****

**Documentation technique**

Projet de visualisation des façades de bâtiments en 3D

**Projet co-réalisé par :**

Defauconpret Arthur

Marechal Alexandre

**Stratégie Globale**

L’objectif principal du projet est de mettre en place un algorithme capable de calculer les profils des façades de bâtiments de part et d’autre d’une rue donnée. La méthode employée consiste à projeter des rayons orthogonaux à l’axe de la route (et de z constant) puis à calculer leurs intersections avec les bâtiments de la rue.

L’algorithme prend en entrée des multipolygones 3D, qui correspondent à des bâtiments et une polyligne qui correspond à la rue étudiée. Les données utilisées dans ce projet correspondent aux entités « batiment » et « troncon\_de\_route » de la BD TOPO® de l’institut national de l’information géographique et forestière.

Concernant la sortie, l’algorithme génère deux profils de façades sous la forme de fichiers au format image (JPG) correspondant aux profils de chacun des côtés de la route. Dans le projet, toutes les valeurs de distances sont exprimées en mètres.

L’algorithme est quant à lui organisé en 5 parties principales. La première partie consiste à lire les données. Deux méthodes sont possibles, selon si l’utilisateur peut ou non accéder au service WFS. Dans un second temps, l’algorithme sélectionne les bâtiments retenus pour le calcul. Il génère ensuite les rayons à partir de la polyligne de départ, puis calcule les intersections entre ces rayons et les bâtiments. Enfin, il génère l’image de sortie via le module openCV.

## **Les applications possibles**

Le BTP, la protection du patrimoine historique et l’urbanisme sont des domaines où la visualisation 3D de certains éléments – et plus particulièrement des façades – est devenu un moyen simple de représenter des objets sous forme de projet réaliste. De plus, la visualisation 3D est devenu un outil incontournable, notamment dans le BTP où elle permet de réaliser des mesures directement sur le modèle.

# **Détail de l’algorithme**

## **Entrées utilisateur**

* json\_file (version hors connexion) : chemin du fichier json contenant les entités polygonales correspondant aux bâtiment, au format <string>.
* bbox (version avec connexion) : emprise de la zone de travail définie par les points A et B, au format longitude-latitude <lat\_a,long\_a,lat\_b,long\_b>.
* rue : polyligne de la rue étudiée, au format <list>.
* file\_path : chemin du dossier d'écriture des images de sortie et nom du fichier (sans extension, elle est ajoutée automatiquement), au format <string>.
* buffer : distance (en mètres) entre la route et le bâtiment en dessous de laquelle le bâtiment sera pris en compte dans le calcul de silhouette <float>.
* precision\_min : précision minimale (en mètres). Sera équivalent à la résolution de la silhouette de sortie sur l'axe des z et des x <float>.
* hauteur\_max : hauteur maximale jusqu'à laquelle les points « origines » seront calculés. Elle est exprimée en hauteur relative par rapport à la route.

## **Partie 1 : Lecture des données (WFS/JSON)**

Dans ce projet, le choix a été fait de laisser 2 possibilités à l’utilisateur pour récupérer les données. La première méthode consiste à récupérer les données vectorielles de la BD TOPO® via le service WFS (Web Feature Service) de l’IGN selon une emprise de donnée, notée bbox. L’algorithme extrait les coordonnées des différents bâtiments et tronçons qui sont contenus dans la bbox. À partir du dictionnaire, seules les géométries sont donc récupérées et organisées sous forme de liste (utilisation de la fonction « get\_street\_feature\_from\_url » et « get\_bat\_feature\_from\_url »). L’inconvénient de cette méthode est la nécessité d’avoir accès à une connexion internet stable, ainsi qu’au serveur proxy si nécessaire.

La deuxième méthode ne nécessite cette fois pas de connexion (sauf lors du téléchargement). L’algorithme va cette fois directement aller lire dans le fichier JSON, téléchargé préalablement via le service WFS de l’IGN et stocké sur le disque de l’ordinateur. Cette opération se fait grâce à la fonction « get\_feature\_from\_json ».

## **Partie 2 : Sélection des bâtiments pris en compte dans le calcul de la silhouette.**

Afin de sélectionner les bâtiments susceptibles de concourir au calcul de la silhouette, plusieurs méthodes sont possibles :

**Dans la première méthode**, chaque polygone présent dans l’emprise (ou « bbox »), est associé au segment de la polyligne dont il est le plus proche. Il est possible d’obtenir cette association en réalisant au préalable le calcul des barycentres des polygones présents dans l’emprise définit au début de l’algorithme.

Cette méthode possède l’avantage que l’association entre les bâtiments et les segments prend la forme d’une relation de 1 à n, ce qui implique que le nombre de calculs à réaliser serait moins important et que l’algorithme serait plus rapide. Cependant, elle possède de nombreux inconvénients car sans buffer, l’algorithme calculera les intersections avec tous les polygones de la bbox, ce qui peut augmenter la durée de calcul dans certains cas. De plus, dans le cas où la rue choisie ne serait pas rectiligne, il est possible que plusieurs segments soient face à un même bâtiment. Ce bâtiment ne sera donc pris en compte dans les calculs que pour le segment auquel il aura été associé (Figure 1). Cela peut donc engendrer la présence de vides artificiels dans l’image de sortie.

**La deuxième méthode** consiste cette fois à sélectionner dans la bbox une liste de bâtiments « candidats » à l’aide d’un buffer. Puis pour chaque rayon issu de la polyligne, on cherche de potentielles intersections avec tous les polygones candidats.

L’avantage de cette méthode est qu’en utilisant un buffer, seuls les bâtiments de la bbox susceptibles d’être intersectés seront pris en compte lors du calcul. Il n’y a donc pas de risque de créer un trou dans la silhouette puisqu’un même bâtiment peut être "vu" depuis différentes portions d’une même rue. Cependant, cette méthode demande un nombre plus important de calculs puisqu’il y a une relation de n à n entre les segments et les bâtiments. De plus, une portion de façade peut être enregistrée plusieurs fois depuis deux tronçons de rues différents (Figure 2). Cela a donc pour effet d’allonger artificiellement la silhouette en doublant l’affichage de cette portion de façade.

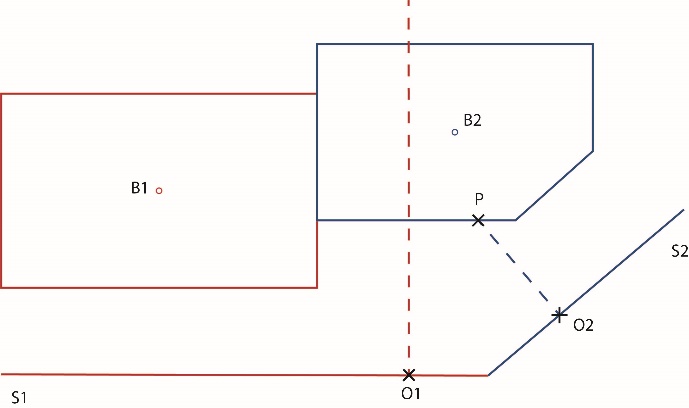
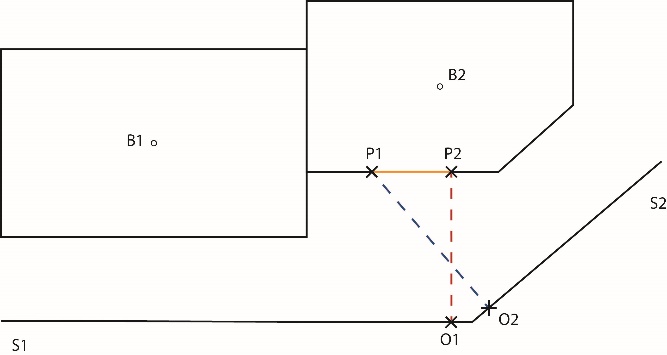
****

Figure 2: Calcul d’intersection avec la méthode 2. La portion définie par le segment [P1;P2] apparaîtra deux fois dans la façade.

Figure 1: Calcul d’intersection avec la méthode 1. Le rayon issu de O1 n’intersectera jamais le bâtiment B2, créant un vide dans la façade.

Pour cet algorithme, la deuxième méthode a été choisie car malgré la nécessité d’une certaine puissance de calcul pour parcourir les éléments, elle permet d’éviter la présence de vides artificiels dans l’image de sortie. De plus, quelle que soit la méthode employée, la silhouette de sortie sera nécessairement déformée car les rayons sont orthogonaux à la rue. Or, les segments de la rue n'ayant pas tous la même direction, les rayons ne sont pas parallèles entre eux. La distance entre deux intersections n'est donc pas régulière ce qui implique une déformation le long de l'axe des abscisses sur l'image de sortie.

Dans cette partie, l’algorithme va donc dans un premier temps calculer les barycentres de chaque polygone via la fonction « barycentre\_liste\_poly ». Puis l’algorithme va s’occuper de découper la polyligne en segment (« polyligne\_to\_segment »), pour permettre d’associer les barycentres à chacun des segments. Par la suite, la fonction « distance\_batiment\_polyligne » va renvoyer, pour chaque élément, un conteneur où chaque polygone est associé à sa distance au segment le plus proche de la polyligne choisie. À partir d’un buffer on peut ensuite récupérer les bâtiments dans la zone maximum définie par l’utilisateur. Enfin, la fonction « liste\_bat\_candidat » va permettre à l’algorithme de déterminer quels sont les polygones candidats à la génération du profil de la rue.

## **Partie 3 : Calcul des rayons**

Dans cette partie, les coordonnées des origines sont d’abord calculées au niveau de la route par la fonction « def\_base\_rayons » selon la résolution choisie par l’utilisateur. Par la suite, l’algorithme permet d’obtenir les images par une suite de projections le long de l’axe z avec la fonction « def\_origines\_rayons ». Le nombre d’origine dépend donc de la résolution et de la hauteur maximum choisie par l’utilisateur. Les équations des rayons sont calculées à partir de ces origines.

## **Partie 4 : Calcul des intersections entre rayons et bâtiments**

Dans la BD TOPO® de l’IGN, les polygones correspondent aux toitures des bâtiments. Lorsqu’une géométrie a été obtenue par "interpolation bâti BDTopo", ses sommets ont tous un z identique. Ce n'est pas le cas lorsque les objets sont issus de la photogrammétrie. Ces objets sont minoritaires et aucun de ce type n’est enregistré dans la bbox choisie pour tester le code. Puisque les toits sont « plat », on décide donc de ne pas prendre en compte d’éventuelles inclinaison des toitures lors du calcul de la silhouette. Cela permettra de simplifier les calculs d'intersections entre les rayons et les bâtiments, en ignorant temporairement la dimension z.

Il est donc nécessaire de calculer les équations de chaque polygone de la liste (en ignorant le Z) via les fonctions « calc\_equa\_poly » et « calc\_equa\_list\_poly ». Pour chaque rayon issu de la polyligne, l’intersection est calculée avec tous les segments de tous les polygones candidats. Pour chaque intersection, on détermine grâce à la fonction intersection\_rayon\_batiments, si elle est située à gauche ou à droite de la rue, de manière à construire deux profils distincts en sortie de l’algorithme. Enfin, via la fonction liste\_distance, l’algorithme va calculer la distance entre l’origine du rayon projeté et l’intersection et la stocker dans une liste en sortie de la fonction « liste\_distance » qui permettra la création du profil de la rue sélectionné par l’utilisateur.

## **Partie 5 : Création de l’image de sortie**

À partir de la liste des résultats de la fonction « liste\_distance », l’algorithme fait appel aux fonctions « create\_distance\_array » et « matrice\_distance » pour remettre en forme la liste obtenue à la fin de la partie 4, et ainsi calculer la matrice des distances entre origines et intersections sous la forme d’un tableau. La structure de la matrice des distances possèdera la même structure que l’image de sortie en fin d’algorithme. Les valeurs dans la matrice sont ensuite converties en valeurs de pixel (entre 0 et 255) via la fonction « conversion ». Enfin, la fonction « silhouette\_tab\_pixel » va permettre de calculer le tableau de pixel pour les deux côtés de la rue.

Pour terminer, l’écriture de l’image se fait avec la fonction cv2.imwrite du module cv2. Comme pour l’importation des données, deux fonctions d’écritures existent en fonction de la méthode d’importation choisie lors de la première partie. Si la méthode d’importation passe par le WFS, il sera nécessaire d’utiliser la fonction « calcul\_silhouette\_url ». Dans le cas où la méthode d’importation se fait par les JSON (hors connexion) stockés sur le disque, il sera nécessaire d’utiliser « calcul\_silhouette\_json ». Les entrées de la fonction ont été décrite dans la partie « Entrées utilisateur ».

Le fichier JPEG est ensuite écrit sur le disque. Ces étapes sont réalisées pour les deux listes de résultats correspondant aux deux côtés de la rue.

Temps exécution du code estimé :

* résolution 101 m : 10-1
* résolution 100 m : 101 s
* résolution 10-1 m : 103s
* résolution 10-2 m :