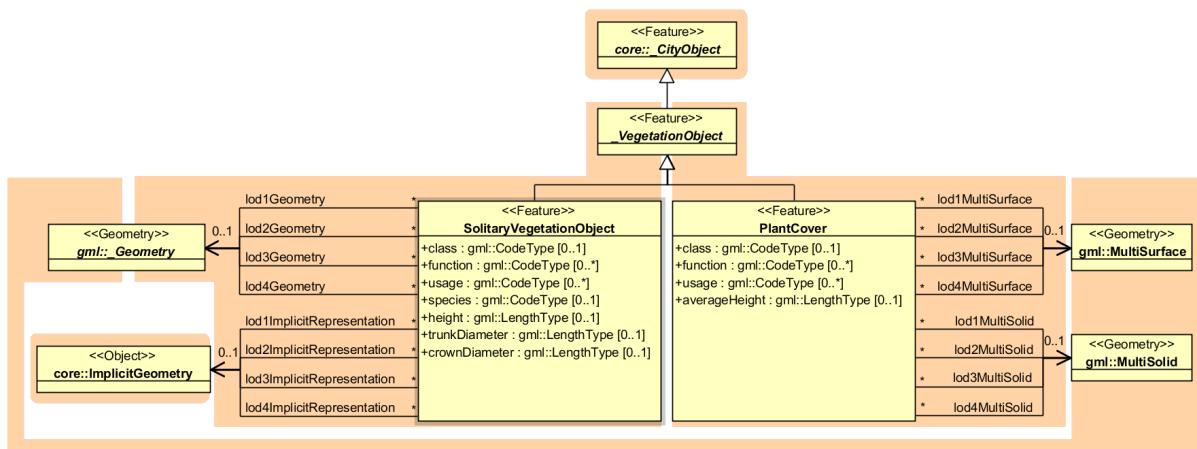


FIGURE 3.1 : Diagramme de classe lié à la végétation issus de la [documentation de 3DCityDB](#)

3. Étude technique

Comme dit précédemment dans 1.3, les géométries implicites sont présentes sous forme de "lien" avec un objet 3D présent dans un fichier en annexe du CityGML. Cependant, le CityGML contient les coordonnées du point d'ancrage de cette géométrie en Well-Known Binary (WKB) ainsi qu'une matrice de transformation 4x4 afin de géoréférencer la géométrie (cf figure 3.2). On a ainsi une table implicit geometry qui référence toute les géométries implicites comme l'on peut le voir sur la figure 3.3.

	Data Output	Messages	Notifications
1	01010000A0E610000091F2933A38392B410F2899CC520557410EBE30992AC03840	[null]	[null]
2	01010000A0E6100000554AC22846392841363CB3652D55741D6390664AFFE3840	[null]	[null]
3	01010000A0E61000007348ABF1C5C928415825585A4FD55741F35487DC0C1F3840	[null]	[null]
4	01010000A0E6100000107233EC4339284108556AE5E00557416C09FA067383940	[null]	[null]
5	01010000A0E6100000CD5915392841C93E768C1D55741053E10B198B3840	[null]	[null]
6	01010000A0E61000005CE8C1A654928410B8031EC51D557419B82ZFC03EA238..	[null]	[null]
7	01010000A0E61000003C8A3825C5C928417654353305D557419F12936A973E940	[null]	[null]
8	01010000A0E6100000601A99BF2828410D003F75CD45741336DFFCA4AA83E..	[null]	[null]
9	01010000A0E61000002499FCB7E39284163081FB4ED55741B809CAF5AC039..	[null]	[null]
10	01010000A0E61000008B28CE523928410551F77150D5574134D769AA4A523940	[null]	[null]
11	01010000A0E6100000CEAACF356D3928412A3A923951D557410B293F9F6813840	[null]	[null]
12	[null]	[null]	[null]
13	01010000A0E61000001B8178D78382841DEB0601B8D545741F085C954C105140	[null]	[null]
14	[null]	[null]	[null]
15	[null]	[null]	[null]
16	[null]	[null]	[null]
17	[null]	[null]	[null]
18	[null]	[null]	[null]
19	[null]	[null]	[null]
20	[null]	[null]	[null]
21	01010000A0E6100000E9C11A55A382B41A9C134E6C0D45741FCC6D79E59B240..	[null]	[null]
22	01010000A0E6100000DC2E3407823828417E5704BD0C8D45741D210C4CE148E40..	[null]	[null]
23	01010000A0E61000002026F0F4A382841A4C2D8C4BCD4574110E9B7AF039F40..	[null]	[null]
24	01010000A0E6100000120A724E50382841F54A594C8D045741342E1C08C9964040	[null]	[null]

FIGURE 3.2 : Exemple des données utiles dans le traitement des géométries implicites (Point d'ancrage + Matrice 4x4)

	Data Output	Messages	Notifications
1	1	[null]	[null]
2	2	[null]	[null]
3	3	[null]	[null]
4	4	[null]	[null]
5	5	[null]	[null]
6	6	[null]	[null]
7	7	[null]	[null]
8	8	[null]	[null]
9	9	[null]	[null]
10	10	[null]	[null]
11	11	[null]	[null]
12	12	[null]	[null]
13	13	[null]	[null]
14	14	[null]	[null]
15	15	[null]	[null]
16	16	[null]	[null]
17	17	[null]	[null]
18	18	[null]	[null]
19	19	[null]	[null]
20	20	[null]	[null]
21	21	[null]	[null]

FIGURE 3.3 : Extrait de la table implicit geometry du cas d'études

4. Réalisation et suivi du projet

4.1 Les risques

Le principal risque de ce projet est temporel. En effet, étant seul pour réaliser ce projet le temps risque de manquer afin de rendre un produit totalement finit et optimisé. Les données CityGML étant des données relativement volumineuse il faudra donc bien adapté les méthodes de tests à ce type de donnée afin de ne pas perdre de temps lors de traitement trop long pour obtenir un résultat peu convenable. De plus, la gestion AGILE du projet permettra d'aller droit au but et d'accomplir des tâches de manière méthodique ainsi que de diriger ce dernier.

4.2 Phasage du projet

Le projet a été divisé en 4 phases, la 4ème étant la plus chronophage. Ces phases ayant été ajoutées au fur et à mesure de l'avancement du projet par Antoine Lavenant. À chaque fin de phase d'avancement du projet nous faisions un point afin de savoir ce qui était réalisable pour la suite du projet et mettre en place les prochaines tâches à effectuer.

La 1ère phase constituait à prendre connaissance de l'existant et des outils avec lesquels j'ai été ammené à travailler tel que Py3DTilers, 3DCityDB et iTowns. La 2ème phase de ce projet consistait à convertir les données CityGML fournit par le commanditaire en 3DTiles. Ensuite, la 3ème phase constituait en la prise en main d'iTowns afin de visualiser les données convertit précédemment. Après avoir fait l'observation que Py3DTilers ne prennait pas en compte les géométries implicites contenues au sein des données CityGML il a donc été décidé pour la 4ème phase de mettre en place une manière de convertir et visualiser ce type de données.

4.3 Travail réalisé

Voici le travail réalisé durant ce projet, en plus de l'analyse :

- Création de scripts python venant s'ajouter à Py3DTilers afin de prendre en charge le plus de thèmes de données présents en CityGML possible ;
- Création de scripts bash et python permettant la prise en charge des géométries implicites ;
- Création d'un script javascript permettant la visualisation et le filtrage des données à l'aide d'iTowns ;
- Création d'un repository GitHub contenant les différents script ainsi qu'un README.md faisant guise de tutoriel ;
- Création d'un jeu de données 3DTiles sur Marseille à partir du CityGML fournit.

4.4 Évolutions techniques

Bien que le code soit fonctionnel et que le résultat soit présent on ne peut dire que ce dernier est efficient. En effet, de part l'utilisation de Py3DTilers qui ne prend pas en compte les géométries implicites il a fallu ajouter une étape intermédiaire à la conversion. Or, cette étape consiste en la création d'objet 3D géoréférencé sous forme de fichiers .obj ce qui crée un fichier de transition volumineux et de manière chronophage car une fois la création des .obj fait il faut les convertir en 3DTiles. Cet conversion de .obj à 3DTiles peut se faire par le biais de Py3DTilers. Cependant cette opération ne peut être réalisée en une fois, il faut traiter .obj par .obj ce qui ralonge fortement le traitement puisque pour chaque objet le script bash va devoir ouvrir puis fermer python en exécutant la commande issus de Py3DTilers.

Ainsi une première piste intéressante d'optimisation serait d'apporter des modifications au script de Py3DTilers afin de pouvoir traiter tous les fichiers .obj d'un seul coup et ainsi obtenir un fichier 3DTiles peu volumineux en peu de temps. En effet, cette mise à niveau permettrait ainsi au script bash de n'avoir à ouvrir python qu'une seule fois pour tout le traitement. De plus, actuellement le résultat est volumineux car la fonctionnalité de merging de Py3DTilers ne prend en compte que le tileset.json du 3DTiles et non les .b3dm, on a ainsi un .b3dm par objet 3D.

Une seconde piste d'amélioration, qui ici concerne l'ergonomie de la pipeline, serait d'intégrer la fonctionnalité de lire des CityGML directement au sein d'iTowns. Cette option permettrait de faciliter la visualisation pour l'utilisateur qui pourrait passer directement par le biais de son navigateur web.

Une dernière amélioration serait de faire le lien entre les objets créés et les informations complémentaires du CityGML afin de pouvoir réaliser du picking par le biais d'iTowns. Cela peut-être fait via le nom des fichiers de transitions, en effet, j'ai fait en sorte que le nom des fichiers créés fassent apparaître l'id de l'objet dans la base 3DCityDB afin de pouvoir refaire le lien.

4.5 Résultat

Malgré les défauts d'efficiencies notés précédemment le résultat est bien présent et qualitatif. En effet, le CityGML est bien convertit dans sa quasi intégralité, géométrie implicite comprise. De plus, ce dernier est visualisable en utilisant iTowns et les informations non implicites peuvent être "picker" directement.

4. Réalisation et suivi du projet

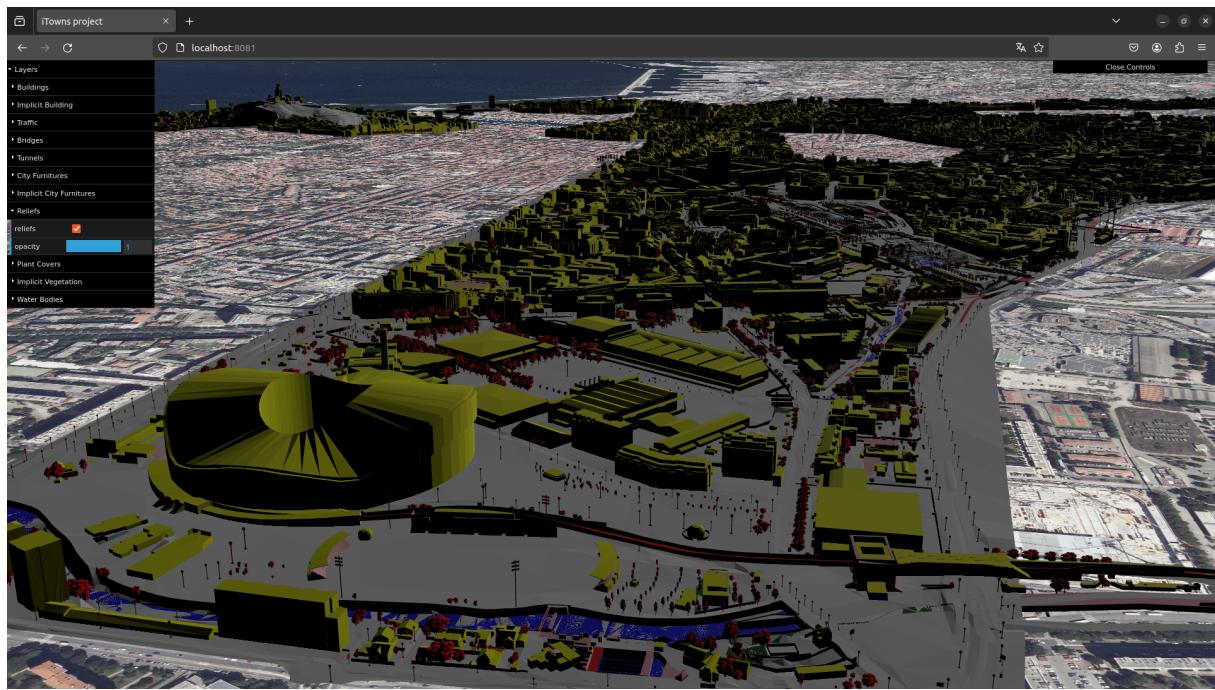


FIGURE 4.1 : Résultat finaux de la visualisations sur iTowns

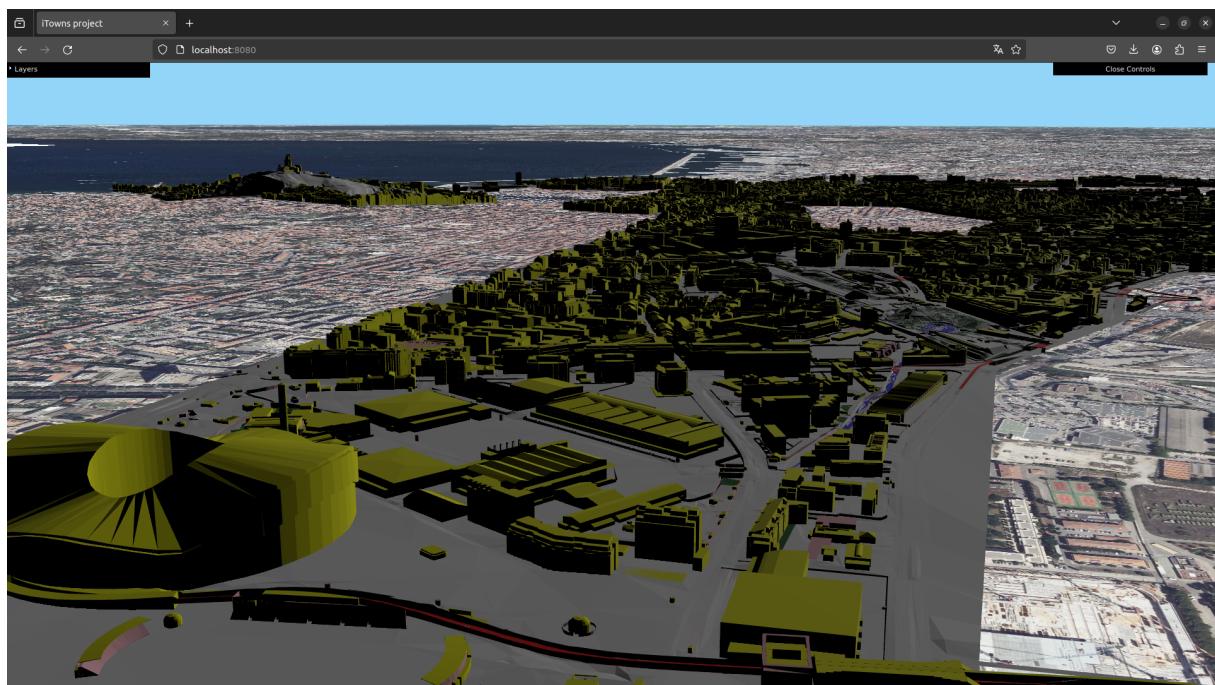


FIGURE 4.2 : Résultat finaux de la visualisations sur iTowns (sans géométrie implicite)

4. Réalisation et suivi du projet



FIGURE 4.3 : Résultat finaux de la visualisations sur iTowns (avec filtre sur les reliefs)

Conclusion

Le travail réalisé a permis de mettre en place une méthode de conversion des CityGML afin de permettre leur visualisation sur un navigateur web grâce à iTowns en passant par une 3DCityDB et Py3DTilers, malgré des améliorations nécessaires sur l'efficience.

Ce projet m'a permis de mettre en pratique bon nombre de compétences apprises lors de la formation TSI tel que le web 3D, le développement bash en passant par l'utilisation d'une base donnée géographiques 3D, consolidant ainsi ces compétences.