Segmentation manuelle de nuage de points au sein d'un système d'information géographique web 3D

Auteur : Delsart, Benjamin
Promoteur(s) : Hallot, Pierre; Kasprzyk, Jean-Paul
Faculté : Faculté des Sciences
Diplôme : Master en sciences géographiques, orientation géomatique, à finalité spécialisée en géomètre-expert
Année académique : 2019-2020
URI/URL : http://hdl.handle.net/2268.2/9143

Faculté des sciences
Département de géographie

Segmentation manuelle de nuage de points au sein d'un système d'information géographique web 3D

Mémoire présenté par : **Benjamin DELSART**

pour l'obtention du titre de

Master en sciences géographiques, orientation géomatique et géométrologie

Président de jury :
Pr. **René WARNANT**

Promoteur :
Pr. **Jean-Paul KASPRZYK**
Pr. **Pierre HALLOT**

Année académique : 2019/2020
Date de défense : Janvier 2020

Remerciements

Mes premiers remerciements sont dédiés à mes promoteurs, Mr. Hallot et Mr. Kasprzyk, pour m'avoir guidés et conseillés durant ce long travail.

Je tiens également à remercier Mr. Nys, toujours disponible pour répondre à mes questions.

Je remercie mes proches et mes amis pour leur soutien morale au quotidien.

Un grand merci à Emmanuelle Moyart pour son soutien inconditionnel.

Enfin, un merci particulier à mes parents et à Mathieu Tits, qui ont toujours cru en moi tout au long de mes études.

Résumé

Un système d'information géographique (SIG) utilise des objets vectoriels pour représenter, analyser et stocker des phénomènes géographiques discrets. Pour exploiter un nuage de points au travers une analyse spatiale, un travail de segmentation est nécessaire. Ce traitement permet de structurer un nuage de points en éléments logiques décrivant simplement l'environnement représenté. La recherche scientifique s'attache à développer des algorithmes automatiques de segmentation. Cependant, l'automatisation présente des limites, et reste un procédé long et fastidieux.

Ce travail cherche donc à développer un SIG web 3D, comprenant un outil de segmentation manuelle permettant de produire des objets vectoriels 3D afin de permettre une analyse spatiale ultérieure.

La réalisation de l'application, avec son architecture et les requêtes implémentées, est détaillée au sein de la méthodologie. Le résultat obtenu est un SIG web 3D, permettant de digitaliser des objets vectoriels 3D à partir de nuage de points et d'encoder manuellement de l'information sémantique, en incluant Potree, PostGIS et GeoJSON.

Abstract

A geographic information system (GIS) uses vector objects to represent, analyze and store discrete geographic phenomena. To exploit a point cloud through spatial analysis, a segmentation work is required. This processing makes it possible to structure a point cloud into logical elements that simply describe the represented environment. Scientific research is focused on developing automatic segmentation algorithms. However, automation has limits, and remains a long and tedious process.

This work therefore seeks to develop a 3D web GIS, including a manual segmentation tool to produce 3D vector objects for subsequent spatial analysis.

The realization of the application, with its architecture and implemented queries, is detailed within the methodology. The result is a 3D web GIS, allowing to digitize 3D vectorial objects from point clouds and to manually encode semantic information, including Potree, PostGIS and GeoJSON.

Table des matières

Table des figures

1 Introduction

Les nuages de points permettent une représentation discrète aussi bien de petites entités, tels que des artefacts ou un bâtiment, que d'environnement du monde réel comme une ville voire un pays. Ceux-ci peuvent être générés en un temps restreint (Plusieurs dizaines de milliers de points à la seconde), de manière précise (De l'ordre de quelques mm) et à un faible coût grâce à diverses technologies d'acquisitions, comme la photogrammétrie ou la lasergrammétrie. Ainsi, ces données sont devenues un support pour un nombre varié d'applications dans des domaines tels que : l'urbanisme pour contextualiser des plans dans leurs environnements et rendre les logiques du plan sous-jacent explicites (Dambruch et Krämer, 2014) ; l'archéologie pour documenter le patrimoine bâti menacé (Billen et al., 2018) ; l'industrie de la construction pour améliorer le rendement des projets, de la conception à la construction (Wang et Kim, 2019) ; etc. Il existe donc une forte demande pour stocker, gérer, traiter et explorer des nuages de points 3D afin d'exploiter tout leur potentiel et de fournir une représentation non filtrée et détaillée des sites capturés.

Pour pouvoir utiliser un nuage de points, par nature non structuré, comme outil de communication et document technique, un travail de préparation est nécessaire pour les rendre exploitables (Poux et al., 2014). En effet, lors de l'acquisition d'un nuage de points, de l'information spatiale, les coordonnées tridimensionnelles, et éventuellement une information colorimétrique et thermale sont associées à chaque point levé. Cependant, l'information sémantique liée à l'environnement représentée n'est pas récoltée. Cette information est pourtant primordiale pour structurer un nuage de points en élément logique permettant de décrire la scène relevée. Pour pouvoir, par la suite, réaliser une analyse et une interprétation des données, un nuage de points sémantiquement riche doit être disponible. Il est donc nécessaire de procéder à une phase de traitement : la segmentation.

Dans le cadre d'un projet de recherches archéologiques se déroulant au sein du palais de Coudenberg, mené par le département de géomatique de l'ULG, plusieurs scans 3D ont été

réalisés pour numériser le site. Pour réaliser une analyse spatiale du site numérique au sein d'un système d'information géographique (SIG) 3D, des outils appropriés doivent être fournis pour recueillir des données spatiales et les associer à des données attributaires. En effet, dans un SIG des objets vectoriels (Point, ligne, et polygone) sont utilisés pour représenter, analyser et stocker des phénomènes géographiques discrets. L'objectif du présent travail est donc de stocker des objets vectoriels 3D digitalisés à partir d'un nuage de points pour permettre une analyse spatiale ultérieure.

Le premier chapitre de ce travail s'attache à présenter toutes les composantes nécessaires à la mise en place d'un SIG web 3D. L'état d'avancement de la recherche dans le développement d'algorithmes de segmentation est également abordé. L'hypothèse de travail est formulée au sein du second chapitre. Les étapes ayant permis l'implémentation de l'application sont détaillées dans le chapitre méthodologie. Une démonstration du fonctionnement de l'application permettant de souligner ses faiblesses et ses forces est réalisée dans le chapitre validation. Pour finir, une synthèse du travail et les perspectives de développement de l'application clôtureront ce travail.

2 Etat de l'art

Ce chapitre présente la littérature scientifique en lien avec les différentes fonctionnalités requises pour la mise en place d'un SIG web 3D permettant une segmentation de nuage de points.

2.1 Système d'information géographique

2.1.1 Définition

D'après Burrough (1986), un système d'information géographique (SIG) est un système informatique capable de saisir, stocker, analyser et d'afficher de l'information géographique. Bolstad (2002) définit un SIG comme un système informatisé d'aide à la collecte, à la maintenance, au stockage, à l'analyse, à la production et à la diffusion de données et d'informations spatiales. Selon Ian (2010), un SIG est un outil qui stocke les données géographiques, les récupère et les combine pour créer de nouvelles représentations de l'espace. Il fournit des outils d'analyse spatiale permettant à des utilisateurs, spécialisés dans des domaines variés, de s'organiser dans leur projet. Au sein de la littérature scientifique, on peut retrouver de nombreuses définitions d'un SIG. Celles-ci variant selon le domaine de recherche (Cowen,1990 ; Longley et al., 2005 ; Rigaux et al., 2001).

2.1.2 Composantes et fonctions d'un SIG

Un SIG est constitué de cinq éléments (Reddy, 2018). Premièrement, le matériel informatique avec le système informatique sur lequel le logiciel SIG va pouvoir fonctionner. L'ordinateur est l'élément central du système informatique. Le matériel d'acquisition (Ex : scanner, satellite, etc.) constitue la principale source d'entrée de l'information géographique. Tandis que les interfaces web, qui ont progressivement remplacé les imprimantes, constituent la principale source de sortie. Deuxièmement, le logiciel SIG (Ex : QGIS, GRASS, ARCGIS, etc.) fournit les fonctions et les outils nécessaires à l'analyse et à l'affichage de l'information géographique.

Troisièmement, les données spatiales et attributaires sont stockées et organisées au sein d'un système de gestion de base de données (SGBD) comme par exemple PostgreSQL. Les données peuvent être issues de l'organisation même ou être achetées auprès d'un organisme extérieur. Quatrièmement, les personnes possédant une expertise dans la gestion d'un SIG peuvent concevoir, maintenir et analyser les données d'un SIG en lien avec des tâches diverses. Enfin, les méthodes essentielles au bon fonctionnement d'un SIG basées sur des règles bien conçues que sont le modèle et les pratiques opérationnelles propres à chaque organisation.

Les fonctions attendues d'un SIG sont : l'acquisition des données ; la mise à jour et la gestion des données acquises ; la production d'information géographique à travers l'interprétation des données collectées ; la présentation de l'information produite (Ex : web mapping, impression de cartes, etc.) ; l'intégration et la conversion des données (Ex : données vectorielles et rasters, changement de système de projection, etc.).

2.1.3 Système d'information géographique Web

Le web ou WWW est un ensemble de documents hypertextes qui sont liés à d'autres documents localisés dans des ordinateurs situés partout dans le monde (Ingram, 1995). Un ensemble de protocoles est utilisé par le WWW pour permettre l'échange de fichiers et de documents entre ordinateurs identifiés par une adresse IP unique (Stubkjaer, 1997).

De nombreuses organisations sont impliquées dans le développement de standards pour le web tel que le World Wide Web Consortium (W3C), ISO/TC 211, et l'Open Geospatial Consortium (OGC). Le W3C se charge de fournir les informations nécessaires pour la compatibilité des technologies du web (Ex : HTML5, XML, CSS, SOAP, etc.), ISO/TC 211 développe des standards internationaux dans le domaine du SIG et de la géomatique, et l'OGC produit des standards orientés vers les services web géographiques (Ex : WMS, WMTS, WFS et WCS) et les formats de données géographiques (GML et KML, détaillé au point 5.1.) pour répondre aux exigences d'ISO/TC211.

Le développement rapide du web a permis d'ajouter une nouvelle dimension à l'évolution des SIG. L'association des SIG avec la technologie d'Internet a permis un accès interactif aux données géospatiales, une intégration et une transmission des données en temps réel ainsi qu'un accès à des outils d'analyse SIG à un public large.

Agrawal et Gupta (2017) ont réalisé un historique des étapes majeures qui ont marqué l'évolution des SIG web. Sans rentrer dans les détails, il est nécessaire de citer la sortie en 1999

du Web 2.0 qui a conduit à un bouleversement dans l'utilisation du Web et des différentes disciplines liées dont le SIG web. En effet, grâce à cette évolution de nombreuses applications comme Google map, Google Earth et Microsoft Bing Map ont pu faire leur apparition. En 2000, Refractions Research a sorti PostGIS, une extension du SGBD PostgreSQL qui permet la manipulation d'objets spatiaux. En 2005, la sortie de la technologie AJAX a permis de développer des applications web et des sites web côté client. Google maps est un exemple-type utilisant cette technologie (Fu et Sun, 2010). Enfin, dans la même année, la sortie de Google Earth a été un événement majeur au sein des SIG web 3D (Yu et Gong, 2012). Par la suite, d'autres moteurs d'affichage de données 3D ont été développés et seront détaillés au sein du chapitre du point 2.3 "Affichage de données 3D : moteur de rendu de nuage de points".

2.1.4 Architecture d'un SIG web

De manière générale, une application web non-spatiale a un navigateur web comme client qui envoie une requête au serveur web qui répond à la requête. Dans le cas des SIG web, il est nécessaire de disposer d'un serveur additionnel appelé serveur de données permettant de fournir des outils SIG pour la gestion de données spatiale. Il existe plusieurs types d'architecture SIG web : l'approche 3-tiers logique, *plug-and-play*, *service oriented architecture* (SOA) (Yang et al., 2010) et *cloud computing* (Yang et al., 2011).

L'approche 3-tiers logique peut être divisée en trois approches différentes (Agrawal et Gupta, 2017) :

— Architecture client léger (« thin client ») : dans cette configuration (Voir figure 2.1), le client à un rôle minime et la plupart du traitement est réalisé côté serveur. Lorsqu'un client formule une requête, le serveur construit une réponse souvent sous une forme simple comme une carte générée à partir de la base de données spatiales. Dans ce cas, le client ne peut pas directement appeler le serveur SIG ce qui nécessite un interpréteur comme le Common Gateway Interface (CGI). Le serveur peut également utiliser les Servlet qui sont des programmes Java. Ces Servlet sont plus efficaces que les CGI car ils peuvent gérer plusieurs requêtes simultanément. Il existe également d'autres options technologiques comme les API (Application Programming Interface), les ASP (Active Server Pages), et les JSP (Java Server Pages). Cette architecture permet un affichage de données rasters ou de données vectorielles rastérisées. De plus, des données vectorielles peuvent être interrogées à travers le protocole WMS et WFS. Les services géographiques, Web Map Service (WMS) et Web Feature Service (WFS), fournis par

l'OGC, sont des standards d'architecture client léger. Dans le cas du WMS, l'information géographique est produite sous forme d'un fichier image digital, une carte, qui est affichée à l'écran dans un format image de type svg, png, gif ou jpeg. Le Web Feature Service (WFS) permet à un client d'accéder à de l'information géographique en format GML à partir de plusieurs WMS à la fois. En conclusion, cette première approche présente plusieurs avantages. Le client n'a aucune responsabilité. La mise à jour des données est aisée étant donné que le serveur contient toute l'information. Enfin, le principal désavantage réside dans le peu de fonctionnalités que possède le client. Par exemple, pour simplement zoomer, le client effectue une requête vers le serveur.

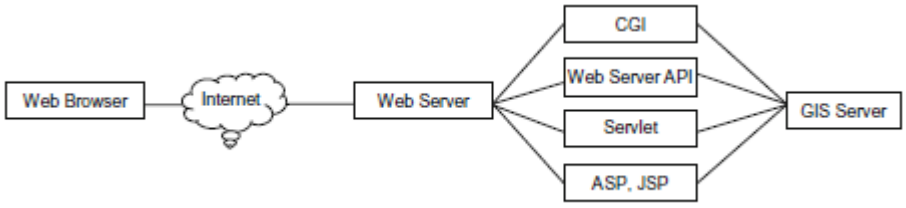


FIGURE 2.1: Architecture client léger (Agrawal et Gupta, 2017).

— Architecture client lourd (« thick client ») : au sein de cette architecture, comme illustré par la figure 2.2 ci-dessous, le client est plus puissant dans le sens où les capacités du navigateur web sont augmentées par des plug-ins, applets, et Active X . Cependant, ces technologies sont progressivement devenues obsolètes, et ont été remplacées par le langage informatique dynamique JavaScript. Ce langage, interprétable par le navigateur web du client, offre la plupart des fonctionnalités de Java et de nombreuses API sont écrites dans ce langage (Ex : leaflet, open-layers, three.js, jQuery, google maps, etc.). Ainsi, les données brutes peuvent être interprétées directement côté client, comme par exemple pour être affichées selon des représentations géométriques. L'avantage de cette architecture est que le client possède un rôle plus important, ce qui permet au serveur de ne pas être réquisitionné sans arrêt, donc certains traitements peuvent être réalisés aussi bien côté client que côté serveur.

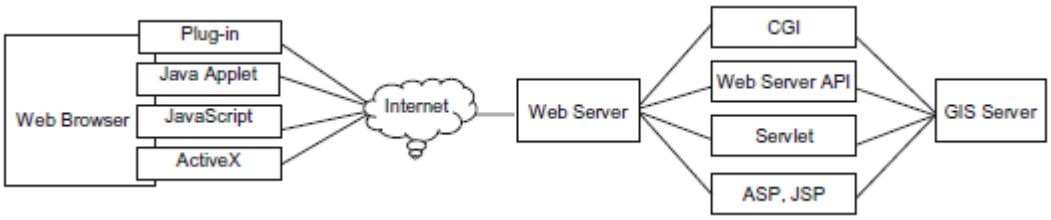


FIGURE 2.2: Architecture client lourd (Agrawal et Gupta, 2017).

— Architecture hybride : correspond à la combinaison des deux types d'architecture décrit ci-dessus.

2.2 Données spatiales 3D : les nuages de points

2.2.1 Définition

Un nuage de points est un ensemble de points exprimé dans un système de coordonnées à 3 dimensions. Les points sont généralement définis par leurs coordonnées en X, Y et Z. Ces points permettent de représenter la forme, la taille, la position et l'orientation d'un ou plusieurs objets dans l'espace. Cette information peut être complétée par une information colorimétrique (RGB), une information thermale, etc. (Grilli et al., 2017).

2.2.2 Techniques d'acquisitions

Un nuage de points est le résultat de la numérisation du monde réel obtenu via diverses méthodes d'acquisition de données :

— La lasergrammétrie fait appel à des scanners qui, par balayage d'un rayon laser, va permettre de relever des points et générer un nuage de points en 3 dimensions. Pour calculer ces points en X Y et Z, il est nécessaire d'enregistrer la direction des rayons (angle vertical et horizontal) et la distance oblique séparant l'appareil de l'objet mesuré. La détermination de la distance peut se faire par : mesure du temps de vol, triangulation, mesure de différences de phases (Maumont, 2010). Il existe différents types de scanner : scanner laser terrestre (TLS), scanner laser mobile (MLS), scanner laser aérien (ALS), LIDAR, etc. (Remondino et El-Hakim, 2006).

— La photogrammétrie digitale fonctionne suivant le principe de la corrélation d'image (Maumont, 2010). Il est possible de produire des modèles en 3 dimensions grâce à plusieurs clichés selon plusieurs points de vue et ainsi en extraire un nuage de points colorisé (Billen et al., 2018).

Le géoréférencement du nuage de points, qui correspond à positionner l'ensemble des points dans un système de coordonnées de référence, peut se faire de deux manières : directement ou indirectement. Dans le premier cas, les points sont levés directement dans le système de coordonnées souhaité. Le scanner positionné sur le terrain est donc directement localisé dans un système de coordonnées de référence grâce à des points de référence comme les points de stations et de visés. Dans le second cas, les points sont relevés dans un système local. Il est nécessaire de réaliser une transformation Helmert 3D linéarisée pour passer du système local

vers le système de coordonnées de référence désiré (Djemâ, 2018).

Les différentes techniques d'acquisition de données peuvent aboutir à la production de nuages de points 3D massifs. Ce qui pose des défis en matière de stockage, prétraitement, présentation et post-traitement des données.

2.3 Affichage de données 3D : moteur de rendu de nuage de points

Les nuages de points possèdent à la fois une grande précision géométrique et un côté « immersif ». Deux qualités leur permettant d'être de bons outils de communication, notamment dans le domaine de la conservation du patrimoine culturel, où ils sont utilisés comme interface pour partager et visualiser les informations collectées (Manferdini et Remondino, 2010). Cependant, les données collectées grâce aux scanneurs peuvent dans certains cas atteindre plusieurs milliards de points, un volume de données qui n'est généralement pas supporté par la mémoire d'un ordinateur. Le développement d'ordinateurs plus puissants et d'algorithmes dit « out-of-core » a été essentiel pour permettre l'affichage, l'exploration, et l'interprétation de nuages de points massifs en temps réel, sans quoi ces données auraient été rendues obsolètes (Poux et al., 2014).

2.3.1 Algorithme "out-of-core"

La plupart des méthodes utilisent des variations de représentation de structure de données spatiales à multi-résolution comme le kd-trees (Goswami et al., 2013), l'octree (Elseberg et al., 2013) ou encore le quadtree (Gao et al., 2014). Ces structures permettent de diviser les données en sous-ensembles pouvant ensuite être sélectionnées de manière dynamique. Par exemple (Voir figure 2.3), sur base du tronc de vue, les nœuds en dehors des régions visibles ne seront pas affichés et les nœuds proches de l'utilisateur seront plus détaillés que ceux éloignés (Martinez-Rubi et al., 2015).

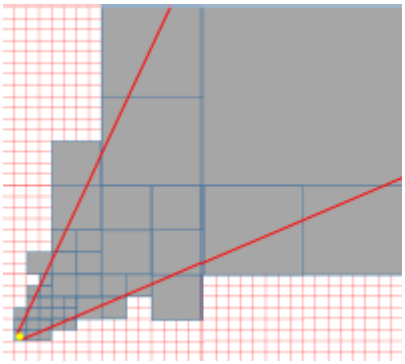


FIGURE 2.3: Simplification 2D du tronc de vue et les niveaux de détails qui sont possibles avec des structures de données à multi-résolution. Les données ont une résolution faible lorsqu'elles sont éloignées et la résolution augmente progressivement à mesure que l'on se rapproche de l'utilisateur (Martinez-Rubi et al., 2015).

.

2.3.2 WebGL

La sortie du standard WebGL (Web Graphics Library) a permis la création de solutions de visualisation 3D de nuage de points à partir d'un simple navigateur web. WebGL est une extension de l'élément canvas de HTML 5 qui est à présent largement utilisé pour développer des applications web permettant une visualisation 3D. WebGL est supporté par une majorité des navigateurs web ainsi que par les smartphones (Android, IOS). WebGL se présente sous forme d'une API graphique 3D, écrit dans un langage de faible niveau, et basé sur OpenGL ES 2.0. L'avantage de WebGL est qu'il définit une connexion directe entre les scripts du navigateur et les capacités graphiques sous-jacentes de l'ordinateur. Par conséquent, les plugins supplémentaires tels que Java ou Adobe Flash ne sont plus nécessaires.

2.3.3 Les logiciels

Certains moteurs de rendu de nuage de points sont des logiciels installés sur le bureau (« desktop ») tandis que d'autres solutions de visualisation sont accessibles via le web.

2.3.3.1 Logiciel basé sur le bureau (desktop)

Scanopy (Scheiblauer et Wimmer, 2011) est un des premiers moteurs de visualisation de nuage de points capable d'afficher un nuage de points composé de milliard de points. Cette approche utilise une structure de données en octree. Un kd-trees est utilisé dans la solution développée par Goswami et al. (2010). Dans le logiciel développé par Richter et al. (2019), également basé sur une structure en kd-trees, des nuages de points classifiés peuvent être

visualisés.

Parmi les solutions commerciales, ESRI ou Bentley incluent des options permettant la visualisation de nuage de points mais uniquement avec des volumes de données restreints. Euclideon ("Euclideon Vault", p. d.) fonctionne avec des volumes de données important et, selon Martinez-Rubi et al. (2015), le logiciel offre une expérience de visualisation de nuage de points excellente.

2.3.3.2 Logiciel basé sur le Web

L'ensemble Plasio, Greyhound et Entwine est une solution de visualisation de nuage de points développé par Butler et al. (2014). Entwine permet de convertir les données en une structure optimisée, un octree, pour un affichage et une diffusion des données efficaces via le serveur http Greyhound. Plasio est une application web basé sur WebGL. Ce moteur de rendu de nuage de points peut gérer jusqu'à 16 millions de points et permet différents rendus : RGB, intensité et élévation. Plasio supporte les fichiers Las et Laz et peut également afficher des nuages de points à partir du serveur Greyhound.

PotreeConverter (Schütz, 2019) et Potree (Schütz, 2016) ont été développés par l'université de TU Wien. Le convertisseur permet de convertir des fichiers de format Laz, Las, PTX et PLY en une indexation à multi-résolution octree requise pour être affiché dans Potree. Ces fichiers peuvent être affichés dans Potree, soit directement à partir du système de fichiers, soit à partir de n'importe quel serveur web (Ex : Apache, Greyhound, etc.). Potree est basé sur WebGL, three.js et Javascript. Three.js est une API, basé sur WebGL, développée pour faciliter l'utilisation de contenu 3D. Potree est capable de gérer des nuages de points massifs. Pour donner un ordre de grandeur de la capacité de Potree, Martinez-Rubi et al. (2015) ont réussi à visualiser le deuxième levé Lidar national des Pays-Bas dont le nuage de points était composé de 640 milliards de points. Ce moteur de rendu possède également des outils de mesure et différents types de rendu sont possibles (classification, RGB, l'intensité, etc.).

GVLiDAR (Deibe et al., 2017) et Vilma (Deibe et al., 2019) sont deux moteurs de rendu de nuage de points basés sur WebGL. Les auteurs se sont essentiellement intéressés aux données Lidar et à leurs manipulations via des outils de mesure géospatiales pouvant être utilisés directement sur le nuage de points.

2.4 Traitement des données 3D : la segmentation

Comme le rappelle Poux et al. (2014), les scanners laser peuvent en quelques secondes « obtenir plusieurs millions de points repérés en 3D avec une densité élevée et une précision de l'ordre de quelques millimètres ». Cependant, cette phase d'acquisition très courte est contrastée par une phase de post-traitement chronophage dû à la production de nuages de points non-structurés. Pour pouvoir exploiter le potentiel qu'offrent les nuages de points, il est nécessaire et essentiel de fournir aux données 3D des attributs qui caractérisent et donnent un sens aux objets représentés. Par exemple, dans le domaine de la conservation du patrimoine culturel, en raison de la complexité des artefacts ou des sites architecturaux et archéologiques, leurs modèles numériques doivent être subdivisés en sous-composantes et organisés selon des régions sémantiques, afin de faciliter l'analyse des données et la production de nouvelles informations (Manferdini et Remondino, 2010).

2.4.1 Définition

La segmentation est le processus de classification d'un nuage de points en différents groupes homogènes. Tous les points appartenant à un même groupe doivent partager une ou plusieurs caractéristiques communes. La classification ou la segmentation sémantique correspondent à la définition de classe et à l'attribution de ces groupes aux classes correspondants (Grilli et al., 2017). Les deux processus étant liés, ils sont parfois confondus et réalisés ensemble (Lesecque, 2019).

2.4.2 Type d'algorithmes de segmentation

Il existe de nombreux algorithmes de segmentation. Poux et al. (2014), Grilli et al. (2017), Nguyen et Le (2013) ont chacun réalisé une classification des différents algorithmes. Les méthodes basées sur la détection des bords, la segmentation basée sur la croissance de régions, et la segmentation par reconnaissance de formes, apparaissent au sein des trois classements.

La segmentation par détection des contours peut être divisée en deux étapes (Rabbani et al., 2007). Tout d'abord, la détection des limites des régions. Suivi d'un regroupement des points à l'intérieur de chaque région pour obtenir le segment final. Les limites sont définies à partir des points qui ont un changement rapide de leurs propriétés géométriques telles que la normale (Bhanu et al., 1986), la courbure (Jiang et al., 1996), et le gradient. Nguyen et Le (2013) soulignent que cette méthode permet une segmentation rapide, mais peut être amenée à produire des résultats imprécis en cas de bruit ou d'une densité insuffisante. Grilli et al.

(2017) rajoutent que la méthode détecte très bien les régions ayant des limites non-connectées, mais identifie mal les segments proches sans une intervention et une interprétation humaine.

La segmentation par reconnaissance de formes part d'une observation simple (Grilli et al., 2017). La majorité des objets réalisés par l'homme peut être décomposée en des primitives géométriques comme des cylindres, des sphères, des plans, etc. Si dans les données du nuage de points, un ensemble de points correspond à la représentation mathématique d'une forme géométrique primitive alors ce groupe est considéré comme un segment. Les algorithmes de RANSAC (Random Sample Consensus) (Fischler et Bolles, 1981) et de HT (Hough Transform) (Ballard, 1981 ; Vosselman et al., 2004) sont les plus fréquemment utilisés. Poux et al. (2014) constatent que cette méthode est « conceptuellement simple, robuste face au bruit et facilement extensible ». Cependant, l'algorithme peut conduire à une sur/sous-segmentation, il possède des difficultés d'isolement des détails fin, et le temps de traitement des données est conséquent.

Une troisième approche, la segmentation par croissance de régions, consiste à faire croître des régions à partir de points de départ (« graines »). Ces derniers comparent des points proches spatialement pour former un nouveau segment, à condition qu'ils présentent des caractéristiques similaires telles que l'orientation de la surface ou la courbure (Jagannathan et Miller, 2007 ; Rabanni et al., 2007). Cette méthode peut être divisée en deux sous approche : « bottom-up » ou « top-down ». Dans son ensemble, la segmentation par croissance de région est plus résistante au bruit que la segmentation par détection des contours.

À ces 3 groupes , Grilli et al. (2017) mentionnent les méthodes hybrides combinant plusieurs méthodes pour tenter de tirer profit au maximum des forces de chacune des méthodes et d'atténuer leurs faiblesses (Rusu, 2010 ; Vieira et Shimada, 2005).

Grilli et al. (2017) citent également les algorithmes de segmentation basés sur le « machine learning » comme la classification hiérarchique (Lu et al., 2016) et le partitionnement en k-moyennes (Comaniciu et Meer, 2002 ; Lavoué et al., 2005 ; Yamauchi et al., 2005). Le « machine learning » cherche à développer des algorithmes basés sur l'intelligence artificielle. Ces derniers permettent aux ordinateurs de prendre des décisions de façon autonome sur base de données empiriques. Ces méthodes sont assez robustes et flexibles, mais elles reposent sur la densité du nuage et nécessitent un temps de calcul très long (Grilli et al., 2017).

2.4.3 Les obstacles à l'automatisation des algorithmes

L'automatisation de la segmentation ne permet pas encore d'obtenir des résultats satisfaisants, dus entre autres :

- à une absence explicite de structure des données des nuages de points (Nguyen et Le, 2013). Poux et al. (2016) tentent d'apporter une solution.
- à une densité d'échantillonnage inégale (Nguyen et Le, 2013).
- à la présence récurrente de bruit et d'occlusion au sein des données qui mènent à des complications pour trouver et ajuster des primitives géométriques aux objets. Ce qui conduit à un phénomène de sur-segmentation (Nguyen et Le, 2013 ; Poux et al., 2014).
- au nombre de paramètres à fournir en entrée et à ajuster pour chaque nuage de points limite l'automatisation (Grilli et al., 2017).
- à la singularité de chaque méthode. Certains algorithmes ne fonctionnent qu'avec des données LiDAR ou avec des nuages de points colorisés par exemple (Grilli et al., 2017).

Selon Nguyen et Le (2013), dû au trop grand nombre de difficultés que posent les données des nuages de point, la mise en place de méthode robuste en temps réel n'est pas encore atteinte. La majorité des étapes nécessitent toujours une intervention humaine pour obtenir des résultats satisfaisants (Grilli et al., 2017). À l'heure actuelle, la segmentation reste toujours manuelle ou légèrement semi-automatisée (Poux et al., 2014).

2.5 Langages de descriptions et d'échange de données géographiques

Dans un SIG, les données géographiques sont stockées en tant que géométrie (point, ligne, polygone et surface) et géoréférencées. Les données spatiales définissent un objet. Ce dernier possède une orientation et une relation avec d'autres objets dans un espace à deux ou trois dimensions, c'est ce qu'on appelle également des données topologiques. Les attributs, l'information sémantique, permettent de décrire ces objets. Ces données attributaires sont stockées dans des bases de données, relationnelles ou NoSQL, et associées à l'information spatiale pour permettre une analyse des données.

Il existe de nombreux formats de description des données géographiques : KML, GML, GeoJSON, Shapefile, WKT, WKB, etc. Dans notre cas, il ne s'agit pas de passer en revue l'ensemble des formats de données géographiques, mais de présenter des formats adaptés et couramment utilisés au sein des SIG web.

2.5.1 Format XML et ses dérivés géographiques : GML, KML et CityGML

XML (Extensible Markup Language) est un format de codification textuelle basé sur une grammaire précise et fournissant des outils standardisés d'édition, de recherche et de présentation. XML est un standard du W3C et il est « échangeable » sur Internet.

Un fichier XML est composé de deux parties :

— Le schéma qui décrit la structure d'une instance d'un document XML (suffixe .xsd).

— Un document XML (suffixe .xml) qui contient les données c'est-à-dire les instances des éléments.

Un élément XML est entouré de balises (« Tags ») d'ouverture et de fermeture, et les éléments peuvent être structurés sous une forme hiérarchique

```
<chapitre>
    <sous-chapitre> Langage de description </sous-chapitre>
</chapitre>
```

Un élément est soit un objet, soit une propriété de l'objet. Les propriétés sont toujours des éléments enfants des objets. Chaque élément doit être encadré par ses balises d'ouverture et de fermeture. Un élément, objet ou propriété, peut-être caractérisé par des attributs.

GML (Geography Markup Language) est un langage à balise de description des objets géographiques construits sur base de XML. C'est un standard développé par l'OGC pour encoder, manipuler et échanger des données géographiques. Il permet d'assurer l'interopérabilité des données dans le domaine de la géomatique. GML permet de décrire un objet géographique, un système de projection, une géométrie, une topologie, le temps, une unité de mesure, et les attributs des objets géographiques.

KML (Keyhole Mark-up Language) est un langage à balise également basé sur XML pour la gestion et l'affichage d'objets géographiques 2D et 3D. Ce format est moins complet et moins riche que GML, mais garantit une plus grande interopérabilité des données. Depuis 2008, KML est un standard de l'OGC. De nombreuses applications utilisent KML : ArcGIS, google Earth, google Maps, Bing Maps, open layers, etc.

CityGML est un format de représentation des objets 3D, notamment urbain, pour le stockage et l'interopérabilité d'accès aux modèles 3D. CityGML est basé sur le standard GML3

de l'OGC et couvre les aspects géométriques, topologiques, d'apparence et sémantiques des modèles d'objets 3D. Les types d'objets 3D comprennent les modèles numériques de terrain (MNT), les bâtiments, la végétation, les plans d'eau, les infrastructures de transport, et les objets urbains. Les différents types d'objets peuvent être décrits d'après cinq niveaux de détail (« Level of Detail (LoD) ») (van Oosterom et al., 2005).

2.5.2 Format JSON, GeoJSON et TopoJSON

JSON (JavaScript Object Notation) (Bray, 2014) est un format léger d'échange de texte, indépendant des langages de programmation et facilement compréhensible par l'homme. Ce format est plus compact, moins redondant et plus léger que le format XML. Le JSON peut rapidement et facilement être parsé à partir du langage JavaScript, pHp et SQL grâce à de simples manipulations, sans avoir besoin d'une API. De plus, il est utilisé comme langage d'échange de données par la technologie AJAX et par les services web.

Le JSON est construit à partir de deux structures :

— Une collection de pairs noms/valeurs : ("name" : "value").

```
{"firstName" : "Jenny", "lastName" : "Hunt"}
```

— Une liste ordonnée de valeurs ([object1, object2]).

```
{"people" : [{"firstName" : "Jenny", "lastName" : "Hunt"},
             {"firstName" : "Jack", "lastName" : "Brown"} ]}
```

Le GeoJSON (Geographic JSON) (Butler et al., 2016) est un format d'encodage de données géospatiales utilisant la norme JSON. Ce format a été développé par un groupe d'utilisateur d'Internet et non par l'OGC. Le GeoJSON supporte les types de géométries suivants : point, ligne, polygone, ensemble de points, ensemble de segments et ensemble de polygones. La position des points est obligatoirement exprimée en coordonnées géodésiques WGS84.

Un objet GeoJSON peut être de huit types différents :

— Les types de géométries cités ci-dessus.

— « Feature » : contient un objet géométrique et une propriété (feature) additionnel.

— « FeatureCollection » : contient une liste de « feature » :

```
{
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
```

```
            "geometry": {
                "type": "Point",
                "coordinates": [
                    494332.1999999881,
                    5419823.798999786,
                    299.1690001487732
                ]
            },
            "properties": {
                "typeMesure": "Point",
                "nom": "Test",
                "description": "Propriété additionnelle"
            }
        }
    ]
}
```


Le GeoJSON permet d'associer aux objets géométriques des attributs d'information non-spatiale (« properties »). Enfin, il peut être décomposé via des requêtes SQL permettant d'accéder facilement à l'information spatiale et sémantique, pour être stocké au sein d'une BDD spatiale.

Le TopoJSON (andrewharvey et mbostock, 2019) est une extension du GeoJSON. Ce format permet l'encodage de la topologie basée sur l'identification des arcs (origine, polygone à droite et à gauche, etc.). TopoJSON permet d'éliminer les redondances, qu'il est possible de rencontrer au sein des géométries associées lorsque le format GeoJSON est utilisé, en les stockant au sein d'un même fichier. Le langage est reconnu par la librairie de PostGIS.

3 Hypothèse

Comme souligné dans l'état de l'art, le développement d'algorithme de segmentations automatiques est loin d'être encore aboutie. La segmentation reste manuelle ou au mieux semi-automatique et correspond toujours à un procédé long et fastidieux. Pour permettre de développer des outils d'analyse spatiale de nuage de points au sein d'une interface web, nous proposons une segmentation manuelle. Celle-ci est ainsi adaptée à tous les cas de figure qu'il est possible de rencontrer au sein de données hétérogènes que sont les nuages de points.

Pour réaliser notre SIG web 3D, nous utiliserons :

- Potree, car c'est le moteur de rendu de nuage de points "open-source" de référence. Il est cité dans de nombreux articles scientifiques et possède les meilleurs résultats en terme de performance d'affichage. De plus, il utilise des standards de technologie web comme WebGL, three.js et javascript. Ce qui permet d'implémenter de nouvelles fonctionnalités aisément. Enfin, Potree inclut déjà une fonction d'exportation de mesure en format GeoJSON.
- PostGIS, l'extension spatiale du système de gestion de base de données "open-source" PostgreSQL, fournissant un ensemble de fonction d'analyse spatiale.
- GeoJSON comme format de description et d'échange de données géographiques. Ce format est supporté par la technologie AJAX permettant de transférer les données entre les différentes strates de l'architecture de l'application. De plus, ce format peut être stocké directement dans PostgreSQL et être décomposé pour accéder à l'information spatiale et sémantique.

L'hypothèse de ce travail est donc de développer un SIG web 3D, permettant de digitaliser des objets vectoriels 3D à partir de nuages de points et d'encoder manuellement de l'information sémantique, en incluant Potree, PostGIS et GeoJSON.

4 Méthodologie

Ce chapitre est structuré en deux parties. Tout d'abord, l'architecture mise en place pour réaliser l'application est détaillée. Le rôle de chaque composante de l'architecture est présenté, ainsi que la manière d'utiliser et/ou d'installer chaque logiciel et librairie utilisés au cours de ce travail. La seconde partie est composée des différentes étapes ayant permis l'implémentation de requêtes entre Potree et PostGIS pour la gestion d'objets vectoriels 3D digitalisés sur base de nuages de points.

4.1 Architecture

La solution mise en place se base sur une approche 3-tiers logique et 2-tiers physique. Comme illustré sur la figure 4.1 ci-dessous, le niveau logique est composé de trois strates : client, application et données. Le niveau physique, quant à lui, est composé de la machine virtuelle d'une part, supportant la strate application et données, et d'autre part de l'ordinateur de l'utilisateur, se chargeant de la strate client.

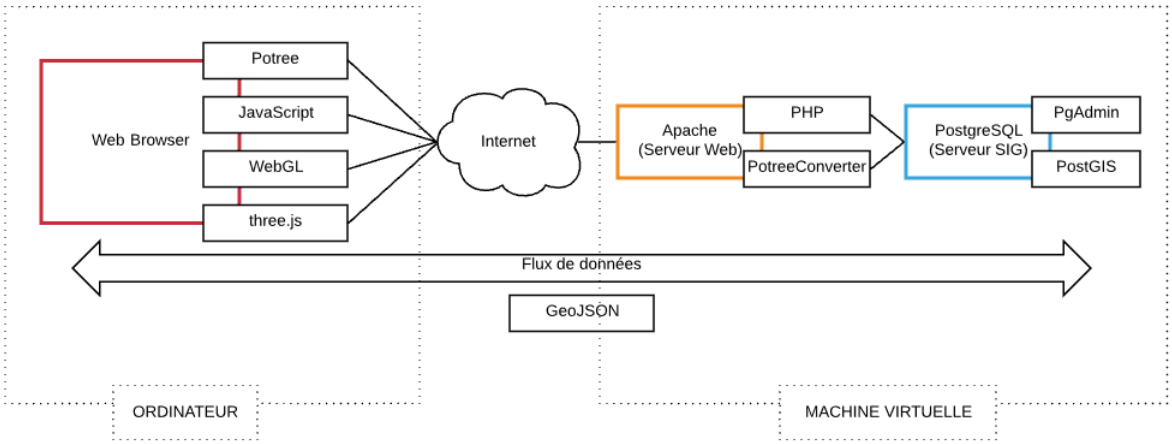


FIGURE 4.1: Schéma de l'architecture 3-tiers logique avec la strate client (rouge), la strate application (orange) et la strate données (bleu).

4.1.1 La strate application

La strate application concerne les tâches exécutées par le serveur. Son rôle est de décrire les requêtes émises par l'utilisateur, d'aller chercher les informations dans la strate données et de renvoyer les résultats des requêtes à la strate client.

4.1.1.1 Apache et PHP

Cette strate va donc exécuter l'ensemble du code de notre application et va comporter un serveur web capable de comprendre le protocole HTTP. Un serveur web permet ainsi de mettre en ligne de l'information statique ou dynamique en lisant le protocole HTTP.

Il existe différents logiciels de serveur web (Ex : Apache, Tomcat, nginx, etc.). Apache est un logiciel libre et son installation est simple (Voir annexe A). Ce serveur web peut être amplifié en y installant des composantes supplémentaires capables de compiler ou d'exécuter des programmes écrits dans différents langages. Dans notre cas, un module interne est installé en parallèle d'Apache : PHP (Voir annexe B). Ce dernier est un langage de script utilisé le plus souvent côté serveur. Il a été conçu pour permettre la création d'application dynamique. PHP est régulièrement couplé à un serveur web. Ce qui permet, par exemple, dans le cadre de ce travail :

— de se connecter à une base de données à l'aide de PDO.

— de récupérer des données issues d'une base de données via des requêtes écrient en SQL (Structured Query Language) et pouvant être intégrées dans un script PHP pour être transmis à un navigateur web.

— de récupérer des données issues d'un navigateur web et de les transmettre à une base de données.

Enfin, il est nécessaire de souligner qu'en développement web, il faut régulièrement tester son site localement avant de le publier. Dans le domaine du DNS («Domain Name System »), l'adresse spéciale appelée localhost fait référence au serveur situé sur l'ordinateur utilisé. Dans le cas d'Apache, le dossier "racine" par défaut est `C:/Apache24/htdocs`. Localhost permet donc d'accéder à l'ensemble des pages web, écrites en HTML ou PHP, situées dans ce dossier à partir d'un navigateur web, même sans connexion à Internet.

4.1.1.2 PotreeConverter

La strate application est complétée par le logiciel PotreeConverter. Ce programme permet de convertir un nuage de points en format Las/Laz, ply, xyz et ptx, en une structure Octree

interprétable par Potree. Les différentes étapes pour réaliser la conversion d'un nuage de points sont :

- Télécharger les fichiers PotreeConverter et LasTool à l'adresse suivante : https://github.com/potree/PotreeConverter. Créer un répertoire sur le disque dur (Ex : C :/PotreeConverter) et y placer l'ensemble des fichiers. Dézipper les fichiers.
- Créer un répertoire pointClouds sur le disque dur (C :/pointClouds) puis deux sous-répertoires :
  - rawData (C :/pointClouds/rawData) : répertoire contenant le.s nuage.s de points que l'on souhaite convertir.
  - convertedData (C :/pointClouds/convertedData) : répertoire de sortie pour accueillir le.s nuage.s de points converti.s.
- Ouvrir l'invite de commande. Utiliser la commande cd (« change directory »), pour changer de répertoire courant dans la console, suivi du chemin du répertoire qui contient le programme PotreeConverter.exe.

  ```
  C:/Users/Ben>cd C:/PotreeConverter/PotreeConverter_1.6_windows_x64
  C:/PotreeConverter/PotreeConverter_1.6_windows_x64>
  ```

  Si on se trouve dans le bon répertoire et qu'on entre la commande PotreeConverter.exe, l'ensemble des commandes qu'offre le programme doit s'afficher dans la console.
- Pour convertir notre nuage de points, entrer la commande :

  ```
  PotreeConverter.exe C:/pointClouds/rawData -o C:/pointClouds/convertedData
  ```

  Ce qui permet de convertir tous les fichiers se trouvant dans le dossier rawData en un format Octree et d'enregistrer les résultats dans le répertoire convertedData. Si la commande a fonctionné, la console affiche le nombre de points converti et le répertoire convertedData doit contenir le résultat. Les étapes pour afficher le nuage de points dans Potree sont détaillées au point 4.1.2.2.2. "Affichage d'un nuage de points dans Potree".

4.1.2 La strate client

La strate client est responsable de la présentation de l'application au client. Elle se compose du navigateur web complété par : JavaScript, Potree, webGL, three.js, et la méthode AJAX.

4.1.2.1 Navigateur Web et JavaScript

La mise en forme d'une page web se fait au moyen de balise HTML (Hyper Text Markup Language) et permet ainsi de décrire la structure d'une page web. Les pages HTML sont dites

statiques, c'est-à-dire que le langage ne permet pas d'interactivité avec le client, ni de réaliser des pages dynamiques. L'amélioration des capacités d'interaction d'une page web se fait par l'incorporation de scripts en JavaScript, via une balise <script> insérée dans la page HTML. Les routines et les fonctions sont généralement inscrites dans l'en-tête de la page (<head>) et/ou peuvent être invoquées dans le contenu de la page (<body>). JavaScript est un langage interprétable par le navigateur web du client et offre la plupart des fonctionnalités de Java, plus quelques objets prédéfinis propres à l'utilisation dans un navigateur permettant une certaine interactivité. Par exemple la réponse à un événement lors du passage de la souris.

Le CSS, ou Cascading Style Sheet, est un autre langage également interprétable par le navigateur. Son principe est de séparer la structure de la page et son contenu (contenu = HTML et mise en page = CSS). Les types d'éléments participant à la structure d'une page (Paragraphe, élément unique ou groupe) sont identifiés par un sélecteur et définis par leurs paramètres d'affichage (Mise en page, taille, couleur, etc.). C'est l'ensemble des règles d'affichage appliquées à la série de sélecteurs qui constitue une feuille de style. Chaque sélecteur dispose d'un degré de priorité, de sorte qu'il est possible, par exemple, de redéfinir le style d'un élément unique au sein d'un paragraphe associé à un style différent (Principe de la cascade des feuilles de style). Une même feuille de style peut être appliquée à plusieurs pages.

4.1.2.2 Potree

Potree, le moteur de rendu de nuage de points, constitue l'interface de notre application. Potree est composé d'une scène, où le nuage de points est affiché, et d'un menu par défaut offrant des outils divers. L'ensemble de l'interface est présenté en détail au sein du point 5.1 "Présentation de l'interface".

4.1.2.2.1 Potree : Installation

Pour installer Potree au sein du serveur Web, les étapes suivantes doivent être réalisées :

— Télécharger Potree à l'adresse suivante : https://github.com/potree/potree. Créer un répertoire de travail, par exemple « mémoire », dans la racine htdocs d'Apache (`C:/Apache24/htdocs/memoire`). Insérer le fichier téléchargé et le dézipper (`C:/Apache24/htdocs/memoire/potree`).

— Ouvrir l'invité de commande. Changer de répertoire courant à l'aide de la commande cd en fournissant le chemin du répertoire potree. Dans notre cas :

```
cd C:/Apache24/htdocs/memoire/potree
```

— S'assurer au préalable que node.js est téléchargé et installé. Sinon se rendre à l'adresse suivante : https://nodejs.org/en/.

— Ouvrir le fichier package.json (C :/Apache24/htdocs/memoire/potree) dans un éditeur de texte. Ce fichier indique les dépendances à installer pour pouvoir utiliser Potree. Pour cela, dans la console, utiliser la commande npm install suivi du nom de la dépendance à installer. Par exemple, pour installer la dépendance gulp-connect :

```
C:/Apache24/htdocs/memoire/potree>npm install gulp-connect
```

— Installer l'outil de construction gulp, en fournissant les trois commandes suivantes :

  — *npm install*

  — *npm install -g gulp*

  — *npm install -g rollup*

Si le serveur Web Apache et les dépendances ont bien été installés, on peut maintenant visualiser dans un navigateur Web différents projets que Potree fournit dans le dossier examples (C :/Apache24/htdocs/memoire/potree/examples). Pour cela, entrer l'adresse suivante : http ://localhost/memoire/potree/examples. La liste complète des différents projets apparaît. Sélectionner ensuite un exemple que vous souhaitez visualiser.

4.1.2.2.2 Affichage d'un nuage de points

Pour afficher un nuage de points de son choix, il est d'abord nécessaire de le convertir, à l'aide de PotreeConverter, en un format Octree pour pouvoir être interprétable par Potree. L'opération de conversion a été réalisée précédemment au point 4.1.1.2 "PotreeConverter". Pour rappel, le résultat final obtenu est un dossier convertedData
(C :/pointClouds/convertedData) contenant le nuage de points converti.

Pour afficher un nuage de points de son choix dans Potree, les étapes suivantes doivent être réalisées :

— Copier/coller le dossier convertedData dans le dossier pointclouds de Potree (`C:/Apache24/htdocs/memoire/potree/pointclouds`).

— Créer un dossier, dans le répertoire de Potree, qui va accueillir l'ensemble des scripts du prototype. Le dossier dans ce cas s'appelle sig3D (`C:/Apache24/htdocs/memoire/potree/sig3D`). Ensuite, copier/coller dans le dossier sig3D une page html contenue dans le dossier example. Cette page va servir de base au développement des différents scripts lors de l'implémentation des requêtes. Ici, la page lion.html a été copiée/collée et a été renommée en sig3D.html.

— Ouvrir dans un éditeur de texte la page sig3D.html. Dans la ligne suivante, modifier l'argument de la méthode loadPointCloud en spécifiant le chemin du répertoire contenant le nuage de points converti donc convertedData.

```
// Affichage du nuage de points de son choix
Potree.loadPointCloud("../pointclouds/convertedData/cloud.js",
"Nom du nuage de points", function(e){
viewer.scene.addPointCloud(e.pointcloud);
});
```

— Pour vérifier le résultat, entrer dans un navigateur Web : `http://localhost/memoire/potree/sig3D/sig3D.html`. Le nuage de points converti doit s'afficher dans Potree.

4.1.2.3 three.js

three.js est une librairie javascript pour la gestion d'objet vectoriel 3D côté client. Nous exploitons cette librairie pour la création et la surimpression d'objet 3D dans Potree sur fond de nuage de points afin de pouvoir leur associer de l'information sémantique et de les stocker dans une base de données spatiales.

4.1.2.3.1 Les fonctionnalités de three.js

three.js est régulièrement confondu avec WebGL. En effet, le premier utilise le second pour dessiner en 3D. WebGL est un système de très bas niveau qui ne dessine que des points, des lignes et des triangles. Pour réaliser du contenu 3D avec WebGL cela nécessite plusieurs lignes de code. three.js facilite ceci en gérant de nombreux éléments comme les scènes, la lumière, les ombres, le matériel, la texture, etc. Des éléments que l'on devrait coder si on utilisait que WebGL.

Les fonctionnalités de three.js sont définies dans un seul fichier JavaScript nommé three.js. L'API est utilisée par Potree et on peut retrouver le fichier three.js dans le dossier libs (C :/Apache24/htdocs/memoire/potree/libs). Il existe une version compressée, three.min.js, qui contient les mêmes définitions, mais elle est écrite dans un format qui n'est pas conçu pour être lisible par l'homme. C'est cette version qui est utilisée par Potree. Pour faire appel à une librairie, il est nécessaire de l'inclure via l'élément <script>. Le fichier sig3D.html étant issu d'un des exemples que fourni Potree, la ligne est déjà écrite :

```
<script src="../libs/jquery/jquery-3.1.1.min.js"></script>
```

Toutes les classes de three.js utilisent les propriétés d'un objet appelé THREE et commencent

donc toutes par THREE. Les trois classes de base dans three.js sont :

— THREE.scene : correspond à l'espace 3D qui contiendra les lumières, les géométries, les caméras, etc.

— THREE.camera : type particulier d'objet qui représente un point de vue à partir duquel une image du monde 3D peut être visualisée. Il existe deux types de caméra, une utilisant une projection orthographique et l'autre utilisant une projection perspective. Elles sont représentées par les classes THREE.OthographicCamera et THREE.PerspectiveCamera qui sont des sous-classes de THREE.camera. Plusieurs paramètres permettent de définir la caméra. Par exemple, la caméra perspective permet d'obtenir une vue 3D de la scène où les éléments loin semblent plus petits que les éléments proches. Pour cela, la caméra perspective permet de définir un tronc (« frustrum »). Un tronc est la partie d'un solide situé entre deux plans parallèles. Le solide est généralement un cône ou une pyramide. Le tronc est défini à partir de quatre paramètres :

```
new THREE.PerspectiveCamera(fOv, aspect, near, far);
```

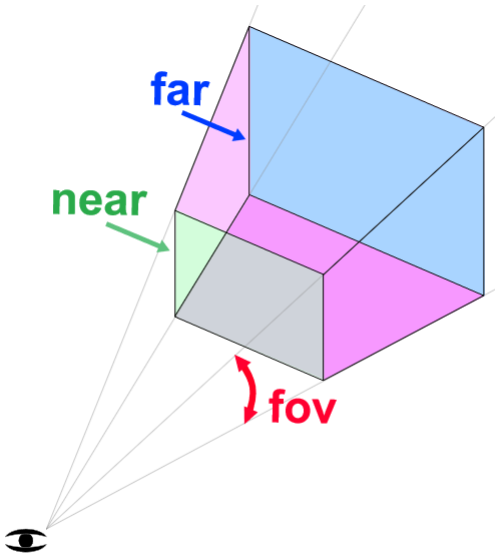


FIGURE 4.2: Schéma du tronc de vue.

Au sein de la ligne de code et sur la figure 4.2 :

— Near : défini où l'avant du tronc commence.

— Far : défini où le tronc se termine.

— Fov ("field of view" = champ de vision) défini la hauteur de l'avant et de l'arrière du tronc.

— Aspect : définit la largeur de l'avant et de l'arrière du tronc.

— THREE.Objects : La classe object permet d'associer une primitive géométrique et le matériel. Un object peut être une ligne, un point, un mesh, etc.

La caméra et la scène sont déjà déclarées et définies au sein de Potree. Pour ajouter des géométries 3D, il suffit de les ajouter à la scène grâce à la fonction :

```
scene.add(object)
```

4.1.2.3.2 Primitive géométrique

Pour la construction d'objet vectoriel 3D, three.js propose un grand nombre de primitives géométriques. Les primitives sont des formes 3D composées d'un ensemble de paramètres qui sont générées au moment de l'exécution du script. Par exemple :

— Si on souhaite générer un cube, on fait appel à la primitive BoxBufferGeometry ou BoxGeometry. On peut également définir des paramètres comme la largeur, la hauteur, la profondeur du cube, etc.

```
new THREE.BoxBufferGeometrie(width, height, depth);
new THREE.BoxGeometry( width, height, depth);
```

— Pour un cercle : CircleBufferGeometry ou CircleGeometry

```
new THREE.CircleBufferGeometry(radius, segment);
new THREE.CircleGeometry(radius,segment);
```

— La liste complète des primitives se trouvent à l'adresse suivante : `https://threejs.org/docs/#api/en/geometries/BoxGeometry`. Dans la fenêtre de gauche, descendre jusqu'à l'onglet Geometries pour visualiser la liste.

Pour chaque primitive, comme on peut le constater, il existe deux types différents : Geometry ou BufferGeometry. La différence entre les deux types réside dans la performance et la souplesse qu'offre chacune des méthodes.

Les primitives basées sur BufferGeometry sont des types orientés sur la performance. Les sommets (« the vertices ») de la géométrie sont générés directement dans un format de tableau efficace pour être téléchargés par le GPU pour l'affichage de la primitive. Cela implique une utilisation de la mémoire moindre comparée au type Geometry. Leur principal désavantage est le manque de flexibilité qu'offre la méthode. En effet, si on souhaite personnaliser une forme, en terme de programmation, la tâche peut se révéler complexe.

Les primitives basées sur le type Geometry sont dites plus flexibles. Elles sont construites à partir de la classe Vector3 pour les points 3D ou encore Face3 pour les triangles. Cependant,

ces classes prennent plus de mémoire que le type BufferGeometry car three.js doit les convertir dans un format similaire à BufferGeometry avant de pouvoir les afficher.

L'objectif étant de générer des formes géométriques personnalisées, on utilisera le type Geometry pour sa flexibilité.

Par exemple, si on souhaite générer un cube via le type Geometry :

— Déclarer la classe Geometry

```
var geometry = new THREE.Geometry();
```

— Déclarer la position des 8 sommets en fournissant en paramètre de la classe Vector3 les coordonnées en X, Y et Z de chaque sommet :

```
geometry.vertices.push(
    new THREE.Vector3(-1, -1, 1), //0
    new THREE.Vector3(1, -1, 1),  //1
    new THREE.Vector3(-1, 1, 1),  //2
    new THREE.Vector3(1, 1, 1),   //3
    new THREE.Vector3(-1, -1, -1),//4
    new THREE.Vector3(1, -1, 1),  //5
    new THREE.Vector3(-1, 1, -1), //6
    new THREE.Vector3(-1, 1, -1), //7
);
```

— Pour construire la face du cube, on utilise des triangles, deux pour chaque face du cube, à partir de la classe Face3 en spécifiant en paramètre l'indice des trois sommets composant la face. Attention, l'ordre dans lequel les indices des sommets sont déclarés est important. Pour que les faces soient visibles et dirigées vers l'extérieur du cube, les indices doivent être spécifiés dans le sens inverse des aiguilles d'une montre :

```
geometry.faces.push(
 //avant
 new THREE.Face3(0, 3, 2),
 new THREE.Face3(0, 1, 3),
 //droite
 new THREE.Face3(1, 7, 3),
 new THREE.Face3(1, 5, 7),
 //arrière
 new THREE.Face3(5, 6, 7),
 new THREE.Face3(5, 4, 6),
```

```
    //gauche
    new THREE.Face3(4, 2, 6),
    new THREE.Face3(4, 0, 2),
    //dessus
    new THREE.Face3(2, 7, 6),
    new THREE.Face3(2, 3, 7),
    //bas
    new THREE.Face3(4, 1, 0),
    new THREE.Face3(4, 5, 1),
);
```

— Il est nécessaire de définir un type de matériel. Le matériel permet de définir l'apparence des objets dans la scène. three.js fournit plusieurs types de matériaux :

```
var material = new THREE.MaterialName({options});
```

— Déclaration de la classe Mesh, pour associer la géométrie et le matériel, et ainsi créer le cube :

```
var cube = new THREE.Mesh(geometry, material);
```

— Enfin, le cube est ajouté à la scène pour être affiché à l'écran :

```
scene.add(cube);
```

4.1.2.4 Méthode AJAX

L'API jQuery, écrite en JavaScript, permet de faciliter l'utilisation de la méthode AJAX (asynchronous JavaSCript and XML). L'idée même d'AJAX est de faire communiquer une page Web avec un serveur Web sans occasionner le rechargement de la page. Le dialogue entre le navigateur et le serveur se déroule la plupart du temps de la manière suivante : un programme écrit en JavaScript, incorporé dans une page web, est exécuté par le navigateur. Celui-ci envoie en arrière-plan des demandes au serveur Web, puis modifie le contenu de la page actuellement affichée par le navigateur Web en fonction du résultat reçut du serveur, évitant ainsi la transmission et l'affichage d'une nouvelle page complète. Par exemple, Potree fait appel à des requêtes AJAX pour l'affichage d'un nuage de points. À chaque fois qu'un utilisateur zoom ou change d'angle de vue, le nombre de pixels affiché sur la scène varie, mais la page complète n'est pas rechargée.

Dans notre cas, la méthode AJAX est utilisée pour transférer des données géographiques en format GeoJSON entre la strate client et la strate données.

Comme Potree utilise déjà l'API jQuery, par défaut, celle-ci est déjà incluse dans le fichier sig3D.html.

```
<script src="../libs/jquery/jquery-3.1.1.min.js"></script>
```

La méthode Ajax s'écrie de la manière suivante :

```
$.ajax({options});
```

L'ensemble des options possibles se trouvent à l'adresse suivante : `https://api.jquery.com/jQuery.ajax/`. La méthode AJAX ci-dessous illustre les options régulièrement utilisées au sein de ce travail.

```
$.ajax({
    url: 'exemple.php', // adresse à laquelle la requête est envoyée
    data: {"nameData": data}, // Les données à envoyer au serveur
    dataType : 'type', // le type de données transmis au serveur
                       // Peut être de type : XML, html, JSON, script
    type: "POST", // Le type de requête :
            // POST = envoyer des données
            // GET = réceptionner des données
    //La fonction à appeler si la requête a abouti
    success: function (data, status, xhr) {
    // Permet de vérifier si la requête a été réalisée
            alert(geojson + " " + status + " " + xhr);
    },
    //La fonction a appeler si la requête n'a pas abouti
    error: function (req, status, error) {
    // Permet de savoir le type d'erreur
                    alert(req + " " + status + " " + error);
    }
});
```

4.1.3 La strate données

La dernière strate de l'architecture est dédiée au stockage des données. Elle est composée du serveur de données, correspondant à un ordinateur, supportant le système de gestion de base de données (SGDB) relationnelle PostgreSQL. PostGIS, l'extension spatiale de PostgreSQL, complète la strate.

Les étapes permettant l'installation de PostgreSQL et de PostGIS ainsi que la connexion du SGBD au serveur web Apache se trouvent en annexe C. Il a également été nécessaire de créer une base de données spatiales (Voir annexe C) pour stocker, d'une part, les objets vectoriels 3D qui vont être digitalisés, et d'autre part pour associer et stocker l'information sémantique encodée par l'utilisateur.

PostgreSQL est un logiciel libre de SGBD objet-relationnel et pgAdmin est sa plateforme d'administration. PostgreSQL supporte une grande partie du standard SQL tout en offrant de nombreuses fonctionnalités modernes : requêtes complexes, clés étrangères, triggers, vues modifiables, intégrité transactionnelle, contrôle des versions concurrentes.

PostGIS est développé comme un jeu de fonctions et de types de données permettant de spatialiser les tables du SGBD relationnel PostgreSQL. Des entités spatiales peuvent ainsi être stockées dans des tables de données et on peut y appliquer des traitements spatiaux. Les fonctions supportées par PostGIS couvrent les fonctionnalités proposées par l'OGC :

— De gestion : création et suppression de colonnes spatiales, assignation d'un SRID par géométrie

— Relationnelles : basées sur la proximité (Ex : distance) et les relations(Ex : prédicats) d'algèbre topologique (Ex : disjoint, intersect, etc.).

— De traitement géométrique : centroïde, longueur, surface, polygone convexe, espace tampon, etc.

— D'accès aux propriétés géométriques (SRID, enveloppe, etc.)

— De construction des géométries au départ de fichiers WKT, WKB ou GeoJSON

4.2 Implémentation des requêtes

L'installation de l'architecture de l'application est à présent terminée. Ce chapitre s'attache à présenter la méthodologie pour la mise en oeuvre d'outils de segmentation manuelle et d'analyse spatiale de données 3D. Cette méthodologie se veut facilement reproductible. Chaque étape ayant permis l'implémentation des requêtes est détaillée. L'ensemble des scripts sont écrits dans un seul dossier de travail : sig3d. Aucun script source de Potree n'a été modifié. Ainsi, si un acteur intéressé souhaite utiliser les scripts développés, il lui suffit de télécharger le dossier de travail et de l'intégrer dans l'emplacement adéquat au sein de l'architecture de l'application.

Trois requêtes ont été implémentées au sein de l'application :

- La 1er requête correspond à l'implémentation de l'outil de segmentation manuelle. Il permet à un utilisateur de digitaliser sur le nuage de points une géométrie (Point, ligne ou polygone) et d'associer de l'information sémantique à celle-ci. L'information spatiale et sémantique produite par l'utilisateur sont ensuite stockées dans la base de données spatiales.
- La 2ème requête se charge de créer et d'afficher sur la scène de Potree un objet vectoriel 3D issue de la base de données spatiales.
- La 3ème requête est l'implémentation d'une fonction spatiale qu'offre PostGIS. Elle permet de déterminer l'objet 3D le plus proche spatialement d'un autre objet 3D. Son but est de démontrer qu'il est possible de réaliser un traitement spatial à partir des objets vectoriels 3D produits lors de la 1ère requête.

4.2.1 Requête n°1 : Segmentation manuelle

La requête n°1, du point de vue de l'utilisateur doit lui permettre :

- De digitaliser des formes géométriques (point, ligne ou polygone) à partir d'un nuage de points affiché dans Potree.
- De classifier le segment digitalisé en encodant manuellement de l'information sémantique.
- D'enregistrer chaque segment classifié au sein d'une BDD spatiale.

Pour répondre à ces trois objectifs :

- La segmentation manuelle va être réalisée en utilisant les outils de mesure disponibles dans le menu par défaut de Potree. Ceux-ci vont permettre à l'utilisateur de digitaliser

différents types de géométrie : l'outil de mesure de point permet de digitaliser un point ; l'outil de mesure de distance et d'angle permet de digitaliser une ligne ou une polyligne ; l'outil de mesure d'aire permet d'obtenir un polygone. L'outil de mesure de volume ne peut pas être utilisé. En effet, ce type de géométrie n'est pas encore supporté par le format GeoJSON. Quant à l'outil fournissant un profil en long, il n'a pas d'intérêt car il fournit uniquement les coordonnées des points correspondant au début et à la fin du profil au long.

— Pour l'encodage manuel de l'information sémantique, deux champs de saisie de texte appelé respectivement nom et description vont être créés et permettront à l'utilisateur de fournir des attributs sémantiques au segment.

— Un bouton digitalisation va permettre de valider la digitalisation ainsi que les attributs saisis par l'utilisateur. Enfin, cette information sera enregistrée et stockée dans la BDD spatiale.

La figure 4.3 ci-dessous illustre le déroulement de l'implémentation de la requête n°1. Trois étapes ont été nécessaires. Chaque étape est détaillée lors des trois points suivants.

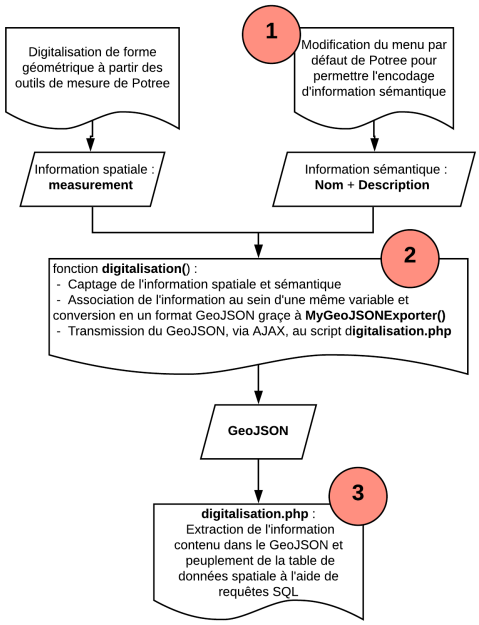


FIGURE 4.3: Déroulement de l'implémentation de la première requête.

4.2.1.1 Etape 1 : Modification du menu par défaut de Potree

La première étape de la requête consiste à modifier le menu par défaut de Potree pour permettre à l'utilisateur de saisir un nom et une description à la forme géométrique digitalisée.

Pour personnaliser le menu de Potree :

— Le script *customsidebarsection.html* (C :/Apache24/htdocs/memoire/potree/examples) fournit un exemple sur comment ajouter une nouvelle section au menu de Potree et la manière de la peupler. Si on ouvre le script dans un éditeur de texte. On s'intéresse au bloc suivant :

```
// Charge dans Potree l'interface graphique c-à-d le menu de Potree
viewer.loadGUI(() => { //GUI : Graphic User Interface
  viewer.toggleSidebar();
  //Nom de la section
  let section = $(`
  <h3 id="menu_meta" class="accordion-header ui-widget">
  <span>Metadata</span></h3>
  <div class="accordion-content ui-widget pv-menu-list"></div>
  `);
  // Contenu de la section
  let content = section.last();
  content.html(`
    <div class="pv-menu-list">
        A custom Section in the sidebar!<br>
                <br>
                Uncomment "content.hide();" to hide content by default.<br>
                <br>
                Take a look at src/viewer/sidebar.html and sidebar.js to
                learn how the other sections were populated.
        </div>
        `);
  //Si click sur la section, celle-ci se déroule
  section.first().click(() => content.slideToggle());
  // Défini l'endroit où la nouvelle section est insérée
  section.insertBefore($('#menu_about'));
});
```

— Le bloque décrit ci-dessus a été utilisé comme modèle et copié/collé à partir de *viewer.loadGUI()* dans le script sig3D.html (`C:/Apache24/htdocs/memoire/potree/sig3D`).

— Dans la définition de la variable *section*, la balise contient le nom donné à la section. Le nom a été changé en *mémoire*.

— Dans la variable *content*, l'ensemble du code HTML a été remplacé par :

  — Un premier (qui signifie division du document) permet de créer la sous-section *digitalisation* dans la section *mémoire* :

```
<div class="divider"><span>Digitalisation</span></div>
```

— Deux éléments <input> (*inputNom* et *inputDescription*) chacun avec un attribut type = 'text' permettent de créer des champs de saisie de texte sur une seule ligne pour permettre à l'utilisateur de saisir le nom et la description qu'il souhaite associer à la géométrie digitalisée. Il est important de fournir aux balises un identifiant unique pour pouvoir accéder au contenu des balises par la suite.

```
<span>Nom : </span> <input type="text" id="inputNom">
<span>Description : </span> <input type="text" id="inputDescription">
```

— Un élément <button> permet de créer un bouton. Il pourra être utilisé par l'utilisateur pour valider la mesure réalisée et l'information sémantique saisie. Au sein de l'élément <button>, on ajoute une propriété *onclick* permettant d'enclencher la fonction *digitalisation()* (décrite au point suivant) lors du clic sur le bouton par l'utilisateur :

```
<button id = "digitalisation" onclick="digitalisation()">Digitalisation</button>
```

4.2.1.2 Etape 2 : Association et conversion de l'information en format GeoJSON

Lorsque l'utilisateur clique sur le bouton digitalisation, après avoir digitalisé et classifié un segment, la fonction *digitalisation()* est enclenchée. Cette dernière, écrite dans la page web sig3D.html (Voir annexe D), permet :

— De capter le texte entré par l'utilisateur dans les deux champs de saisie de texte donc *inputNom* et *inputDescription*.

— De capter l'information géographique de la forme digitalisée.

— D'associer ces deux types d'information dans une variable et de la convertir en format GeoJSON.

— De transmettre la variable, via AJAX, à un script PHP : digitalisation.php.

Pour comprendre la manière dont l'information spatiale et non-spatiale est captées par la fonction textitdigitalisation(), il est nécessaire d'introduire le DOM (Document Object Model). Ce dernier est une API qui représente et interagit avec tout type de document HTML ou XML. Le DOM est un modèle de document chargé dans un navigateur web. La représentation du document HTML est un arbre nodal. Chaque nœud représente une partie du document. Ainsi, le DOM est crucial en développement Web, car il autorise du code exécuté dans un navigateur à accéder et interagir avec chaque nœud dans un document.

Pour inspecter la structure de la page web sig3D.html :

- Ouvrir la page Web dans un navigateur Web : `http://localhost/memoire/potree/sig3D/sig3D.html`.
- Accéder à la console web du navigateur Web. Cette console permet d'afficher plusieurs informations associées à une page web, comme par exemple : les requêtes réseaux (Ex : AJAX), les erreurs d'exécutions de code dans la page web, un inspecteur permettant de visualiser la structure HTML, le DOM de la page Web, etc.
- Cliquer sur le DOM, l'arbre nodal de la page Web s'affiche. Il est également possible d'appliquer un filtre pour trouver plus facilement des éléments. Par exemple, la partie de droite de l'interface de Potree correspond au nœud viewer au sein de l'arbre nodal.

À présent, on peut présenter le code contenu au sein de la fonction *digitalisation()* :

- Pour capter le texte entré par l'utilisateur dans les deux champs de saisie de texte, on utilise la méthode *getElementById().value* en fournissant en paramètre l'ID des deux éléments qui nous intéressent donc *inputNom* et *inputDescription*. L'objet *document* qui fait appel à cette méthode constitue le nœud d'entrée de l'ensemble du contenu HTML de la page web. Les deux variables *nom* et *description* sont chargées de stocker l'information collectée.
- L'information spatiale est contenue dans le nœud *viewer.scene.measurements* qu'on enregistre dans un tableau *measurements*. Ensuite, on ajoute à ce tableau les variables *nom* et *description* : *measurements.nom* et *measurements.description*.
- Dans le menu de Potree, il est possible d'exporter une mesure en format GeoJSON. La fonction qui réalise cette tâche est contenue dans le script GeoJSONExporter.js. Cette fonction prend en paramètre une variable *measurements[]* et retourne la variable en format GeoJSON, en reprenant uniquement l'information spatiale. On va donc modifier cette fonction, pour simplement lui demander d'inclure l'information sémantique (nom et description) dans le GeoJSON de sortie, pour cela :
  - Copier/coller le script source dans le dossier sig3D et renommer le script en MyGeoJSONExporter.js.
  - Inclure la dépendance dans sig3D.html au moyen de balise <script> suivi du chemin du fichier.
  - Ouvrir MyGeoJSONExporter.js(Voir annexe E pour consulter le code complet) et ajouter les lignes *measurement.nom* et *measurement.description* dans chacun des blocs *properties* de chaque type de géométrie. Un exemple est illustré dans le code ci-dessous pour la géométrie point.

```
let feature = {
  type: 'Feature',
  geometry: {
    type: 'Point', //type de géométrie mesurer
        coordinates: coords[0] //coordonnées de la mesure effectué
  },
  properties: {
        /* ajout des attributs sémantiques,
        rentrer par l'utilisateur, a l'objet GeoJSON*/
        typeMesure: measurement.name,
        nom: measurement.nom,
       description: measurement.description
        }
};
```

— Enfin, une fois la conversion réalisée, le fichier GeoJSON est transmis à un script PHP digitalisation.php via une méthode AJAX.

4.2.1.3 Etape 3 : Importation du GeoJSON dans la BDD spatiale

Le script digitalisation.php, le code complet se trouve en annexe F, permet de se connecter à la base de données (via PDO) et d'effectuer des requêtes SQL pour insérer le fichier GeoJSON dans la base de données. Cependant, il est nécessaire de créer au préalable une table de données spatiale permettant de stocker et structurer les données.

4.2.1.3.1 Table de données spatiale

Dans une base de données relationnelles :

— Une classe est matérialisé par une table. Un schéma de classe, rédigé en un langage de formalisation (Ex : UML), permet de décrire une classe.

— Une table est organisée sous forme de tableau permettant de stocker un ensemble de données.

— Le nombre de colonnes, au sein de la table, correspond au nombre d'attributs caractérisant la classe.

— Le nombre de lignes correspond au nombre d'entités appartenant à cette classe.

Une table spatiale possède une colonne avec un type spécial : geometry. C'est un type de données qui permet de stocker de l'information spatiale relative à une entité (point, ligne

ou polygone) dans un système de coordonnées. Dans notre cas, la colonne contenant le type geometry se nomme *geom* et la table que l'on va créer s'appellera *geojson*

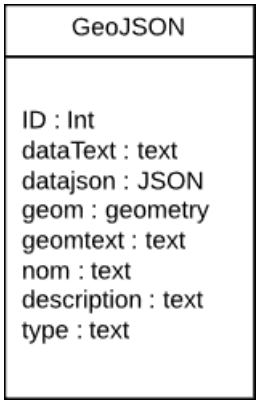


FIGURE 4.4: Schématisation en format UML de la table GeoJSON.

Dans la figure 4.4, les différents attributs de la classe correspondent à :

- ID : identifiant unique.
- datatText : le fichier GeoJSON de type text.
- datajson : le fichier GeoJSON converti en JSON. Il est nécessaire de le convertir en un type JSON pour permettre d'extraire via des requêtes SQL les différentes composantes du GeoJSON (les coordonnées, le type de géométrie, les attributs non-spatiales).
- geom : représentation géométrique.
- geomText : conversion de la colonne *geom* en type text pour être lisible.
- nom : le nom attribué par l'utilisateur.
- description : la description attribuée par l'utilisateur.
- type : le type de géométrie (point, ligne, polygone, etc.).

Pour créer et spécifier la structure de la table dans PostgreSQL, on réalise les étapes suivantes :

- Écrire en langage SQL (Structured Query Language) la structure de la table dans un simple éditeur de texte :

```
CREATE TABLE geojson
( id SERIAL PRIMARY KEY,
  datatext text,
  datajson json,
```

```
 geom geometry,
 geomtext text,
 nom text,
 description text,
 type text
);
```

Dans le cas du type id, on utilise le type serial. Ce dernier n'est pas réellement un type de données, mais seulement une notation permettant à PotsgreSQL de créer automatiquement un identifiant unique correspondant à un nombre entier.

— Exécuter le code dans pgAmin. Pour cela, on réalise un clic droit sur la base de données dans lequel on souhaite insérer la table. On clique sur l'option « Query tool ». Enfin, on insère le script et on l'exécute. La table spatiale est à présent créée.

4.2.1.3.2 Requêtes SQL

La structure de la table étant réalisée, il faut à présent la peupler. Le script digitalisation.php va permettre de récupérer le fichier GeoJSON envoyé par la fonction *digitalisation()* via une méthode AJAX. Ensuite, à l'aide de requêtes SQL, les différentes composantes du GeoJSON vont être extraites et insérées dans les colonnes adéquates de la table.

Le script se déroule de la manière suivante :

— La méthode POST permet de capter toutes les données envoyées par la méthode AJAX de type Post :

```
$geojson = $_POST['geojson'];
```

— Ensuite, l'extension PHP Data Object (PDO) permet d'établir une connexion avec PostgreSQL. En paramètre du constructeur, on spécifie la source de la base de données, le nom de l'utilisateur et le mot de passe :

```
    // paramètre de connexion à la BDD
$bdd = new PDO("pgsql:host=localhost;port=5432;dbname=potree","postgres","Mot de passe");
// Execution de la connexion
$bdd->exec("SET CHARACTER SET utf8");
```

— Lorsque la connexion est établie, plusieurs requêtes SQL sont effectuées :

  — La première permet d'insérer le GeoJSON dans la colonne dataText :

  ```
  INSERT INTO geojson (dataText) VALUES(?)
  ```

  — La seconde convertie le GeoJSON de type text en type JSON pour pouvoir accéder aux éléments dans les requêtes suivantes :

```
UPDATE geojson SET datajson = datatext::json
```

— Ensuite, à partir du GeoJSON et en utilisant la fonction *ST-GeomFromGeoJSON(json)*, la géométrie PostGIS est créée et insérée dans la colonne geom :

```
UPDATE geojson SET geom =
ST_GeomFromGeoJSON((datajson -> \'features\' -> 0 -> \'geometry\')::text))
```

— Les deux requêtes suivantes permettent d'extraire l'information non-spatiale du GeoJSON, le *nom* et la *description*, et de l'insérer dans les colonnes adéquates :

```
//// Insertion du nom
UPDATE geojson SET nom = (datajson
-> \'features\' -> 0 -> \'properties\' ->> \'nom\')
/////Insertion de la description
UPDATE geojson SET description = (datajson
-> \'features\' -> 0 -> \'properties\' ->> \'description\')
```

— Enfin, le type de géométrie (point, ligne, polygone) est également extrait du GeoJSON et introduit en text dans la colonne « type » :

```
UPDATE geojson SET type =
(datajson -> \'features\' -> 0 -> \'geometry\' ->> \'type\')
```

L'objectif de la première requête est accompli. Le résultat est visible au sein de la base de données. Cette dernière est composée d'une seule table *geojson* peuplée de l'information spatiale issue de la digitalisation, et de l'information spatiale encodée par l'utilisateur.

4.2.2 Requête n°2 : objet 3D

Toute l'information géographique et sémantique liée à chaque segment digitalisé à partir du nuage de points est contenue dans la base de données. L'objectif principal de la seconde requête est d'afficher des objets vectoriels 3D par-dessus le nuage de points à l'aide de l'API three.js.

L'implémentation de la seconde requête, dont le déroulement est illustré par la figure 4.5, est à nouveau composée de trois étapes :

— Dans la première étape, il s'agit de créer deux menus déroulants qui permettront à l'utilisateur de choisir le segment qu'il souhaite afficher sur la scène de Potree.

— Au sein de la deuxième étape, l'implémentation de la fonction *object3D()* est détaillée.

— La troisième étape explique le fonctionnement de la fonction *drawThreeGeo(json)* chargé de dessiner un objet vectoriel sur la scène de Potree à partir d'un GeoJSON.

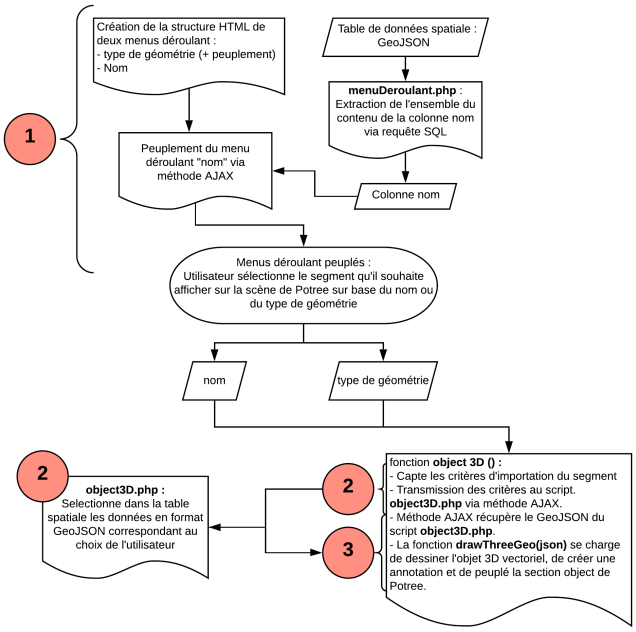


FIGURE 4.5: Déroulement de l'implémentation de la seconde requête.

L'ensemble des résultats attendus concernant la seconde requête :

— L'utilisateur puisse importer et afficher dans Potree des segments contenus dans la base de données sur base du nom et/ou du type de géométrie du segment.

— Création et affichage des objets vectoriels 3D sur la scène de Potree.

— Une annotation doit être associée à chaque objet 3D. Le titre de l'annotation correspond au nom de l'objet 3D et le contenu à la description entrée par l'utilisateur.

— Dans le menu de Potree la section scène -> objects doit être peuplée de l'objet 3D importé sur la scène et de l'annotation. Ce qui permettra à l'utilisateur d'avoir un contrôle sur ces deux éléments en les affichant ou non.

4.2.2.1 Etape 1 : Création et peuplement des menus déroulants

Pour que l'utilisateur puisse savoir quel segment importer, il faut qu'il puisse visualiser une liste contenant l'ensemble des segments stocké dans la BDD. Pour répondre à ce besoin, deux menus déroulants, respectivement *nom* et *type de géométrie*, ont été créés. Pour cela :

— La structure de la nouvelle sous-section « Importation » est écrite en HTML en dessous de la précédente sous-section « Digitalisation » :

```
<div class="divider"><span>Importation</span></div>
```

— De manière général, la structure d'un menu déroulant est composée des éléments suivants :

  — L'élément permet de fournir un nom au menu déroulant

  — L'élément <select> permet de créer un menu déroulant

  — L'élément <option> permet de définir les options d'un menu déroulant

— Le premier menu déroulant s'appelle «Type de géométrie ». Concernant les options, il n'y a que trois types de géométries différentes : point, ligne ou polygone. Il est nécessaire d'ajouter un choix supplémentaire dans le cas où l'utilisateur ne spécifierait pas le type de géométrie qu'il souhaite importer, mais uniquement le nom du segment.

```
<!-- Menu deroulant : Type de geométrie -->
<span>Type de géométrie :</span> </br>
<select id="typeGeom">
        <option value='ALLTYPE'>Geometrie</option>
        <option value='Point'>Point</option>
        <option value='LineString'>Ligne</option>
        <option value='Polygon'>Polygone</option>
</select> </br>
```

Le premier menu déroulant est à présent finalisé.

— Le second menu déroulant va contenir l'ensemble des noms des objets digitalisés par l'utilisateur. Dans ce cas, il n'est pas possible de définir les options au préalable.

```
<!-- Menu deroulant : Nom -->
<span>Nom :</span></br>
<select id="idNom"/></br>
```

— Pour peupler le menu déroulant reprenant le nom des segments enregistrés dans la base de données, une méthode AJAX de type get est utilisée. Celle-ci fait appel au script menuDeroulant.php (Voir annexe J).

```
$.ajax({
        url: 'menuDeroulant.php', // adresse à laquelle la requête est envoyée
        data: 'jsonData', // Les données reçues du serveur
        dataType: 'json', // le type de données reçut
        type: "get"
```

— Au sein du script PHP, une requête SQL permet de récupérer l'ensemble de la colonne nom de la table GeoJSON.

```
SELECT nom FROM geojson ORDER BY nom
```

— En cas de succès de la méthode AJAX, une fonction est enclenchée. Elle s'attache à peupler le menu déroulant. Pour réaliser cela, un nouvel élément <option> est créé pour chaque ligne de la colonne nom de la table GeoJSON :

  — Tout d'abord une variable *option* est déclarée dans laquelle seront enregistrés tous les éléments <options> du menu déroulant.

  ```
  var options = '';
  ```

  — Une seconde variable *size* permet de calculer le nombre de lignes, c'est-à-dire le nombre de noms repris dans la base de données.

  ```
  var size = Object.keys(jsonData).length;
  ```

  — On réalise ensuite une boucle de type *for*. À chaque itération, un élément <option> est stocké dans la variable *option*. La boucle terminée, la variable *option* est ajoutée au menu déroulant via la méthode *append* en fournissant l'identifiant du menu déroulant : idNom.

  ```
  for (var i = 0; i < size; i++) {
  // boucle passant en revue les lignes du résultat de datalist
          option += '<option value="' + jsonData[i].nom + '">'
          + jsonData[i].nom + '</option>';
  //variable option, augmentée a chaque itération de la boucle d'une nouvelle
  //<option> correspondant a un nom d'objet 3D
  }
  $("#idNom").append(option);
  //ajout des élèments <options> au menu déroulant contenant le nom des segments.
  ```

4.2.2.2 Etape 2 : Transmission du choix de l'utilisateur

La création d'un bouton *affichage*, en dessous de la structure HTML des deux menus déroulants, permet à l'utilisateur de valider sa sélection au sein des menus déroulants et d'enclencher la fonction *object3D()*.

```
<button id="affichage" onclick="object3D()">Affichage</button>
```

Dans la fonction *object3D()* (l'ensemble de la fonction se trouve en annexe D dans le script sig3d.html) :

— Deux variables *typeGeom* et *nom* permettent d'enregistrer les valeurs du nom et du type de géométrie entrées par l'utilisateur en utilisant la méthode *getElementByID()*.

— Une méthode AJAX du type GET est ensuite utilisée. Les deux variables précédentes sont fournies en paramètres et transmis au script object3D.php (Voir annexe G).

— Ce script PHP commence par enregistrer la valeur de l'attribut et le type de géométrie. Après s'être connecté à la base de données, le code est alors divisé en quatre conditions

en lien avec les quatre scénarios qu'il est possible de rencontrer. À chaque condition correspond une requête SQL. Il est nécessaire d'utiliser une fonction d'agrégat, *JSONBAGG()*, dans le cas où plusieurs segments répondraient à la condition. La fonction d'agrégat permet de renvoyer un tableau en format JSON contenant un ou plusieurs GeoJSON.

Donc, soit l'utilisateur :

— Spécifie le segment qu'il souhaite importer sur base d'un attribut uniquement. La valeur de « attribut » est donc différente de la valeur par défaut « ALL » :

```
SELECT JSONB_AGG(datajson) AS datatext  FROM geojson where nom = '$nom'
```

— Spécifie le segment qu'il souhaite importer sur base uniquement du type de géométrie :

```
SELECT JSONB_AGG(datajson) AS datatext FROM geojson where type = '$typeGeom'
```

— Spécifie le segment qu'il souhaite importer sur base d'un attribut et d'un type de géométrie :

```
SELECT JSONB_AGG(datajson) AS datatext  FROM geojson
where type = '$typeGeom' AND nom = '$nom'
```

— Ne spécifie ni un attribut ni un type de géométrie, et il souhaite importer l'ensemble des segments contenus dans la base de données :

```
SELECT JSONB_AGG(datajson) AS datatext  FROM geojson
```

En cas de succès de la méthode AJAX et donc du script PHP, une fonction récupère et enregistre les résultats en format JSON dans la variable *data*. Ensuite, la fonction *DrawThreeGeo(json)* est enclenchée. Cette dernière permet de créer un objet 3D à partir d'un segment en format GeoJSON (contenant l'information spatiale et sémantique de chaque segment) à l'aide de three.js.

4.2.2.3 Etape 3 : Affichage des objets 3D

La fonction *DrawThreeGeo(json)* a été écrite dans le script DrawGeoJSON.js et la fonction complète se trouve en annexe H. Pour pouvoir utiliser le script, il ne faut pas omettre d'inclure la dépendance dans la page web sig3d.html.

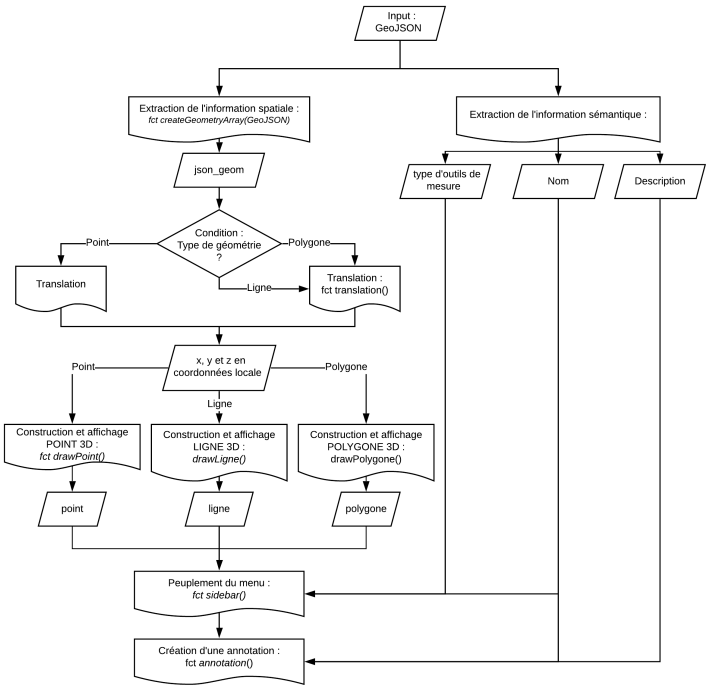


FIGURE 4.6: Schéma du déroulement de la fonction DrawThreeGeo(json).

Comme illustré sur la figure 4.6, la fonction prend tout d'abord en entrée un GeoJSON. À partir de ce dernier, un tableau *json_geom* est créé. Celui-ci va contenir les coordonnées de la géométrie, ainsi que le type de géométrie. Ce traitement est réalisé par la fonction *createGeometryArray(GeoJSON)* écrite dans le script *DrawGeoJSON.js*.
Ensuite, selon le type de géométrie contenu dans le tableau *json_geom*, trois conditions sont créées. On constate que dans les trois cas de figure une opération de translation est d'abord réalisée.

La raison pour laquelle une translation des coordonnées doit être réalisée est que pour afficher des objets vectoriels 3D, il faut définir la position des sommets d'un objet. Ceci est réalisé à partir de la classe Vector3 permettant de stocker les coordonnées des sommets. Cependant, les coordonnées issues d'un nuage de points géoréférencé exigent une très haute précision qui

dépasse généralement les limites des nombres à virgule flottante de précision 32 bits que possède la classe Vector3. Par exemple, les coordonnées en X, selon la projection Lambert 2008, varient entre 517579,6608 et 797118,5854. Les types à virgule flottante ont une haute précision pour des valeurs proches de 0, mais ne peuvent pas traiter avec précision de telles valeurs élevées. Pour que l'objet s'affiche correctement, il est nécessaire d'effectuer une translation vers un système de coordonnées locale ce qui permet de réduire le nombre de bits nécessaire. Sinon, le problème se traduit par un clignotement (« flickering ») des objets vectoriels 3D affichés lors d'une navigation à travers la scène de Potree. Le problème n'apparaît pas pour la visualisation du nuage de points, car Potree effectue également une translation de tous les points.

Pour fixer le problème, on doit :

- Calculer les coordonnées minimums en x, y et z. Cette tâche est réalisée par la fonction *getMin(number)*.
- Soustraire ces valeurs minimums à chaque coordonnée avant de les stocker dans la classe *Vector3*. La fonction *translation(coordinates)* réalise cette opération et retourne les coordonnées translatées : *x_trans*, *y_trans* et *z_trans*. Comme décrit dans la figure 4.6, dans le cas d'un point, il n'est pas nécessaire de faire appel à cette fonction car la valeur de la soustraction sera dans tous les cas égal à zéro.
- Fixer l'origine de chaque objet vectoriel 3D créé à partir de ces valeurs minimums. Ceci sera réalisé ultérieurement dans les fonctions *drawPoint/Line/Polygone*.

Une fois l'opération de translation réalisée, il s'agit de créer et d'afficher sur la scène les trois types de géométrie. Les fonctions *drawPoint, drawLine et drawPolygon* sont chargées de réaliser ces opérations. Ces trois fonctions présentent la même structure avec des spécificités propre à chaque géométrie. Elles se déroulent de la manière suivante :

- Tout d'abord, le matériel est défini :
  - Point :
    ```
    var pointMaterial = new THREE.PointsMaterial({color: 0Xff000});
    ```
  - Ligne :
    ```
    var lineMaterial = new THREE.LineBasicMaterial({color: 0X00ffff});
    ```
  - Polygone :
    ```
    var meshMaterial = new THREE.MeshStandardMaterial({
        color : 0x0055ff,
        side : THREE.DoubleSide
    });
    ```

— Le type *Geometry* est déclaré :

```
var point/line/meshGeom = new THREE.Geometry();
```

— Les coordonnées translatées des sommets sont stockées dans la classe *Vector3* et chaque sommet est ensuite ajouté à la géométrie :

  — Point :

  ```
  pointGeom.vertices.push(new THREE.Vector3(0,0,0));
  ```

  — Dans le cas des géométries ligne et polygone, on fait appel à la fonction *createVertexForEachPoint(objectGeometry, xAxis, yAxis, zAxis)* :

  ```
  function createVertexForEachPoint(objectGeometry, xAxis, yAxis, zAxis) {
      for (var i = 0; i < xAxis.length; i++) {
          objectGeometry.vertices.push
              (new THREE.Vector3(xAxis[i],yAxis[i], zAxis[i]));
          objectGeometry.faces.push(new THREE.Face3(0, i + 1, i));
          }
      }
  ```

  Comme on peut le voir dans le code ci-dessus, il est également nécessaire de créer des faces dans le cas d'un polygone via la classe *THREE.Face3(indice sommet)*. Pour rappelle, les indices des sommets doivent être déclarés dans le sens anti-horlogique pour être visible.

— Les constructeurs adéquat à chaque géométrie sont déclarés. Ceux-ci permettent d'associer la géométrie et le matériel, défini lors des étapes précédentes, et ainsi de créer l'objet vectoriel 3D :

  — Point :

  ```
  var point = new THREE.Points(pointGeom, pointMaterial);
  ```

  — Ligne :

  ```
  var line = new THREE.Line(lineGeom, lineMaterial);
  ```

  — Polygone :

  ```
  var mesh = new THREE.Mesh(meshGeom, meshMaterial);
  ```

— L'origine de chaque objet vectoriel 3D est fixée à partir des valeurs minimums calculées précédemment pour finaliser l'opération de translation :

```
point/line/mesh.position.set(x_min, y_min, z_min);
```

— Les objets vectoriels 3D sont ajoutés à la scène de Potree pour qu'ils y soient affichés :

```
scene.add(point/line/mesh);
```

— Enfin, les fonctions retournent les objets 3D pour être utilisé lors de l'avant-dernière opération c'est-à-dire le peuplement de la section scène dans le menu de Potree :

```
return point/line/mesh;
```

Le peuplement du menu de Potree des objets vectoriel 3D permet à l'utilisateur de pouvoir afficher ou désafficher les objets 3D sur la scène et de savoir quel objet a déjà été importé. La fonction *sidebar(object, nom, typeMesure)* se charge du peuplement du menu. En paramètre, la fonction utilise un objet vectoriel 3D que l'on vient de créer, ainsi que le nom de la géométrie et l'outil de mesure utilisé pour digitaliser le segment. Cette information est obtenue à partir du GeoJSON d'entrée. Pour réaliser le peuplement :

— En fonction de l'outil de mesure utilisé pour digitaliser le segment, on récupère l'icône de cet outil pour pouvoir l'utiliser comme symbole à la géométrie importée. Pour cela, on fournit le chemin du dossier qui contient les images des icônes (`C:\Apache24\htdocs\memoire\potree\build\potree\resources\icons`) et on l'enregistre dans la variable *icon* :

```
var icon;
       if (typeMesure == "Point"){
         icon = `${Potree.resourcePath}/icons/point.svg`;
       } else if (typeMesure == "Height") {
         icon = `${Potree.resourcePath}/icons/height.svg`;
       } else if (typeMesure == "Distance") {
         icon = `${Potree.resourcePath}/icons/distance.svg`;
       } else if (typeMesure == "Angle") {
         icon = `${Potree.resourcePath}/icons/angle.png`;
       } else if (typeMesure == "Area") {
         icon = `${Potree.resourcePath}/icons/area.svg`;
       }
```

— Ensuite, on ajoute les objets à la section en précisant l'icône.

```
viewer.onGUILoaded(() => { // GUI = Graphical User Interface
        let tree = $(`#jstree_scene`);
        let parentNode = "measurements";
        let Object3DID = tree.jstree('create_node', parentNode, {
          "text": nom,
          "icon" : icon
        },
          "last", false, false);
         //possibilité de rendre visible l'objet ou non
        tree.jstree(object.visible ? "check_node" : "uncheck_node", Object3DID);
      });
```

Enfin, la fonction *annotation(xPopup,yPopup,zPopup)* permet d'associer une annotation à chaque objet vectoriel 3D importé dans Potree. Le titre de l'annotation correspond au *nom* fourni par l'utilisateur et le contenu à la *description*.

```
function annotation(xPopup, yPopup, zPopup){
   let annotation = new Potree.Annotation({
     position: [xPopup, yPopup, zPopup],
     title: nom,
     description: description

   });
   viewer.scene.annotations.add(annotation);
 }
```

Pour la postion de l'annotation :

- Pour le point, les coordonnées de l'annotation correspondent logiquement à celles du point.
- L'annotation de la ligne est fixée à l'origine de la géométrie.
- Enfin, pour le polygone, la fonction *getCenter()* permet de déterminer le centre du polygone. La fonction retourne les coordonnées du centre du polygone sous la forme d'un Vector3.

4.2.3 Requête n°3 : requête spatiale

La troisième requête implémentée permet à l'utilisateur de réaliser une requête spatiale à partir des objets vectoriels 3D digitalisés et stockés dans la base de données spatiales.

Les fonctions spatiales dans PostGIS se présentent sous forme de commande SQL. Il est possible d'utiliser ces commandes directement à partir de l'interface d'administration pgAdmin du SGBD spatial. Dans notre cas, la commande SQL est insérée dans un script PHP. Nous allons utiliser la fonction spatiale 3D *St_3DDistance(geom1, geom2)* qui retourne la distance cartésienne minimum entre deux géométries. Cette commande va nous permettre de déterminer l'objet vectoriel 3D le plus proche spatialement d'un autre objet que l'utilisateur aura choisi.

La structure de la troisième requête est semblable à la seconde requête. Elle se déroule de la manière suivante :

- Une nouvelle sous-section "Requête spatiale" est créée. Elle contient un menu déroulant avec l'ensemble des noms des objets vectoriels digitalisés par l'utilisateur. Le mécanisme de peuplement du menu déroulant est le même qu'au point 4.2.2.1. Un bouton permet d'enclencher la fonction *closestObject()* écrite dans le script sig3D.html.

— La fonction *closestObject()* est identique à la fonction *object3D()*. À l'exception que cette nouvelle fonction récupère le nom du segment choisi par l'utilisateur dans le nouveau menu déroulant et que le nom du segment est transmis au script PHP getClosest_object.php qui contient la requête spatiale.

— Dans le script PHP, la requête spatiale en SQL est écrite :

```
SELECT JSONB_AGG(datajson) AS datatext
FROM geojson
GROUP BY geom
ORDER BY ST_3DDistance((SELECT geom FROM geojson WHERE nom = '$object')::geometry, geom)
OFFSET 1 FETCH NEXT 1 ROW ONLY
```

Cette requête est composée de plusieurs commandes SQL :

— La commande SELECT permet de retourner la colonne *datajson* contenant les informations liées à chaque objet en format GeoJSON. Il est nécessaire d'utiliser une commande d'aggrégat *JSONB_ADD()* dans le cas où plusieurs objets seraient retournés.

— La commande ORDER BY permet de classer les résultats de la commande St_3DDistance(geom1, geom2) en une liste en ordre ascendant.

— Dans la commande *St_3DDistance(geom1, geom2)*, la première géométrie correspond à la géométrie choisie par l'utilisateur. La seconde géométrie est l'ensemble de la colonne *geom* contenant la géométrie de chaque objet digitalisé.

— On calcule la distance entre l'objet sélectionné par l'utilisateur et tous les objets contenus dans la table. Ensuite, on classe le résultat par ordre ascendant. Le premier résultat de la liste correspondra à l'objet sélectionné par l'utilisateur, car l'objet le plus proche sera toujours l'objet lui-même. La commande OFFSET permet de spécifier le nombre de lignes à sauter avant de commencer à retourner les lignes de la liste. On décale donc le début de la liste à une ligne.

— La clause FETCH NEXT spécifie le nombre de lignes à retourner après le traitement de la clause OFFSET. On souhaite obtenir uniquement l'objet le plus proche donc une seule ligne.

— Le résultat de la requête spatiale en format GeoJSON est retourné à la fonction *closestObject()*. Enfin, la fonction *drawThreeGeo()* se charge d'afficher l'objet vectoriel 3D sur la scène de Potree.

5 Validation

Dans ce chapitre, le résultat principal, l'interface de l'application, est présenté. Une démonstration de l'utilisation de l'application est réalisée, permettant de vérifier que les attentes concernant les requêtes implémentées sont respectées. Enfin, les limitations de l'application sont abordées.

Au moment de la rédaction du présent document, l'application est accessible à l'adresse suivante : fsc-cloud66.segi.ulg.ac.be

5.1 Présentation de l'interface

L'interface de l'application peut être divisée en deux parties :

— Le menu de l'application composé du menu par défaut de Potree et de la nouvelle section *mémoire* implémentée (Partie gauche de la figure 5.1).

— La scène de visualisation du nuage de points (Partie droite de la figure 5.1) où les objets vectoriels 3D sont dessinés.

FIGURE 5.1: Interface de l'application.

5.1.1 Menu par défaut de Potree

Le menu par défaut de Potree est divisé en cinq sections : apparence, outils, scène, filtre de classification et à propos.

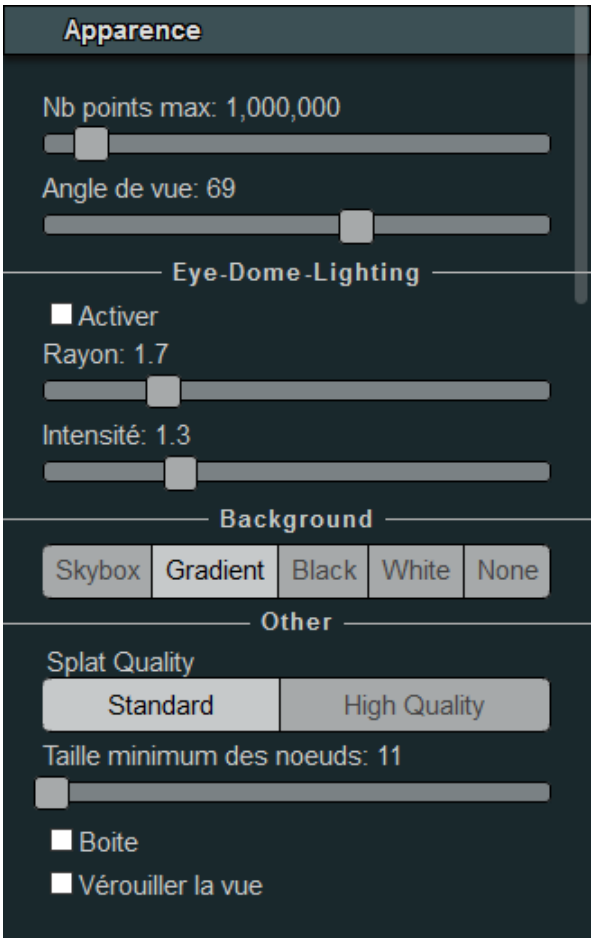


FIGURE 5.2: Section apparence.

Section apparence (du nuage de points) (figure 5.2) :

- **Nombre de points max** : définit le nombre de points stocké dans la mémoire. Les nœuds de l'Octree se chargent jusqu'à ce que le maximum soit atteint.
- **Angle de vue** : Permet le réglage de la caméra virtuelle. Des valeurs basses donnent un effet binoculaire. Des valeurs élevées peuvent déformer l'image étant donné que le modèle 3D est projeté sur un écran plat.
- **Eye-Dome-Lighting** : Technique qui permet d'améliorer la perception de la profondeur d'un nuage de points.
- **Backgroud** (Arrière-plan) : Choix de la couleur en arrière-plan du nuage de points.
- **Splat Quality** : En **Standard**, les points sont affichés selon des carrés, et en **High Quality**, les points sont affichés selon des cercles.

FIGURE 5.3: Section outils.

Section outils (figure 5.3) :

- **Mesure** :
  - Mesure d'angle dans un triangle.
  - Information sur un point (coordonnées).
  - Distance entre deux points ou plus.
  - Différence de hauteur entre deux points.
  - Calcul de l'aire d'une surface.
  - Mesure d'un volume, montre un cube avec ses dimensions (longueur, largeur et hauteur), définies par l'utilisateur.
  - Profil en long
- **Clipping** : Permet à l'utilisateur de se focaliser sur une région d'intérêt en mettant en évidence (« highligthing ») les points se trouvant à l'intérieur de la boîte, ou en affichant uniquement les points se trouvant à l'intérieur (« inside ») ou à l'extérieur (« outside ») de la boîte.
- **Navigation** :
  - Différents modes de navigation : orbitControls, FirstPersonControls, EarthControls.
  - Plusieurs angles de vue prédéfinis : vue d'en haut, d'en bas, etc.
  - Camera projection : choix de la vue : perspective ou orthographique (Explication détaillée au point 4.1.2.3.1).
  - Vitesse : détermine la sensibilité du mouvement du curseur de l'utilisateur.

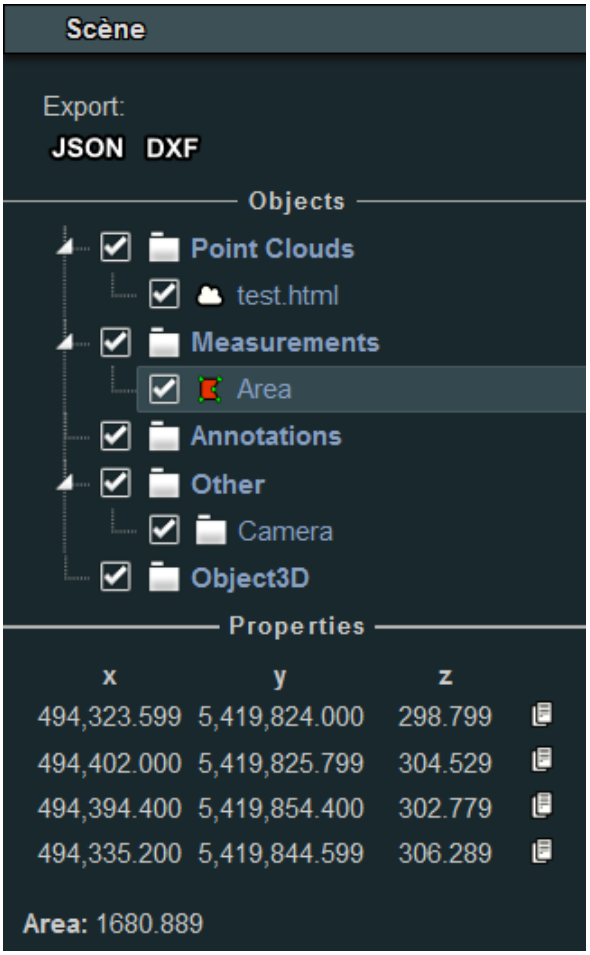


FIGURE 5.4: Section scène.

Section scène (figure 5.4) :

- **Export** : exporter en format JSON ou DXF les valeurs de la mesure affichées au sein de « properties ».
- **Objects** : possibilité d'afficher ou de désafficher : le nuage de points, les mesures réalisées, les annotations, les vues de caméra enregistrées et les objets 3D importés.
- **Properties** : affichage des coordonnées des sommets de la mesure et de la valeur de la mesure (dans l'exemple, mesure d'aire d'un carré).

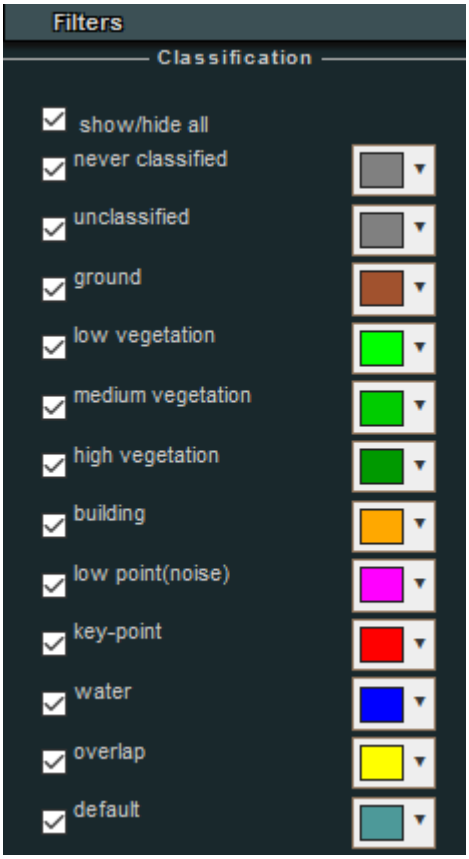


FIGURE 5.5: Section filtre.

Section filtre (figure 5.5) :

— Lorsque les données du nuage de points sont classifiées et que l'information est conservée lorsqu'on utilise PotreeConverter, il est possible d'appliquer des filtres.

5.1.2 Section mémoire

L'ensemble des requêtes ont été implémentées dans la section *mémoire*. Cette dernière est divisée en trois sous-sections : digitalisation, importation, et requête spatiale. Chacune de ses sous-sections est le résultat de l'implémentation d'une requête.

5.1.2.1 Sous-section : digitalisation

La sous-section digitalisation (Voir la figure 5.6) se compose :

— D'un champ de texte *nom* pour fournir un nom au segment digitalisé.

— D'un champ de texte *description* pour fournir une description au segment digitalisé.

— D'un bouton *digitalisation* pour valider le *nom*, la *description* et le segment digitalisé par l'utilisateur.

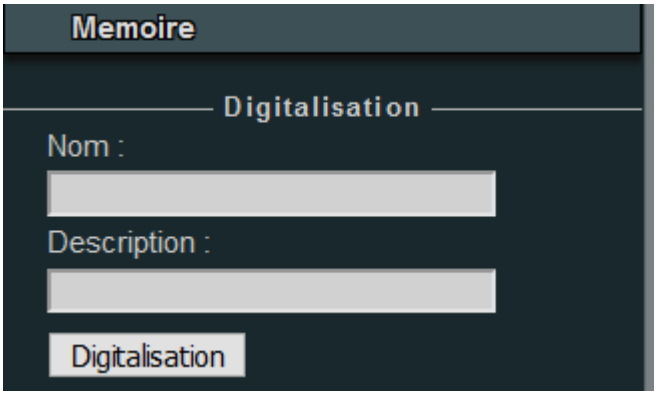


FIGURE 5.6: Sous-section digitalisation.

5.1.2.2 Sous-section : Importation

La sous-section importation (Voir la figure 5.7) se compose :

— D'un premier menu déroulant *nom* peuplé de l'ensemble des noms des segments digitalisés et stockés dans la base de donnée spatiales.

— D'un second menu déroulant *type de segmentation* contenant quatre options : point, ligne, polygone et géométrie (correspondant à l'option par défaut dans le cas où l'utilisateur ne spécifie pas le type de géométrie).

— D'un bouton *affichage* permettant d'afficher le.s segment.s répondant au critère choisi par l'utilisateur.

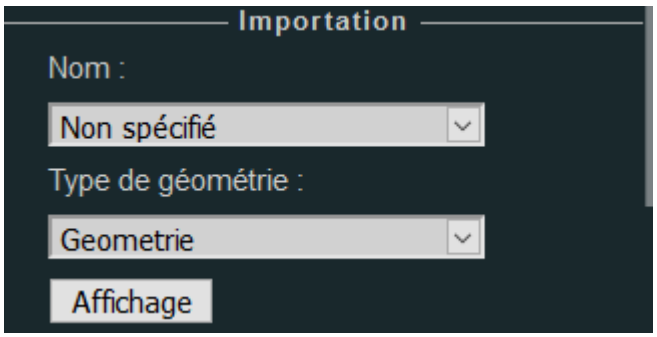


FIGURE 5.7: Sous-section importation.

5.1.2.3 Sous-section : Requête spatiale

La sous-section requête spatiale (Voir figure 5.8) se compose :

— D'un menu déroulant *Select Object* peuplé du nom des segments contenu dans la base de données spatiales.

— D'un bouton de validation *Closest Object* permettant d'afficher l'objet vectoriel 3D le plus proche spatialement par rapport à l'objet sélectionné dans le menu déroulant.

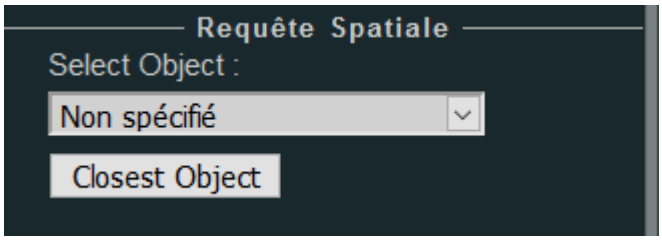


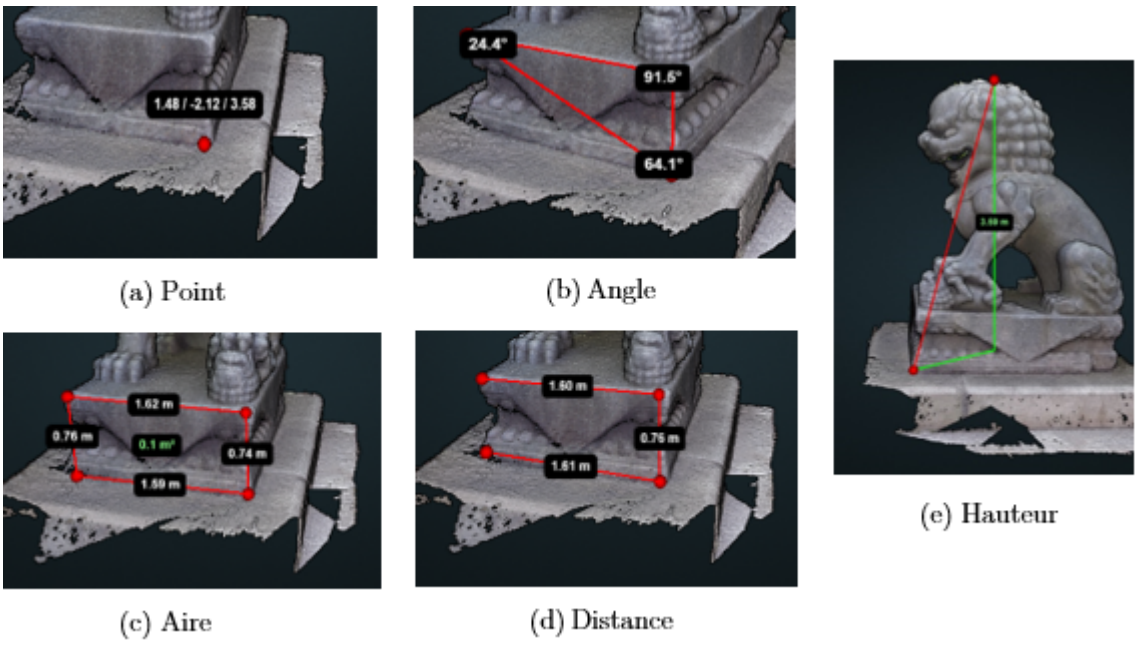
FIGURE 5.8: Sous-section requête spatiale.

5.2 Démonstration de l'application

Cette partie montre comment utiliser les nouvelles fonctionnalités implémentées.

5.2.1 Digitalisation

Pour digitaliser un segment, l'utilisateur utilise les outils de mesure de Potree, disponible dans la section outils, comme illustré sur la figure 5.9 ci-dessous.



(a) Point (b) Angle

(e) Hauteur

(c) Aire (d) Distance

FIGURE 5.9: Les différents outils de mesure permettant de digitaliser un segment.

Sur la figure 5.9 :

- L'outil point (a) permet de digitaliser un point.
- L'outil de mesure d'angle (b) permet de digitaliser un polygone correspondant uniquement à un triangle. En effet, lors de la mesure, après avoir choisi la position du deuxième sommet, une ligne de fermeture s'active automatiquement.
- L'outil de mesure d'aire (c) permet de digitaliser un polygone.
- L'outil de mesure de distance (d) permet de digitaliser une ligne ou une polyligne.
- L'outil de mesure de hauteur (e) permet de digitaliser uniquement une ligne (Celle en rouge).

Un avantage qu'offrent ces outils est qu'il est possible de déplacer les sommets (Point rouge) après avoir finalisé une mesure. Par exemple, après avoir digitalisé un carré à l'aide de l'outil d'aire (Figure 5.10 (a)), on se rend compte que le sommet dans le coin supérieur gauche n'est pas positionné à l'endroit que l'on souhaitait réellement (Figure 5.10 (b)). Il suffit de réaliser un clique gauche sur le sommet problématique et de le déplacer (Figure 5.10 (c)). À présent, le carré que l'on souhaitait digitaliser correspond bien à notre attente (Figure 5.10 (d)).

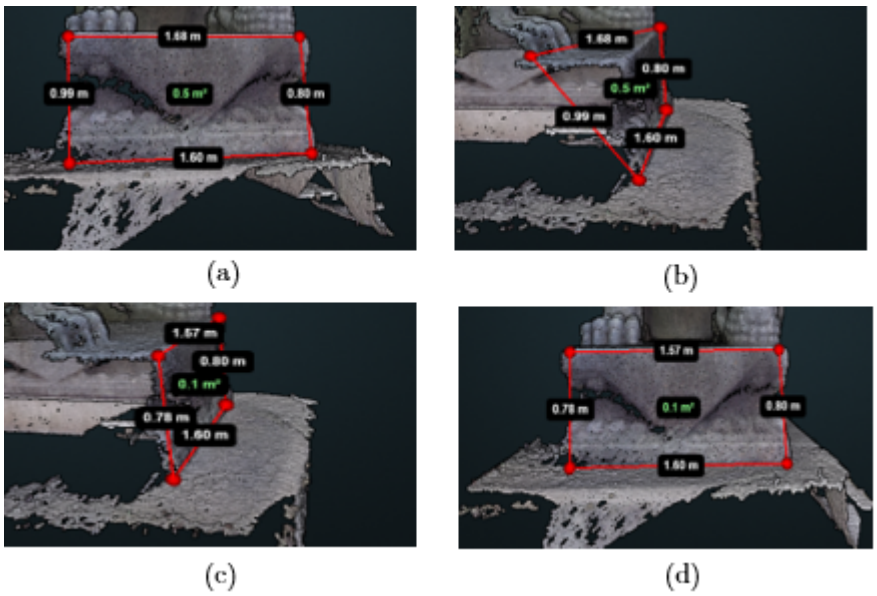


FIGURE 5.10: Déplacement d'un sommet d'une mesure.

Une fois le segment digitalisé, l'utilisateur procède à la classification en se rendant dans la sous-section *digitalisation* (Figure 5.6). Il fournit un nom et une description de son choix. Enfin, pour enregistrer le segment digitalisé au sein de la base de données, il suffit d'appuyer sur le bouton *digitalisation*. L'utilisateur peut s'assurer que la digitalisation a fonctionné en vérifiant que le *nom* du segment apparaît bien dans le menu déroulant *nom* dans la sous-section importation (Figure 5.11)

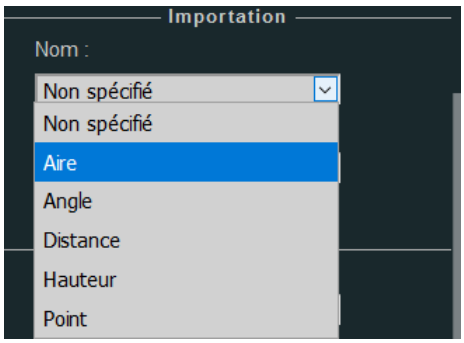


FIGURE 5.11: Le menu déroulant peuplé des segments digitalisés.

5.2.2 Importation

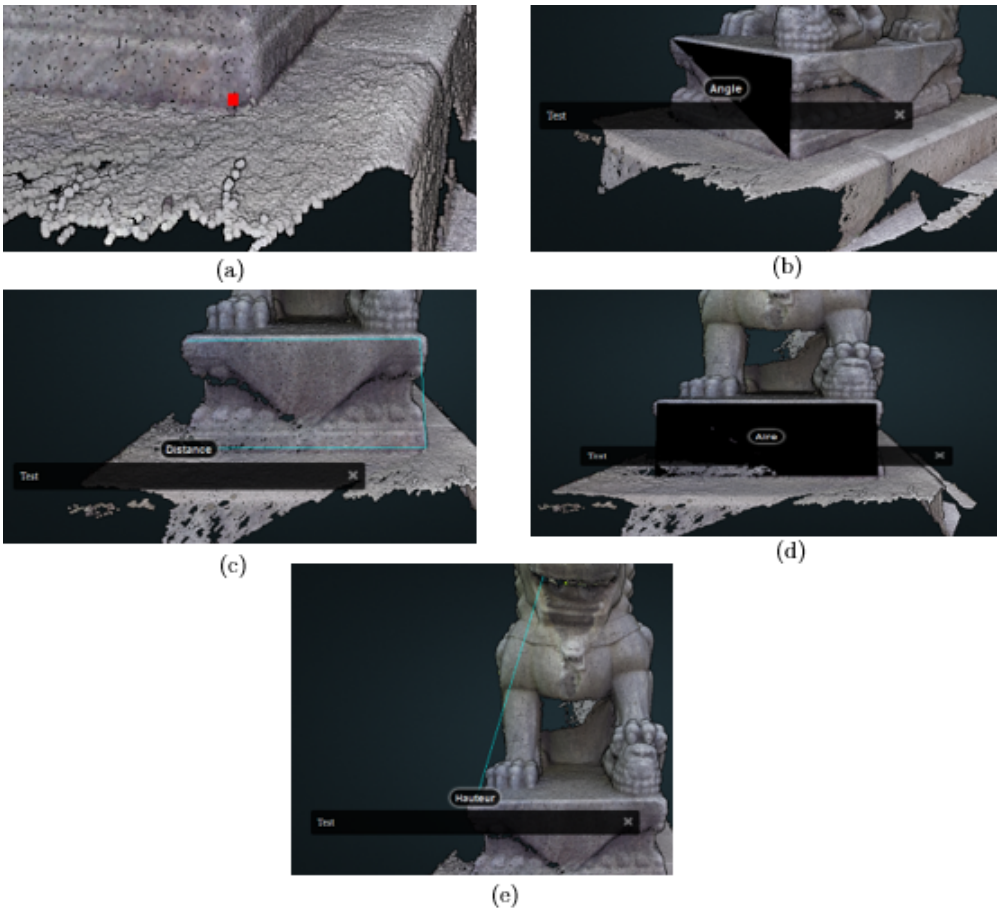
Pour afficher un segment de son choix dans la scène de Potree, il faut se rendre dans la sous-section *importation*. Au sein de celle-ci, l'utilisateur peut importer :

— L'ensemble des segments enregistré dans la base de données en appuyant directement sur le bouton *affichage*.

— Un seul objet en choisissant un nom parmi le menu déroulant *nom*.

— Tous les segments ayant une géométrie point, ligne ou polygone en spécifiant une de ces trois options.

Lorsqu'un segment est importé, la forme géométrique est affichée sur la scène de Potree (Figure 5.12). Chaque segment est accompagné d'une annotation, le titre de l'annotation correspond au nom du segment. Lorsqu'on passe la souris sur l'annotation, la description s'affiche et correspond à la description du segment.



(a) (b) (c) (d) (e)

FIGURE 5.12: Importation sur la scène de Potree : d'un point(a), d'un triangle(b), d'une polyligne(c), d'un polygone(d) et d'une ligne(e). Dans un souci de visibilité, l'annotation du point a été désaffichée à partir de la section scène.

Les segments importés peuvent être affichés ou désafichés à partir de la sous-section objects (Figure 5.13). La même chose peut être réalisée pour les annotations. Dans la liste, lorsqu'on

clique sur une annotation, dans la sous-section propriété, la position de l'annotation, le titre et la description de l'annotation y est affichés.

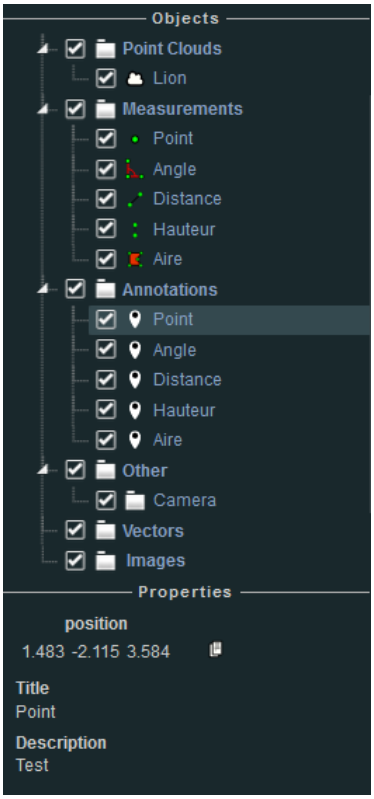


FIGURE 5.13: La sous-section object est peuplée par les objets vectoriels 3D et les annotations. Lorsqu'on clique sur une annotation, la partie propriété affiche le titre et la description.

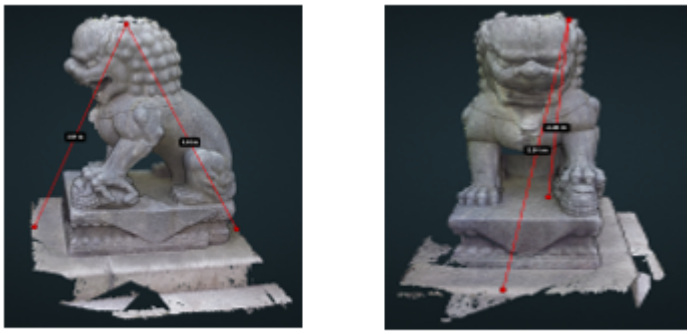
5.2.3 Requête Spatiale

Pour obtenir l'objet vectoriel 3D le plus proche spatialement d'un objet sélectionné. Dans la sous-section *requête spatiale*, l'utilisateur sélectionne le nom de l'objet 3D, à partir du menu déroulant *select object*, auquel il souhaite appliquer la requête spatiale. Après avoir validé sa sélection via le bouton *closest object*, le résultat de la requête affiche l'objet vectoriel 3D sur la scène de Potree. Le résultat de l'affichage est identique au point précédent 5.2.2 "Importation".

5.3 Limitation

Le test de profondeur d'OpenGL permet d'afficher des scènes 3D sur un écran 2D avec un effet de profondeur. Ce test est nécessaire pour faire en sorte qu'une géométrie plus éloignée

soit cachée derrière une géométrie plus proche. Ce test est appliqué au nuage de points, mais pas aux mesures. Par conséquent, ces dernières ne sont pas occultées par le nuage de points, ce qui les rends toujours visibles sur la scène. Comme on peut le voir sur la figure 5.14 ci-dessous, la mesure traverse le lion, mais reste visible.



(a) (b)

FIGURE 5.14: La mesure de distance n'est pas occultée par le nuage de points.

Le test de profondeur est également appliqué aux objets vectoriels 3D (Voir figure 5.15) pourtant issus des mesures sur le nuage de points. Le résultat visuel des objets vectoriels 3D peut donc différer des mesures, car ceux-ci peuvent être masqués en partie par le nuage de points. Ce qui peut créer une confusion chez l'utilisateur.

(a) (b)

FIGURE 5.15: L'objet vectoriel 3D issu de la mesure de distance est occulté par le nuage de points.

6 Conclusion et perspectives

La première partie de ce dernier chapitre conclut ce travail en énumérant les éléments importants. Les perspectives d'amélioration de l'application sont citées en deuxième partie.

6.1 Conclusion

Les moteurs de rendu de nuage de points basé sur le web offrent à un public large et varié de partager, d'afficher et d'explorer des nuages de points en fournissant une représentation détaillée et non filtrée du monde réel. Toutefois, pour pouvoir exploiter, par la suite, l'ensemble du potentiel de nuage de points au sein d'un SIG 3D, des objets vectoriels 3D, issus des données, doivent être disponibles. Un travail de segmentation est nécessaire pour obtenir ceux-ci.

Au travers de la littérature scientifique, nous avons analysé les composantes nécessaires pour la mise en place d'un SIG 3D web. Ensuite, nous avons constaté que la recherche scientifique s'attache à développer des algorithmes de segmentation automatique. Cependant, les différentes techniques ne permettent pas encore d'obtenir des résultats totalement satisfaisants. Le processus de segmentation automatique reste long et fastidieux, et nécessite toujours une intervention humaine. Nous nous sommes donc orientés vers une approche manuelle.

L'objectif du présent travail étant de stocker des objets vectoriels 3D digitalisés à partir d'un nuage de point pour pouvoir réaliser une analyse spatiale d'un site numérisé, nous avons formulé l'hypothèse suivante : "Développer un SIG 3D web, permettant de digitaliser des objets vectoriels 3D à partir de nuage de points et d'encoder manuellement de l'information sémantique, en incluant Potree, PostGIS et GeoJSON".

Pour réaliser l'application, une architecture trois tiers logique a été mise en place, celle-ci comprend :

— La strate application incluant un serveur web Apache, un module complémentaire PHP

et un convertisseur de nuage de point PotreeConverter.

— La strate client avec le navigateur web complété par Potree pour l'affichage de nuage de points et l'API three.js pour créer des objets vectoriels 3D, côté client, par-dessus le nuage de points.

— La strate données constituée du SGBD relationnelles PostgreSQL et son extension spatiale PostGIS permet de stocker l'information spatiale et sémantique, et d'accéder à des fonctions d'analyse spatiale.

Ensuite, trois requêtes ont été implémentées :

— La 1er requête permet à un utilisateur de digitaliser sur le nuage de points une géométrie (point, ligne, polygone) et d'associer de l'information sémantique à la géométrie. L'information spatiale et sémantique produite par l'utilisateur est stockée dans la base de données.

— La 2ème requête se charge d'importer et d'afficher sur la scène un objet vectoriel 3D issue de la base de données.

— La 3ème requête correspond à l'implémentation d'une fonction spatiale qu'offre PostGIS. Elle permet de déterminer l'objet 3D le plus proche spatialement d'un autre objet 3D.

Malgré les différences d'affichage entre les outils de mesure permettant la digitalisation et les objets vectoriels 3D, le résultat final est fonctionnel. La troisième requête démontre qu'il est possible de réaliser une requête spatiale à partir des objets vectoriels 3D produis par un utilisateur. L'outil de segmentation manuelle est ainsi adapté à tous les cas de figure qu'il est possible rencontré au sein des nuages de points (artefact, bâtiment, ville, environnement naturel, etc.). L'application est facile d'usage et permet à un utilisateur de créer une nouvelle information à partir d'un nuage de points. Enfin, le logiciel pourrait être utilisé pour produire des données d'entraînement nécessaire à la segmentation automatique basée sur le machine learning.

6.2 Perspectives de développement

Plusieurs pistes d'amélioration de l'application peuvent être envisagées :

— Dans la sous-section importation, il serait judicieux d'intégrer un bouton permettant à l'utilisateur de supprimer un segment stocké dans la base de données spatiales. En effet, en cas d'erreur lors de la sauvegarde d'un segment digitalisé, l'utilisateur n'a pas de possibilité de corriger son erreur.

— Après avoir sauvegardé un segment en cliquant sur le bouton digitalisation de faire apparaître directement l'objet vectoriel 3D sur la scène de Potree. Ce qui permettrait à l'utilisateur de vérifier que l'objet 3D créé correspond bien à ses attentes. Au lieu de devoir passer par la sous-section importation par la suite.

— Implémenter de nouvelles requêtes spatiales pour permettre de réaliser une analyse spatiale plus complète des objets vectoriels 3D.

— Pour la digitalisation de segment, ne plus utiliser les outils de mesure par défaut de Potree. Mais au sein de la sous-section digitalisation, sur base des outils point, distance et aire, créer trois outils permettant de digitaliser un point, une ligne et un polygone.

— Permettre la digitalisation de volume, une fois que ce type de géométrie sera supporté par le format GeoJSON.

— L'utilisateur ne puisse pas encoder un segment avec le même nom. Ce qui permettrait d'éviter toute confusion lors de la sélection du nom d'un segment lors de l'importation.

— Dans la sous-section digitalisation, il serait intéressant de permettre à l'utilisateur d'associer une couleur à chaque segment pour réaliser une classification visuelle.

Bibliographie

Agrawal, S. & Gupta, R. D. (2017). Web GIS and its architecture: a review. *Arabian Journal of Geosciences*, *10*(23), 518. doi :10.1007/s12517-017-3296-2

andrewharvey & mbostock. (2019). topojson. original-date: 2012-11-16T01:23:44Z. TopoJSON. Récupérée 25 novembre 2019, à partir de https://github.com/topojson/topojson

Ballard, D. H. (1981). Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, *13*(2), 111-122. doi :10.1016/0031-3203(81)90009-1

Bhanu, B., Lee, S., Ho, C. C. & Henderson, T. (1986). Range data processing: representation of surfaces by edges. *Proc. 8th International Conference on Pattern Recognition*. IEEE Computer Society Press, 236-238.

Billen, R., Jonlet, B., Luczfalvy Jancsó, A., Neuville, R., Nys, G.-A., Poux, F., ... Hallot, P. (2018). La transition numérique dans le domaine du patrimoine bâti: un retour d'expériences, 30. Récupérée à partir de http://hdl.handle.net/2268/228570

Bolstad, P. (2002). *GIS Fundamentals: A First Text on Geographic Information Systems*.

Bray, E. (2014). *The JavaScript Object Notation (JSON) Data Interchange Format* (rapp. tech. N° Rfc7159). Internet Engineering Task Force (IETF). Récupérée à partir de http://www.rfc-editor.org/info/rfc7158.

Burrough, P. (1986). Principles of geographical information systems for land resources assessment. *Geocarto International*, *1*(3), 54-54. doi :10.1080/10106048609354060

Butler, H., Daly, M., Doyle, A., Gillies, S., Hagen, S. & Schaub, T. (2016). *The GeoJSON Format* (rapp. tech. N° RFC7946). RFC Editor. doi :10.17487/RFC7946

Butler, H., Finnegan, D. C., Gadomski, P. J. & Verma, U. K. (2014). plas.io: Open Source, Browser-based WebGL Point Cloud Visualization. *AGU Fall Meeting Abstracts*, *2014*, IN23D-3749. Récupérée 2 août 2019, à partir de https://ui.adsabs.harvard.edu/abs/2014AGUFMIN23D3749B/abstract

Comaniciu, D. & Meer, P. (2002). Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(5), 603-619. doi :10.1109/34.1000236

Cowen, D. J. (1990). GIS versus CAD versus DBMS: what are the differences? doi :10.1201/b12579-11

Dambruch, J. & Krämer, M. (2014). Leveraging public participation in urban planning with 3D web technology. In *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies - Web3D '14* (p. 117-124). doi :10.1145/2628588.2628591

Deibe, D., Amor, M. & Doallo, R. (2019). Supporting multi-resolution out-of-core rendering of massive LiDAR point clouds through non-redundant data structures. *International Journal of Geographical Information Science*, *33*(3), 593-617. doi :10.1080/13658816.2018.1549734

Deibe, D., Amor, M., Doallo, R., Miranda, D. & Cordero, M. (2017). GVLiDAR: an interactive web-based visualization framework to support geospatial measures on lidar data. *International Journal of Remote Sensing*, *38*(3), 827-849. doi :10.1080/01431161.2016.1271476

Discher, S., Richter, R. & Döllner, J. (2019). Concepts and techniques for web-based visualization and processing of massive 3D point clouds with semantics. *Graphical Models*, *104*, 101036. doi :10.1016/j.gmod.2019.101036

Djemâ, N. (2018). Segmentation d'un nuage de points obtenu par scanner laser sur base d'un levé topographique classique, 64. Récupérée à partir de http://hdl.handle.net/2268.2/4317

Elseberg, J., Borrmann, D. & Nüchter, A. (2013). One billion points in the cloud – an octree for efficient processing of 3D laser scans. *ISPRS Journal of Photogrammetry and Remote Sensing*, *76*, 76-88. doi :10.1016/j.isprsjprs.2012.10.004

Euclideon Vault. (p. d.). Récupérée 6 novembre 2019, à partir de https://www.euclideon.com/vaultinfo/

Fischler, M. A. & Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, *24*(6), 381-395. doi :10.1145/358669.358692

Fu, P. & Sun, J. (2010). *Web GIS: principles and applications.* Esri Press.

Gao, L. N. Z., Nocera, L., Wang, M. & Neumann, U. (2014). Visualizing Aerial LiDAR Cities with Hierarchical Hybrid Point-Polygon Structures, 8. Récupérée à partir de https://dl.acm.org/doi/abs/10.5555/2619648.2619672

Goswami, P., Erol, F., Mukhi, R., Pajarola, R. & Gobbetti, E. (2013). An efficient multi-resolution framework for high quality interactive rendering of massive point

clouds using multi-way kd-trees. *The Visual Computer*, *29*(1), 69-83. doi :10.1007/s00371-012-0675-2

Goswami, P., Zhang, Y., Pajarola, R. & Gobbetti, E. (2010). High Quality Interactive Rendering of Massive Point Models Using Multi-way kd-Trees. In *2010 18th Pacific Conference on Computer Graphics and Applications* (p. 93-100). doi :10.1109/PacificGraphics.2010.20

Grilli, E., Menna, F. & Remondino, F. (2017). A review of point clouds segmentation and classification algorithms. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XLII-2/W3*, 339-344. doi :10.5194/isprs-archives-XLII-2-W3-339-2017

Ian, H. (2010). *An introduction to geographical information systems*. Pearson Education India.

Ingram, P. (1995). The World Wide Web. *Computers & Geosciences*. Internet, *21*(6), 799-816. doi :10.1016/0098-3004(95)00012-W

Jagannathan, A. & Miller, E. L. (2007). Three-Dimensional Surface Mesh Segmentation Using Curvedness-Based Region Growing Approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*(12), 2195-2204. doi :10.1109/TPAMI.2007.1125

Jiang, X., Meier, U. & Bunke, H. (1996). Fast range image segmentation using high-level segmentation primitives. In *Proceedings Third IEEE Workshop on Applications of Computer Vision. WACV'96* (p. 83-88). doi :10.1109/ACV.1996.572006

Kolbe, T. H., Gröger, G. & Plümer, L. (2005). CityGML: Interoperable Access to 3D City Models. In P. van Oosterom, S. Zlatanova & E. M. Fendel (Éd.), *Geo-information for Disaster Management* (p. 883-899). doi :10.1007/3-540-27468-5_63

Lavoué, G., Dupont, F. & Baskurt, A. (2005). A new CAD mesh segmentation method, based on curvature tensor analysis. *Computer-Aided Design*, *37*(10), 975-987. doi :10.1016/j.cad.2004.09.001

Lesecque, F. (2019). *Extraction et utilisation de connaissances métier structurées pour la classification de nuages de points 3D* (thèse de doct., Université de Liège, Liège, Belgique). Récupérée à partir de http://hdl.handle.net/2268.2/7363

Longley, P. A., Goodchild, M. F., Maguire, D. J. & Rhind, D. W. (2005). *Geographic information systems and science*. John Wiley & Sons.

Lu, X., Yao, J., Tu, J., Li, K., Li, L. & Liu, Y. (2016). PAIRWISE LINKAGE FOR POINT CLOUD SEGMENTATION. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *III-3*, 201-208. doi :10.5194/isprsannals-III-3-201-2016

Manferdini, A. M. & Remondino, F. (2010). Reality-Based 3D Modeling, Segmentation and Web-Based Visualization. In M. Ioannides, D. Fellner, A. Georgopoulos &

D. G. Hadjimitsis (Éd.), *Digital Heritage* (p. 110-124). Lecture Notes in Computer Science. doi :10.1007/978-3-642-16873-4_9

Martinez-Rubi, O., Verhoeven, S., van Meersbergen, M. & van Oosterom, P. (2015). Taming the beast: Free and open-source massive point cloud web visualization. Capturing Reality Forum, 12.

Maumont, M. (2010). L'espace 3D : de la photogrammétrie à la lasergrammétrie. *In Situ. Revue des patrimoines*, (13). doi :10.4000/insitu.6413

Nguyen, A. & Le, B. (2013). 3D point cloud segmentation: A survey. In *2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)* (p. 225-230). doi :10.1109/RAM.2013.6758588

Poux, F. [F.], Hallot, P., Neuville, R. & Billen, R. (2016). Smart point cloud : definition and remaining challenges. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *IV-2/W1*, 119-127. doi :10.5194/isprs-annals-IV-2-W1-119-2016

Poux, F. [Florent], Hallot, P., Jonlet, B., Carré, C. & Billen, R. (2014). Segmentation semi-automatique pour le traitement de données 3D denses : application au patrimoine architectural. *141*, 69-75. Récupérée à partir de http://hdl.handle.net/2268/173626

Rabbani, T., Van Den Heuvel, F. & Vosselman, G. (2007). Segmentation of point clouds using smoothness constraint. *The Photogrammetric Record*, *22*(117), 94-96. doi :10.1111/j.1477-9730.2007.00418.x

Reddy, G. P. O. (2018). Geographic Information System: Principles and Applications. In G. P. O. Reddy & S. K. Singh (Éd.), *Geospatial Technologies in Land Resources Mapping, Monitoring and Management* (p. 45-62). Geotechnologies and the Environment. doi :10.1007/978-3-319-78711-4_3

Remondino, F. & El-Hakim, S. (2006). Image-based 3D Modelling: A Review. *The Photogrammetric Record*, *21*(115), 269-291. doi :10.1111/j.1477-9730.2006.00383.x

Rigaux, P., Scholl, M. & Voisard, A. (2001). *Spatial databases: with application to GIS*. Elsevier.

Rusu, R. B. (2010). Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments. *KI - Künstliche Intelligenz*, *24*(4), 345-348. doi :10.1007/s13218-010-0059-6

Scheiblauer, C. & Wimmer, M. (2011). Out-of-core selection and editing of huge point clouds. *Computers & Graphics*, *35*(2), 342-351. doi :10.1016/j.cag.2011.01.004

Schutz, M. (2019). PotreeConverter. original-date: 2014-02-08T09:23:58Z. Récupérée 6 novembre 2019, à partir de https://github.com/potree/PotreeConverter

Schütz, M. (2016). *Potree: Rendering Large Point Clouds in Web Browsers* (thèse de doct., Technische Universität Wien, Wiedeń).

Stubkjær, E. (1997). The World Wide Web and university education in remote sensing. *ISPRS Journal of Photogrammetry and Remote Sensing*, *52*(6), 281-293. doi :10.1016/S0924-2716(97)00024-5

Vieira, M. & Shimada, K. (2005). Surface mesh segmentation and smooth surface extraction through region growing. *Computer Aided Geometric Design*, *22*(8), 771-792. doi :10.1016/j.cagd.2005.03.006

Vosselman, G., Gorte, B. G. H., Sithole, G. & Rabbani, T. (2004). Recognising structure in laser scanner point clouds. *46*, 33-38.

Wang, Q. & Kim, M.-K. (2019). Applications of 3D point cloud data in the construction industry: A fifteen-year review from 2004 to 2018. *Advanced Engineering Informatics*, *39*, 306-319. doi :10.1016/j.aei.2019.02.007

Yamauchi, H., Seungyong Lee, Yunjin Lee, Ohtake, Y., Belyaev, A. & Seidel, H.-P. (2005). Feature sensitive mesh segmentation with mean shift. In *International Conference on Shape Modeling and Applications 2005 (SMI' 05)* (p. 236-243). doi :10.1109/SMI.2005.21

Yang, C., Goodchild, M., Huang, Q., Nebert, D., Raskin, R., Xu, Y., ... Fay, D. (2011). Spatial cloud computing: how can the geospatial sciences use and help shape cloud computing? *International Journal of Digital Earth*, *4*(4), 305-329. Récupérée à partir de https://www.tandfonline.com/doi/full/10.1080/17538947.2011.587547

Yang, C., Raskin, R., Goodchild, M. & Gahegan, M. (2010). Geospatial Cyberinfrastructure: Past, present and future. *Computers, Environment and Urban Systems*. Geospatial Cyberinfrastructure, *34*(4), 264-277. doi :10.1016/j.compenvurbsys.2010.04.001

Yu, L. & Gong, P. (2012). Google Earth as a virtual globe tool for Earth science applications at the global scale: progress and perspectives. *International Journal of Remote Sensing*, *33*(12), 3966-3986. doi :10.1080/01431161.2011.636081

Annexes

A Installation d'Apache

La procédure à suivre est la suivante :

— Avant de télécharger Apache, il est nécessaire de vérifier son environnement Windows. Il faut d'abord connaitre le processus de son système d'exploitation (32 bits ou 64 bits) (Paramètre de Windows -> Système -> Information système -> type du système). Puis, il faut s'assurer que le port 80 utilisé par Internet soit libre. Pour vérifier, ouvrir l'invite de commande et saisir la commande :

```
netstat -ano | find ":80"
```

Résultat :

```
TCP     0.0.0.0:80      0.0.0.0:0       LISTENING       5000
TCP     [::]:80         [::]:0          LISTENING       5000
```

L'avant dernière colonne montre l'état de la connexion et la dernière le numéro du processus qui utilise ce port (donc ici 5000). Si l'état *LISTENING* apparaît, cela signifie que le port 80 est utilisé. Il faut donc trouver quel est le processus correspondant qui l'utilise. Dans la console, entrer la commande suivante avec le numéro de processus (dans notre cas 5000) :

```
tasklist | find "numéro du processus"
```

Suite à cela, il faut désactiver le processus trouvé dans *les services de Windows*.

— Télécharger les fichiers binaires d'Apache pour Windows à l'adresse suivante : http://httpd.apache.org/docs/current/platform/windows.html#down. Les projets d'Apache Haus mettent à disposition les paquets nécessaires. Il est important de noter quelle version de Visual Studio (par exemple VC14) est utilisée pour compiler ces fichiers. Créer un répertoire *Apache24* sur le disque dur (`C:/Apache24`) et placer l'ensemble des fichiers à l'intérieur.

B Installation de PHP

Pour intégrer PHP au serveur web Apache :

- Télécharger les fichiers de PHP correspondant à la version de Visual Studio utilisée pour Apache à l'adresse suivante : https://windows.php.net/download. Ensuite, créer un répertoire *PHP* sur le disque dur et placer l'ensemble des fichiers à l'intérieur.
- Sur la colonne de gauche du lien PHP, télécharger et exécuter la version correspondante de Visual Studio.
- Pour configurer Apache, dans `C:/Apache24/`, ouvrir le fichier httpd.conf dans un éditeur de texte comprenant un affichage de coloration syntaxique de code source (Ex : notepad++, Atom, etc.), et modifier le chemin de la racine du serveur :

  ```
  ServerRoot "C:/Apache24"
  ```

  À cette étape, on peut tester Apache en lançant httpd.exe ou ApacheMonitor.exe dans `C:/Apache24/bin`. Ensuite, dans un navigateur web, si on saisit localhost, le contenu du fichier index.html qui est dans C :/Apache24/htdocs apparaît.
  A présent, on peut afficher tous les fichiers écrits en .html qui sont contenus dans C :/Apache24/htdocs à partir d'un navigateur web. En développement web, il est nécessaire de tester son site localement avant de le publier. Dans le domaine du DNS (« Domain Name System »), l'adresse spéciale appelée localhost fait référence au serveur situé sur l'ordinateur utilisé. Ceci permet d'accéder à un site situé sur localhost à partir d'un navigateur web, même sans connexion à Internet.
  Si on désire modifier le chemin des fichiers affichés par localhost (par défaut `C:/Apache24/htdocs`), il suffit d'indiquer dans le fichier httpd.conf le nouveau chemin en modifiant les lignes :

  ```
  DocumentRoot "${SRVROOT}/htdocs"
  <Diectory "${SRVROOT}/htdocs">
  ```

- Pour intégrer le module PHP dans Apache, il faut l'ajouter dans le httpd.conf (à la suite du bloc LoadModule) en précisant la version de PHP installée et en vérifiant

que le fichier php[version de php]apache[version d'Apache].dll est bien présent dans le répertoire C :/PHP :

```
LoadModule php7_module "c:/php7apache2_4.dll"
AddHandler application/x-httpd-php.php
```

— Ensuite, dans le répertoire PHP, deux fichiers de configuration sont présents php.ini-production et developpement. Renommer celui de production en php.ini. Préciser le chemin du fichier de configuration de PHP php.ini dans le httpd.conf (en bas du fichier par exemple).

```
PHPIniDir "c:/PHP"
```

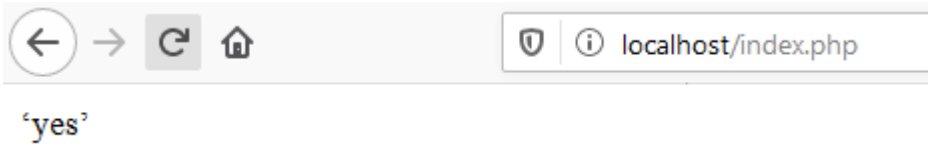
Dans le fichier php.ini, il faut préciser les chemins des extensions de PHP :

```
; On windows:
extension_dir = "C:/php/ext/"
```

— Pour vérifier que le module PHP a bien été installé, on crée un fichier index.php que l'on place dans le dossier htdocs. On l'ouvre dans un éditeur de texte et on entre la ligne :

```
<?php echo 'yes'; ?>
```

Ensuite, on redémarre Apache et dans un navigateur web, on entre l'adresse suivante : http ://localhost/index.php Le navigateur Web doit afficher :



Le serveur Web et le module PHP sont installés. On peut à présent afficher à partir d'un navigateur Web des pages Web écrites en .html et en .php.

C | Installation de PostgreSQL et PostGIS

Pour installer PostgreSQL et son extension spatiale PostGIS :

- Télécharger PostgreSQL à l'adresse suivante : http ://www.enterprisedb.com/products-services-training/pgdownloadwindows. Exécuter le programme et suivre les instructions. Dans le cadre de ce travail :
  - Dossier d'installation : C :/Program Files/PostgreSQL/10.
  - Port par défaut : 5432.
  - Le nom de l'utilisateur par défaut : postgres.
- Dans le répertoire bin (C :/Program Files/PostgreSQL/10/bin), exécuter le programme stackbuilder.exe. Le Stack Builder est un utilitaire qui permet d'installer des extensions de PostgreSQL. Les différentes catégories sont visibles sur la figure C.1. L'extension PostGIS se trouve dans la catégorie Spatial Extensions.

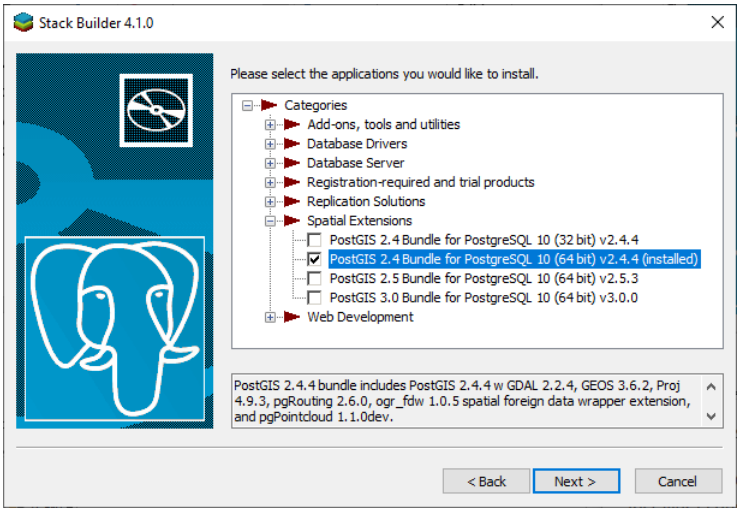


FIGURE C.1: Stack Builder

- Toujours dans le répertoire bin, créer un raccourci sur le bureau pour le programme pgAdmin[version].exe et l'exécuter. L'interface graphique de PostgreSQL, pgAdmin

s'ouvre.

La base de données est installée, il est à présent nécessaire de la connecter au serveur Web Apache et à PHP :

— Dans le fichier de configuration httpd.conf d'Apache, l'ouvrir dans un éditeur de texte et écrire à la fin du fichier :

```
LoadFile "C:/Program Files/PostgreSQL/10/bin/libpq.dll
```

Ce qui permet d'indiquer à Apache de se connecter à PostgreSQL en chargeant la bibliothèque qui se situe dans le répertoire bin de PostgreSQL : libpq.dll.

— Dans le fichier php.ini, on indique à PHP de charger la bibliothèque phppdopgsql.dll pour pouvoir coder en PHP Data Objects (PDO). Pour cela, supprimer le point-virgule de la ligne. Ce qui permet que la ligne soit exécuter et ne soit plus considéré comme un commentaire au sein du code :

```
extension = php_pdo_pgsql.dll
```

PDO permet d'établir une connexion avec PostgreSQL dans un script écrit en PHP.

Pour créer une base de données spatiales, on passe par la plateforme d'administration de PostgreSQL : pgAdmin. La base de données créée s'appelle potree et permet de stocker l'information spatiale et non-spatiale issue de l'utilisateur. Les étapes à suivre :

— Exécution de l'application pgAdmin.

— Une fois que la page est ouverte, dérouler l'onglet postgreSQL 10.

— Réaliser un clic droit sur l'élément database qui affiche un menu avec deux options, il faut choisir l'option Create -> Database...

— Une fenêtre Create – Database s'affiche. Au sein de l'onglet General, définir le nom de la base de données et le propriétaire. Enfin, valider la création de la base de données grâce au bouton save.

D page web : sig3d.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="description" content="">
  <meta name="author" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1.0,user-scalable=no">
  <title>Potree Viewer</title>

  <link rel="stylesheet" type="text/css" href="../build/potree/potree.css">
  <link rel="stylesheet" type="text/css" href="../libs/jquery-ui/jquery-ui.min.css">
  <link rel="stylesheet" type="text/css" href="../libs/openlayers3/ol.css">
  <link rel="stylesheet" type="text/css" href="../libs/spectrum/spectrum.css">
  <link rel="stylesheet" type="text/css" href="../libs/jstree/themes/mixed/style.css">
</head>

<body>
  <script src="../libs/jquery/jquery-3.1.1.min.js"></script>
  <script src="../libs/spectrum/spectrum.js"></script>
  <script src="../libs/jquery-ui/jquery-ui.min.js"></script>
  <script src="../libs/three.js/build/three.min.js"></script>
  <script src="../libs/three.js/extra/lines.js"></script>
  <script src="../libs/other/BinaryHeap.js"></script>
  <script src="../libs/tween/tween.min.js"></script>
  <script src="../libs/d3/d3.js"></script>
  <script src="../libs/proj4/proj4.js"></script>
  <script src="../libs/openlayers3/ol.js"></script>
  <script src="../libs/i18next/i18next.js"></script>
  <script src="../libs/jstree/jstree.js"></script>
  <script src="../build/potree/potree.js"></script>
  <script src="../libs/plasio/js/laslaz.js"></script>

<!-- INCLUDE ADDITIONAL DEPENDENCIES HERE -->
```

```
  <!-- Chemin du script "MyGeoJSONExporter" -->
  <script src="../sig3D/MyGeoJSONExporter.js"></script>
  <script src="../sig3D/DrawGeoJSON.js"></script>
  <!-- <script src="../sig3D/DrawGeoJSONClean.js"></script> -->
  <link rel="stylesheet" type="text/css" href="../sig3D/sig3D.css">

<!-- INCLUDE SETTINGS HERE -->

  <div class="potree_container"
        style="position: absolute; width: 100%; height: 100%; left: 0px; top: 0px; ">
        <div id="potree_render_area" style="background-image:
                url('../build/potree/resources/images/background.jpg');"></div>
        <div id="potree_sidebar_container"> </div>
  </div>

<script>
//**** Requête n°1 : digitalisation ****///
function digitalisation(){

    var name = document.getElementById('inputNom').value;
    var description = document.getElementById('inputDescription').value;

        console.log(viewer);

        var measurements = [];

        for (var i=0; i<viewer.scene.measurements.length;i++){
                measurements[i] = viewer.scene.measurements[i];
                measurements[i].nom = name;
                measurements[i].description = description;
                console.log(measurements[i]);


                if (measurements[i].name == "Angle" ){
                        measurements[i].alpha = measurements[i].angleLabels[0].text;
                        measurements[i].beta = measurements[i].angleLabels[1].text;
                        measurements[i].teta = measurements[i].angleLabels[2].text;
                } else if (measurements[i].name == "Area"){
                        measurements[i].area = measurements[i].areaLabel.text;
                } else if (measurements[i].name == "Height"){
                        measurements[i].height = measurements[i].heightLabel.text;
            }
```

```
        var geojson = Potree.MyGeoJSONExporter.toString(measurements[i]);

        $.ajax({ //Methode AJAX
                url: 'digitalisation.php', // adresse à laquelle la requête est envoyé
                data: {"geojson": geojson}, // Les données à envoyer au serveur
                dataType : 'text', // le type de données transmis au serveur
                // Peut être de type : XML, html, JSON, script
                type: "POST", // Le type de requête,
                                        // POST = envoie des données
                                        // GET = receptionne des données
                //La fonction à appeler si la requête a abouti
                success: function (geojson, status, xhr) {
                        // Permet de vérifier si la requête a été réalisé
                                alert('Data Send');
                                alert(geojson + " " + status + " " + xhr);
                        },
                //La fonction a appeler si la requête n'a pas abouti
                error: function (req, status, error) {
                // Permet de savoir le type d'erreur
                                alert(req + " " + status + " " + error);
                        }
                 });
          }
        }

window.viewer = new Potree.Viewer(document.getElementById("potree_render_area"));

viewer.setEDLEnabled(true);
viewer.setFOV(60);
viewer.setPointBudget(1*1000*1000);
viewer.loadSettingsFromURL();

viewer.setDescription("Segmentation manuelle");
// Chargement du "Graphic User Interface" (GUI)
viewer.loadGUI(() => {
        // Choix de la langue du menu
        viewer.setLanguage('en');
        // Permet de derouler directement l'onglet "apparence"
        //$("#menu_appearance").next().show();
                        viewer.toggleSidebar();

        //*** Personalisation du menu de Potree ***////
        // Création du nouvelle section ==> Memoire
        let section = $(`
```

```
        <h3 id="menu_memoire"><span>Memoire</span></h3>
        <div class="pv-menu-list"></div>
    `);
        let content = section.last();
        // Contenu html de la section
content.html(`

<!-- Sous-section : Digitalisation -->

<div class="divider"><span>Digitalisation</span></div>

<span>Nom : </span> <input type="text" id="inputNom">
<span>Description : </span> <input type="text" id="inputDescription">
<button id = "digitalisation"  onclick="digitalisation()">Digitalisation</button>

<!-- Sous-section : Importation -->

<div class="divider"><span>Importation</span></div>

<!-- Menu deroulant : Nom -->

<span>Nom :</span></br>
<select id="idNom"/></br>

<!-- Menu deroulant : Type de geométrie -->

<span>Type de géométrie :</span> </br>
<select id="typeGeom">
        <option value='ALLTYPE'>Geometrie</option>
        <option value='Point'>Point</option>
        <option value='LineString'>Ligne</option>
        <option value='Polygon'>Polygone</option>
</select> </br>

<button id="affichage" onclick="object3D()">Affichage</button>

<!-- Sous-section : Requête spatiale -->

<div class="divider"><span>Requête Spatiale</span></div>

<span>Select Object : </span>
<select id="spatial_request"/>
<button id = "ST_3DDistance" onclick="closestObject()">Closest Object</button>
```

```
        `);

// Déroulement de la section lors d'un click
section.first().click(() => content.slideToggle());
// Insertion de la nouvelle section avant la section "about"
section.insertBefore($('#menu_about'));



// Affichage du nuage de points : Maison
Potree.loadPointCloud("../pointclouds/lion_takanawa/cloud.js", "Lion", function(e){
    viewer.scene.addPointCloud(e.pointcloud);
        let material = e.pointcloud.material;
        material.size = 1;
        material.pointSizeType = Potree.PointSizeType.ADAPTIVE;
        viewer.fitToScreen();
});



////*** Requête n°2 : Objet 3D ****////

/////////Etape 1 : Allimentation du Menu deroulant : Nom ///////////////
$.ajax({
        url: 'menuDeroulant.php', // adresse à laquelle la requête est envoyé
        data: 'jsonData', // Les données reçu du serveur
        dataType: 'json', // le type de données reçu
        type: "get",
        // on veut obtenir les données de type JSON contenu dans le fichier menuDeroulant.php
        // Si succès de la méthode c-a-d si on a réussi à obtenir les données JSON alors
        // on rempli le menu déroulant
        // en créant à chaque fois un nouvelle élèment <option> correspondant à un nom
        success: function (jsonData) {
// variable option dans laquelle on enregistrera les différentes <options> du menu déroulant
        var option = '';
 //Variable reprenant le nombre de ligne du résultat de "datalist.php"
        var size = Object.keys(jsonData).length;
//première option, valeur par défaut = 'ALL', valeur affichée = 'Non spécifié'
        option += '<option value=' + 'ALL' + '>' + 'Non spécifié' + '</option>';
// boucle passant en revue les lignes du résultat de datalist
        for (var i = 0; i < size; i++) {
//variable option, augmentée a chaque itération de la boucle d'une nouvelle <option>
//correspondant a un nom d'objet 3D
        option += '<option value="' + jsonData[i].nom + '">' + jsonData[i].nom + '</option>';
```

```
                }
//ajout des élements <options> au menu déroulant contenant le noms des objets 3D
                $("#idNom").append(option);
                $("#spatial_request").append(option);
        },
        error: function(){
                alert('An error happened.');
                }
     });

});

// Etape 2 :
// Permet d'importer les données sur base de leur attribut et de leur type et de dessiner
// les formes en 3D sur le nuage de point
function object3D(){
// définition des variables reprenant les valeurs spécifiées dans les champs de sig_3d.html
    var typeGeom = document.getElementById('typeGeom').value;
        var idNom = document.getElementById('idNom').value;
        $.ajax({
        // chargement("get") des données à partir du fichier getType_Attribut.php
                url: 'object3D.php',
                // valeur en paramètre
                data: { 'typeGeom' : typeGeom, 'nom' : idNom},
                // type de données a charger ? ==> de type JSON
                dataType: 'json',
                // TYpe de requete ajax ==> "GET" donc on veut aller chercher des données
                type: "get",
                // si succès
                success: function (data) {
                        //alert('Data get !');
                        for (var i = 0; i<data.length; i++) {
                                //var object = Potree.Draw.GeoJSON(data[i]);
                                var object = Potree.Draw.GeoJSON(data[i]);
                        }
                },
                error: function (req, status, error) {
                        alert(req + " " + status + " " + error);

                }
        });
}
```

```
//**** Requête n°3 : Exemple de requête spatiale ==> Closest Object *****/////
function closestObject(){
        var object = document.getElementById('spatial_request').value;

        $.ajax({
                url: 'getClosest_Object.php',
                data: { 'Object' : object },
                dataType: 'json',
                type: "get",
                success: function (data,geojson) {

                    for (var i = 0; i<data.length; i++) {
                           var object = Potree.Draw.GeoJSON(data[i]); // dessine l'objet 3d
                    }
                },
            error: function (req, status, error) {
                        alert(req + " " + status + " " + error);
                }
        });
}

        </script>
  </body>
</html>
```

E MyGeoJSONExporter.js

```
Potree.MyGeoJSONExporter = class GeoJSONExporter {
   static measurementToFeatures (measurement) {
     let coords = measurement.points.map(e => e.position.toArray());

     let features = [];
     if (coords.length === 1) {
        let feature = {
        type: 'Feature',
                geometry: {
                        type: 'Point', //type de géométrie mesurer
                        coordinates: coords[0] //coordonnées de la mesure effectué
                    },
                properties: {
                // ajout des attributs non spatiale, rentrer par l'utilisateur, a l'objet GeoJSON
                        typeMesure: measurement.name,
                        nom: measurement.nom,
                        description: measurement.description
                    }
            };
        features.push(feature);
        } else if (coords.length > 1 && !measurement.closed  && measurement.name == "Height" ) {
          let object = {
                'type': 'Feature',
                'geometry': {
                        'type': 'LineString',
                        'coordinates': coords
                     },
                'properties': {
                        typeMesure : measurement.name,
                        nom: measurement.nom,
                        description: measurement.description,
                        height : measurement.height
                        }
```

```
        };
    features.push(object);
} else if (coords.length > 1 && !measurement.closed) {
        let object = {
                'type': 'Feature',
                'geometry': {
                'type': 'LineString',
                'coordinates': coords
                },
                'properties': {
                        typeMesure : measurement.name,
                        nom: measurement.nom,
                        description: measurement.description
                        }
                };

        features.push(object);
    } else if (coords.length > 1 && measurement.closed && measurement.name == "Angle") {
        let object = {
                'type': 'Feature',
                            'geometry': {
                                    'type': 'Polygon',
                                     'coordinates': [[...coords, coords[0]]]
                        },
                        'properties': {
                                typeMesure: measurement.name,
                                    nom: measurement.nom,
                                    description: measurement.description,
                                    alpha: measurement.alpha,
                                    beta: measurement.beta,
                                    teta: measurement.teta
                            }
        };
        features.push(object);
    } else if (coords.length > 1 && measurement.closed && measurement.name == "Area") {
                let object = {
                        'type': 'Feature',
                        'geometry': {
                            'type': 'Polygon',
                                'coordinates': [[...coords, coords[0]]]
                                },
                                'properties': {
                                        typeMesure: measurement.name,
                                        nom: measurement.nom,
```

```
  description: measurement.description,
                                area: measurement.area

                        }
                };
                features.push(object);
        }

        if (measurement.showDistances) {
                measurement.edgeLabels.forEach((label) => {
                        let labelPoint = {
                                type: 'Feature',
                                geometry: {
                                        type: 'Point',
                                        coordinates: label.position.toArray()
                                },
                                properties: {
                                        distance: label.text
                                }
                        };
                        features.push(labelPoint);
                });
        }

        if (measurement.showArea) {
                let point = measurement.areaLabel.position;
                let labelArea = {
                        type: 'Feature',
                        geometry: {
                                type: 'Point',
                                coordinates: point.toArray()
                        },
                        properties: {
                                area: measurement.areaLabel.text
                        }
                };
                features.push(labelArea);
        }

        return features;
}

static toString (measurements) {
        if (!(measurements instanceof Array)) {
```

```
                measurements = [measurements];
            }

            measurements = measurements.filter(m => m instanceof Potree.Measure);

            let features = [];
            for (let measure of measurements) {
                let f = Potree.MyGeoJSONExporter.measurementToFeatures(measure);

                features = features.concat(f);
            }

            let geojson = {
                'type': 'FeatureCollection',
                'features': features
            };

            return JSON.stringify(geojson, null, '\t');
        }
};
```

F digitalisation.php

```php
<?php
// <!-- Script à pour objectif d'insérer dans la BDD le fichier geojson digitaliser par
// l'uilisateur accompagné des deux attributs non-spatiale également spécifié ce dernier -->
$geojson=$_POST['geojson']; // Recupère l'objet envoyé par la méthode AJAX via la méthode $_POST
try
{
  // Connexion à la BDD
   $bdd = new PDO("pgsql:host=localhost;port=5432;dbname=potree","postgres","mot de passe");
   $bdd->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

   //////***** ETAPE 1 : On insere le geojson de type text dans la colonne dataText ******///////
   $req1 = $bdd->prepare('INSERT INTO geojson (dataText) VALUES(?)');
   $req1->execute(array($geojson));

   //////****** ETAPE 2  : Conversion du geojson(text) en jsontype (json) ****/////
   $req2 = $bdd->prepare('UPDATE geojson SET datajson = datatext::json');
   $req2->execute();

   /////**** ETAPE 3 : Creation de la geometrie *****////
   $req3 = $bdd->prepare('UPDATE geojson SET geom = ST_GeomFromGeoJSON((datajson ->
   \'features\' -> 0 -> \'geometry\')::text)');
   $req3->execute();
   $req4 = $bdd->prepare('UPDATE geojson SET geomtext = ST_AsText(geom)');
   $req4->execute();
   /////**** Etape 4 : Insertion des attributs non spatiale *****/////
   //// Insertion du nom
   $req5 = $bdd->prepare('UPDATE geojson SET nom = (datajson -> \'features\' -> 0 ->
   \'properties\' ->> \'nom\')');
   $req5->execute();
   /////Insertion de la description
   $req6 = $bdd->prepare('UPDATE geojson SET description = (datajson -> \'features\' -> 0 ->
   \'properties\' ->> \'description\')');
   $req6->execute();
```

```
    /////**** Etape 6 : Insertion du type via le fichier Geojson ****//////
    $req7 = $bdd->prepare('UPDATE geojson SET type = (datajson -> \'features\' -> 0 ->
    \'geometry\' ->> \'type\')');
    $req7->execute();

}
catch(Exception $e)
{
    echo "Connection a la BDD impossible : ", $e->getMessage();
 die();
}

?>
```

G| object3D.php

```php
<?php
// Permet d'aller chercher les données (en format geojson) digitaliser  dans la BDD
$typeGeom=$_REQUEST['typeGeom'];
$nom=$_REQUEST['nom'];

try {

  // connexion à la bdd
   $bdd = new PDO("pgsql:host=localhost;port=5432;dbname=potree","postgres","mot de passe");
   $bdd->exec("SET CHARACTER SET utf8");
   $bdd->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);


    //////***** ETAPE 1 : REQUETE ATTRIBUT  ******//////
    // Permet d'aller chercher les données en format Geojson
    /* Si l'utilisateur a spécifié un attribut
    mais n'a pas spécifié un type de geométrie*/
    if ($nom!='ALL')
    { //création du géojson avec la requete SQL
      $req = $bdd->prepare("SELECT JSONB_AGG(datajson)
                            AS datatext  FROM geojson where nom = '$nom'");
    }
    // Si uniquement le type de geométrie est spécifié par l'utilisateur
    else if ($typeGeom!='ALLTYPE')
    {
      $req = $bdd->prepare("SELECT JSONB_AGG(datajson) AS datatext
            FROM geojson where type = '$typeGeom'");
    }
    // Si le type ET l'attribut sont spécifiés
    else if ($typeGeom!='ALLTYPE' && $nom!='ALL')
    {
      $req = $bdd->prepare("SELECT JSONB_AGG(datajson) AS datatext  FROM geojson
                            where type = \'$typeGeom\' AND nom = \'$nom\'");
```

```
    }
    else if ($typeGeom =='ALLTYPE' && $nom =='ALL') {
      $req = $bdd->prepare("SELECT JSONB_AGG(datajson) AS datatext  FROM geojson");
    }

    $req->execute();

     while ($data=$req->fetch())
     {
       if($data)
       {
         $geojson = $data['datatext'];
         // pour verifier que la requete à bien fonctionné
         //et que la variable geojson contient bien les données souhaitées
         echo $geojson;
       }
       else
       {
         echo'erreur';
       }
     }
   }

catch(Exception $e)
{
    echo "Connection a la BDD impossible : ", $e->getMessage();
 die();
}

 ?>
```

H DrawGeoJSON.js

```
Potree.Draw.GeoJSON = function MydrawThreeGeo(json) {

  // Recupère la scène de Potree
  var scene = viewer.scene.scene;

 /////*** Création de la table qui va contenir les coordonnées de l'objet 3D ***//////

  var json_geom = createGeometryArray(json);

 /////****** Création des objets 3D
 // ==> 3 scénario possible : point, ligne, polygone *****/////
    for (var i=0; i<json_geom.length; i++) {

      ///**** Extraction de l'information sémantique dans le GeoJSON******//////
      var nom = json.features[0].properties["nom"];
      var typeMesure = json.features[0].properties["typeMesure"];
      var description = json.features[0].properties["description"];


      ///// Si le type de geométrie est un POINT
        if (json_geom[i].type == 'Point') {
            var x = json_geom[0].coordinates[0];
            var y = json_geom[0].coordinates[1];
            var z = json_geom[0].coordinates[2];

            var point = drawPoint(x,y,z);
            sidebar (point, nom, typeMesure);
            annotation(x,y,z);

      ////// Si le type de géométrie est une LIGNE
        } else if (json_geom[i].type == 'LineString') {

            var x_values = [];
```

```
        var y_values = [];
        var z_values = [];

      for (var i = 0; i< json_geom[0].coordinates.length; i++) {
        // parseFloat ?
        //besoin de convertir tout les éléments du tableaux en type "number"
        x_values.push(parseFloat(json_geom[0].coordinates[i][0]));
        y_values.push(parseFloat(json_geom[0].coordinates[i][1]));
        z_values.push(parseFloat(json_geom[0].coordinates[i][2]));
      }


      var x_trans = translation(x_values);
      var y_trans = translation(y_values);
      var z_trans = translation(z_values);

      ////// ****** Recherche de chaque valeur minimum
      // en x, y et z pour fixer l'origine de la ligne  ********//////////

        var x_min = getMin(x_values);
        var y_min = getMin(y_values);
        var z_min = getMin(z_values);

      var line = drawLine(x_trans, y_trans, z_trans, x_min, y_min, z_min);

      sidebar (line, nom, typeMesure, x_min, y_min, z_min);
      annotation(x_min, y_min, z_min);

//// Si le type de géométrie est un POLYGONE
 } else if (json_geom[i].type == 'Polygon') {

        var x_values = [];
        var y_values = [];
        var z_values = [];

        for (var i = 0; i< json_geom[0].coordinates[0].length; i++) {
          x_values.push(parseFloat(json_geom[0].coordinates[0][i][0]));
          y_values.push(parseFloat(json_geom[0].coordinates[0][i][1]));
          z_values.push(parseFloat(json_geom[0].coordinates[0][i][2]));
        }

        var x_trans = translation(x_values);
        var y_trans = translation(y_values);
        var z_trans = translation(z_values);
```

```
////// ****** Recherche de chaque valeur minimum en x, y et z
// pour fixer l'origine du polygone ********//////////

            var x_min = getMin(x_values);
            var y_min = getMin(y_values);
            var z_min = getMin(z_values);

            var polygon = drawPolygone(x_trans, y_trans, z_trans, x_min, y_min, z_min);

            // Calcul des coodonnées du centre du mesh
            // pour positionner le popup au centre
            var polygonCenter = getCenterPoint(polygon);
            var xCenter = polygonCenter.x + x_min;
            var yCenter = polygonCenter.y + y_min;
            var zCenter = polygonCenter.z + z_min;

            sidebar (polygon, nom, typeMesure, xCenter, yCenter, zCenter);
            annotation(xCenter, yCenter, zCenter);
        }
    }

    function drawPoint(x, y, z) {
             /// Définition du materiel
             var pointMaterial = new THREE.PointsMaterial({color: 0Xff0000,
                                  size: 0.05});
             /// Utilisation du type "Geometry"
             var pointGeom = new THREE.Geometry();
             // Déclaration de la position (x,y,z) du seul sommet
             // Atention la classe Vector 3 possède une précision float

             pointGeom.vertices.push(new THREE.Vector3(0,0,0));
              // 0,0,0 ==> valeur de ma translation

             var point = new THREE.Points(pointGeom, pointMaterial);
             point.position.set(x,y,z); // fixe l'origine du point
             scene.add(point); // On ajoute le point à la scène

             return point;
      }

    function drawLine(x_trans, y_trans, z_trans, x_min, y_min, z_min) {

          var lineMaterial = new THREE.LineBasicMaterial({color: 0x00ffff});
```

```
        /// Utilisation du type "Geometry"
        var lineGeom = new THREE.Geometry();
        // Appelle de la fonction permettant de créer les sommets pour chaque point
        createVertexForEachPoint(lineGeom, x_trans, y_trans, z_trans);

        var line = new THREE.Line(lineGeom, lineMaterial);

        // origine de la ligne fixé aux coordonnées minimale en x y et z de l'objet 3D
        // ==> permet d'eviter le problème du "flickering"
        line.position.set(x_min, y_min, z_min);

        scene.add(line);

        return line;
}


function drawPolygone(x_trans, y_trans, z_trans, x_min, y_min, z_min) {

        var meshMaterial = new THREE.MeshStandardMaterial( {
                color: 0x0055ff,
                shading: THREE.FlatShading,
                metalness: 0.2,
                side: THREE.DoubleSide
        });


      // Utilisation du type "Geometry"
      var meshGeom = new THREE.Geometry();
      // Création des sommets et des faces du polygone
      createVertexForEachPoint(meshGeom, x_trans, y_trans, z_trans);
      // Construction du mesh
      var mesh = new THREE.Mesh(meshGeom, meshMaterial);
      // On fixe la position du mesh aux coordonnées minimum du mesh
      mesh.position.set(x_min, y_min, z_min);

      scene.add(mesh);

      return mesh;
}

function createVertexForEachPoint(objectGeometry, xAxis, yAxis, zAxis) {
    for (var i = 0; i < xAxis.length; i++) {
```

```
      // Création des sommets pour l'objet 3D en paramètre de la fonction
      objectGeometry.vertices.push(new THREE.Vector3(xAxis[i],yAxis[i], zAxis[i]));
      // Création des faces de l'objet 3D (utile uniquement dans le cas d'un mesh)
      // indice des sommets doivent être déclaré
      //dans le sens anti-horlogique pour apparaitre
      objectGeometry.faces.push(new THREE.Face3(0, i + 1, i));
     }
}
function createGeometryArray(json) {
       // declaration d'un tableau pour contenir
       // les coordonnées et le type de géométrie UNIQUEMENT
        var geometryTypeArray = [];
        // on verifie que notre GeoJSON est valide
        if (json.type == 'FeatureCollection') {
             // boucle permettant de passer en revue toute l'information spatiale
            for (var i = 0; i < json.features.length; i++) {
                geometryTypeArray.push(json.features[i].geometry);
            }
         } else {
             alert ('The geoJSON is not valid.');
         }
         return geometryTypeArray;
}

function getMin(number){

  var min = number[0];

  for(var i = 0 ; i < number.length ; i++){
    if(number[i] < min) {
      min = number[i];
    }
  }
  return min;
}

function translation(coordinates){

  var coord_min = getMin(coordinates);

  var coord_trans = [];

  for (var i = 0; i<coordinates.length; i++) {
    coord_trans[i] = coordinates[i] - coord_min;
```

```
  }
  return coord_trans;
}


// Permet de déterminer le centre du
// polygone pour pouvoir afficher le popup au centre
function getCenterPoint(mesh) {
   var geometry = mesh.geometry;
   // calcul le cadre du polygone
   geometry.computeBoundingBox();
   // retourne le centre du cadre sous la forme d'un Vector3
   var center = geometry.boundingBox.getCenter();

   return center;
 }

 function sidebar (object, nom, typeMesure) {
   var icon;
   if (typeMesure == "Point"){
     icon = `${Potree.resourcePath}/icons/point.svg`;
   } else if (typeMesure == "Height") {
     icon = `${Potree.resourcePath}/icons/height.svg`;
   } else if (typeMesure == "Distance") {
     icon = `${Potree.resourcePath}/icons/distance.svg`;
   } else if (typeMesure == "Angle") {
     icon = `${Potree.resourcePath}/icons/angle.png`;
   } else if (typeMesure == "Area") {
     icon = `${Potree.resourcePath}/icons/area.svg`;
   }
   viewer.onGUILoaded(() => { // GUI = Graphical User Interface
       // Add entries to object list in sidebar
       let tree = $(`#jstree_scene`);
       let parentNode = "measurements";
       let Object3DID = tree.jstree('create_node', parentNode, {
         "text": nom,
         /*"data" : json,
         "object" : json,*/
         "icon" : icon
       },
         "last", false, false);
        //possibilité de rendre visible l'objet ou non
       tree.jstree(object.visible ? "check_node" : "uncheck_node", Object3DID);
     });
```

```
    }
  function annotation(xPopup, yPopup, zPopup){
    let annotation = new Potree.Annotation({
      position: [xPopup, yPopup, zPopup],
      title: nom,
      description: description

    });
    viewer.scene.annotations.add(annotation);
  }
}
```

I getClosestObject.php

```php
<?php
// Permet d'aller chercher les données (en format geojson) digitaliser  dans la BDD

$object=$_REQUEST['Object'];

try {

  // connexion à la bdd
   $bdd = new PDO("pgsql:host=localhost;port=5432;dbname=potree","postgres","mot de passe");
   $bdd->exec("SET CHARACTER SET utf8");
   $bdd->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);


    // Requete Spatiale
    if ($object!='ALL') {
      $req = $bdd->prepare("SELECT JSONB_AGG(datajson) AS datatext  FROM geojson GROUP BY geom
                            ORDER BY ST_3DDistance((SELECT geom FROM geojson WHERE
                            description = '$object')::geometry, geom)
                            OFFSET 1 FETCH NEXT 1 ROW ONLY");
    }


    $req->execute();

     while ($data=$req->fetch())
     {
       if($data)
       {
         $geojson = $data['datatext'];
         echo $geojson;
       }
       else
       {
```

```
            echo'erreur';
          }
        }
      }

catch(Exception $e)
{
    echo "Connection a la BDD impossible : ", $e->getMessage();
 die();
}

 ?>
```

J | menuDeroulant.php

```php
<?php
// Peuple la liste deroulante des attributs entré au préalable par l'utilisateur
$bdd=new PDO("pgsql:host=localhost;port=5432;dbname=potree","postgres","mot de passe");
$req = $bdd->prepare("SELECT nom FROM geojson ORDER BY nom");
$req->execute();
$tab = $req->fetchAll(PDO::FETCH_ASSOC);
//echo json_encode($tab);
$jsonData = json_encode($tab);
echo $jsonData
?>
```