Représentation et échange de données tridimensionnelles géolocalisées de la ville

Jeremy Gaillard

N° d'ordre NNT : 2018LYSE2023

THESE de DOCTORAT DE L'UNIVERSITÉ DE LYON

Opérée au sein de

L'UNIVERSITÉ LUMIÈRE LYON 2

École Doctorale : ED 512 Informatique et Mathématiques

Discipline : Informatique

Soutenue publiquement le 22 mai 2018, par :

Jérémy GAILLARD

Représentation et échange de données tridimensionnelles géolocalisées de la ville

Devant le jury composé de :

Paule-Annick DAVOINE, Professeure des universités, UNIVERSITE GRENOBLE ALPES, Présidente

Jean-Pierre JESSEL, Professeur des universités, Université Toulouse 3, Rapporteur

Sidonie CHRISTOPHE, Chargée de recherche H.D.R., Institut National Géographic, Rapporteure

Hugo LEDOUX, Doctoeur, Delft University of Technology, Invité

Vincent PICAVET, Expert, Invité

Gilles GESQUIERE, Professeur des universités, Université Lumière Lyon 2, Directeur de thèse

Adrien PEYTAVIE, Maître de Conférences, Université Claude Bernard Lyon 1, Co-Directeur de thèse

Contrat de diffusion

Année 2018

THÈSE

pour obtenir le grade de Docteur en Informatique

préparée au **Laboratoire d'InfoRmatique en Image et Systèmes d'information** (LIRIS)
et au sein de l'entreprise **Oslandia**

présentée et soutenue publiquement par

JÉRÉMY GAILLARD

le 22 MAI 2018

REPRÉSENTATION ET ÉCHANGE DE DONNÉES TRIDIMENSIONNELLES GÉOLOCALISÉES DE LA VILLE

sous la direction de

Gilles Gesquière** et **Adrien Peytavie

COMPOSITION DU JURY

Mme.	Sidonie Christophe	Rapporteur	(Chargé de Recherche)
M.	Jean-Pierre Jessel	Rapporteur	(Professeur)
Mme.	Paule-Annick Davoine	Examinateur	(Professeur)
M.	Gilles Gesquière	Directeur de thèse	(Professeur)
M.	Adrien Peytavie	Co-Directeur de thèse	(Maître de Conférence)
M.	Hugo Ledoux	Invité	(Associate Professor)
M.	Vincent Picavet	Invité	(Directeur d'Oslandia)

Abstract

Advances in 3D data acquisition techniques (laser scanning, photography, etc.) has led to a sharp increase in the quantity of available 3D geolocated data. More and more cities provide the scanned data on open access platforms. To ensure the intercompatibility of different data sources, standards have been developed for exchange protocols and file formats. Moreover, thanks to new web standards and the increase in processing power of personal devices, it is now possible to integrate rich content, such as 3D applications, directly in a web page. These two elements make it possible to share and exploit 3D city data into a web browser.

The subject of my thesis, co-financed by the Oslandia company, is the 3D representation of city data on the Web. More precisely, the goal is to retrieve and visualise a great quantity of city data from one or several distant servers in a thin client. This data is heterogenous: it can be 3D representations of buildings (meshes) or terrain (height maps), but also semantic information such as pollution levels (volume data), the position of bike stations (points) and their availability, etc. During my thesis, I explored various ways of organising this data in generic structures in order to allow the progressive transmission of high volumes of 3D data. Taking into account the multiscale nature of the city is a key element in the design of these structures.

Adapting the visualisation of the data to the user is another important objective of my thesis. Because of the high number of uses of 3D city models, the user's needs vary greatly: some specific areas are of higher interest, data has to be represented in a certain way... I explore different methods to satisfy these needs, either by priroritising some data over others during the loading stage, or by generating personalised scenesbased on a set of preferences defined by the user.

Résumé

Le perfectionnement des modes d'acquisition 3D (relevés laser, photographiques, etc.) a conduit à la multiplication des données 3D géolocalisées disponibles. De plus en plus de villes mettent leur modèle numérique 3D à disposition en libre accès. Pour garantir l'interopérabilité des différentes sources de données, des travaux ont été effectués sur la standardisation des protocoles d'échange et des formats de fichier. En outre, grâce aux nouveaux standards du Web et à l'augmentation de la puissance des machines, il est devenu possible ces dernières années d'intégrer des contenus riches, comme des applications 3D, directement dans une page web. Ces deux facteurs rendent aujourd'hui possible la diffusion et l'exploitation des données tridimensionnelles de la ville dans un navigateur web.

Ma thèse, dotée d'un financement de type CIFRE avec la société Oslandia, s'intéresse à la représentation tridimensionnelle de la ville sur le Web. Plus précisément, il s'agit de récupérer et de visualiser, à partir d'un client léger, de grandes quantités de données de la ville sur un ou plusieurs serveurs distants. Ces données sont hétérogènes : il peut s'agir de la représentations 3D des bâtiments (maillages) et du terrain (carte de hauteur), mais aussi d'informations sémantiques telles que des taux de pollution (volumes), la localisation de stations de vélos (points) et le nombre de vélos disponibles, etc. Durant ma thèse, j'ai exploré différentes manières d'organiser ces données dans des structures génériques afin de permettre une transmission progressive de fortes volumétries de données 3D. La prise en compte de l'aspect multi-échelle de la ville est un élément clef de la conception de ces structures.

L'adaptation de la visualisation des données à l'utilisateur est un autre grand axe de ma thèse. Du fait du grand nombre de cas d'utilisations existants pour la ville numérique, les besoins de l'utilisateur varient grandement : des zones d'intérêts se dégagent, les données doivent être représentées d'une manière spécifique... J'explore différentes manières de satisfaire ces besoins, soit par la priorisation de données par rapport à d'autres lors de leur chargement, soit par la génération de scènes personnalisés selon les préférences indiquées par l'utilisateur.

Remerciements

Cette thèse n'aurait pas été possible sans la confiance que m'ont accordée mes encadrants académiques et industriels, Gilles Gesquière, Adrien Peytavie, Olivier Courtin et Vincent Picavet. Je tiens à les remercier pour l'opportunité qu'ils m'ont fournie, ainsi que l'accompagnement et les conseils qu'ils m'ont prodigués au cours de ces trois années.

Je remercie également les membres de mon jury, Sidonie Christophe, Jean-Pierre Jessel, Paule-Annick Davoine et Hugo Ledoux, d'avoir accepter d'évaluer mon travail.

Ma thèse a été l'occasion de collaborer avec de nombreuses personnes, auprès desquelles j'ai beaucoup appris. Merci à toute l'équipe d'Oslandia dont l'expertise m'a apporté énormément. Merci à l'équipe d'iTowns que j'ai pu voir grandir et évoluer depuis ses débuts. Merci à l'équipe VCity du LIRIS, à mes co-doctorants Frédérique et Vincent, et aux stagiaires qui nous ont assistés dans nos projets. Merci aussi à Rémi et Alexandre dont les travaux ont permis de lancer ma thèse.

J'ai eu la chance d'être particulièrement bien entouré dès mon arrivé au laboratoire. Un grand merci à l'ensemble des personnes qui sont passées par le LIRIS-1 et qui, chacun à leur manière, ont contribué à en faire le meilleur bureau du laboratoire : JD, Lérémy, Abdoulaye, François, Matthieu, Alexis et Rémi. Merci aussi aux habitués du coin café qui ont rendu supportables les moments difficiles : Hélène (et ses gâteaux), Marie-Neige, David, Julie, Nicolas et Florian.

Merci aussi à tous mes amis en dehors du laboratoire qui m'ont permis d'oublier ma thèse de temps en temps. En plus de ceux que je vais inévitablement oublier (et que je mets en premier pour me faire pardonner), merci à Rémi, Florent, Michaël, Jérémy, Aurélie, Charlotte, Grégoire, Florence, JB, Gladys, Virgile, Pauline, Alexandre, Simon, Élodie, Romain, Mickaël, Thierry, Léa et Kévin.

Et finalement, merci à ma famille pour leur soutien lors de ma thèse et tout au long de ma scolarité, et tout particulièrement à ma mère pour avoir pris le temps de corriger l'anglais perfectible de mes publications.

Table des matières

1 Introduction

Sommaire

La manière dont nous utilisons l'information géospatiale a grandement évoluée au cours des précédentes décennies. L'avènement de l'informatique grand publique et d'internet a débouché sur une simplification de la navigation dans l'espace cartographique et une diffusion à grande échelle des cartes. Le GPS et les smartphones ont changé la manière dont nous nous repérons et nous déplaçons. Les applications cartographiques font désormais partie de notre quotidien.

Cette démocratisation de l'informatique géospatiale concerne principalement l'aspect 2D du domaine : les cartes. On peut alors naturellement se demander ce qu'une évolution vers le support de la troisième dimension pourrait apporter. Cela paraît d'autant plus intéressant qu'une carte est facilement représentable dans le monde réel, mais que son équivalent 3D — une maquette — est bien plus compliqué à construire.

L'introduction de la troisième dimension à l'informatique géospatiale permet de nouveaux cas d'utilisation, décrits de manière exhaustive dans [Bil+15]. Parmi ceux-ci, on peut citer notamment l'aide à la navigation, l'aménagement urbain, la simulation (d'inondation, de propagation des ondes, d'ensoleillement, etc.), l'analyse de visibilité, ou encore les visites virtuelles. On notera que les cas d'utilisation dépassent la simple visualisation. Même si la visualisation reste une application importante qui nécessite encore la levée de nombreux verrous, il faut garder à l'esprit que les informations 3D géospatiales ont d'autres usages.

La baisse des coûts d'acquisition fait que l'on dispose désormais d'une grande masse de données 3D. L'objectif de ma thèse est de permettre l'exploitation de cette abondance de données géospatiales par une variété de cas d'utilisation à travers un partage efficace des données avec les utilisateurs. Bien que l'augmentation de la puissance des processeurs et le développement du très haut débit facilitent le transfert des données et leur traitement sur des terminaux de faible puissance, le poids des données géospatiales 3D, leur complexité et la variété de leurs natures restent des difficultés à surmonter pour offrir une expérience optimale à l'utilisateur.

Le traitement des données géospatial 3D constitue un domaine de recherche vaste, à l'intersection de l'informatique et de la géographie. Dans cette thèse, je m'intéresse à un sous-ensemble de problèmes plus spécifiques : la visualisation, le transfert et la représentation de la ville sur le Web. En pratique, il s'agit de faciliter le

partage des données 3D de la ville, d'adapter la manière dont la ville est représentée à l'utilisateur et de permettre à l'utilisateur d'interagir avec les données distantes.

Dans la suite de ce chapitre, des éléments de contextes sont apportés au sujet de la thèse en elle-même et des domaines de l'informatique géospatiale, du Web et de la 3D.

1.1 Contexte

Ma thèse a bénéficié d'un financement de type CIFRE entre le laboratoire LIRIS et Oslandia, une société spécialisée dans les systèmes d'information géographique open source. Oslandia offre des prestations, des formations et du conseil autour de logiciels libres que l'entreprise édite où sur lesquels elle possède une expertise forte.

Le fait de travailler dans un cadre industriel amène à définir certains objectifs qui orientent la recherche.

Tout d'abord, du fait de la philosophie open source d'Oslandia, l'intégralité du code que j'allais produire devait être disponible librement et gratuitement sur des plates-formes de partage. De même, toutes les bibliothèques et applications utilisées dans le cadre de mes travaux devaient de préférence être open source.

Ensuite, il y a une volonté de la part de la société d'avoir des solutions génériques et interopérables. Cela implique notamment la nécessité de travailler avec des standards (de communication ou de formatage).

Dernièrement, on m'a encouragé à partir sur des solutions utilisant le système de gestion de base de données PostgreSQL assorti de l'extension PostGIS afin de valoriser les travaux de l'entreprise.

1.2 Notions d'informatique géographique

La cartographie et la géographie disposent d'un socle conceptuel solide, qui est naturellement réutilisé en informatique géographique. Si leurs concepts sont habituellement utilisés pour modéliser des informations en deux dimensions sur des cartes, ils restent valables pour les données tridimensionnelles et nous seront utiles pour définir des méthodes de visualisation pertinentes. Ci-après, je présente une partie de ces concepts qui seront employés dans la suite de ce manuscrit de thèse.

1.2.1 Définitions

Objet géographique Un objet géographique est une abstraction d'un phénomène réel [Her11]. Concrètement, il peut s'agir d'un bâtiment, d'un arbre, d'une route, etc. qui est représenté numériquement. Un objet géographique peut être représenté de plusieurs manières différentes : réaliste ou abstraite, en 2D ou en 3D, avec différents modèles de données... Il est généralement enrichi avec des informations sémantiques (par exemple, pour un bâtiment, son adresse, sa consommation énergétique ou sa date de construction).

Géospatial En informatique, le terme spatial peut se réfèrer à n'importe quel espace qui n'a pas forcément d'équivalent dans le monde réel. On parle de géospatial pour préciser qu'on se trouve dans espace géographique, où les coordonnées correspondent à une position précise sur la Terre.

Géolocalisé Un objet est dit géolocalisé s'il possède une position géographique, c'est-à-dire si ses coordonnées correspondent à sa position réelle sur la Terre (ou dans un système de projection donné).

Généralisation La généralisation consiste en l'adaptation de la représentation d'un ou d'un ensemble d'objets à l'échelle de visualisation [Rua99]. La généralisation permet de ne conserver que les traits essentiels d'une carte afin d'éviter la saturation de l'espace visuel. En informatique, elle permet aussi de réduire les ressources nécessaires pour afficher la carte.

Par exemple, sur une carte routière, à petite échelle, le tracé de la route sera moins précis, et les sentiers, les chemins et les plus petites routes ne seront pas représentés ; l'ensemble des bâtiments d'une agglomération sera abstrait par un polygon décrivant les contours de la ville (fig. 1.1).

Les processus de généralisations seront discutés plus en détails dans la partie 2.4.2.

Symbologie En cartographie, la symbologie désigne la manière d'encoder l'information sur la carte. En d'autres termes, elle définit la façon dont les objets sont représentés. La symbologie peut prendre plusieurs formes : icônes, couleurs, formes géométriques, etc (fig. 1.2). Un même objet peut-être représenté de manières différentes selon l'usage voulu.

Couche de données Les données géographiques sont souvent découpées en plusieurs couches (ou *calques* ou *layers* en anglais) qui peuvent être superposées

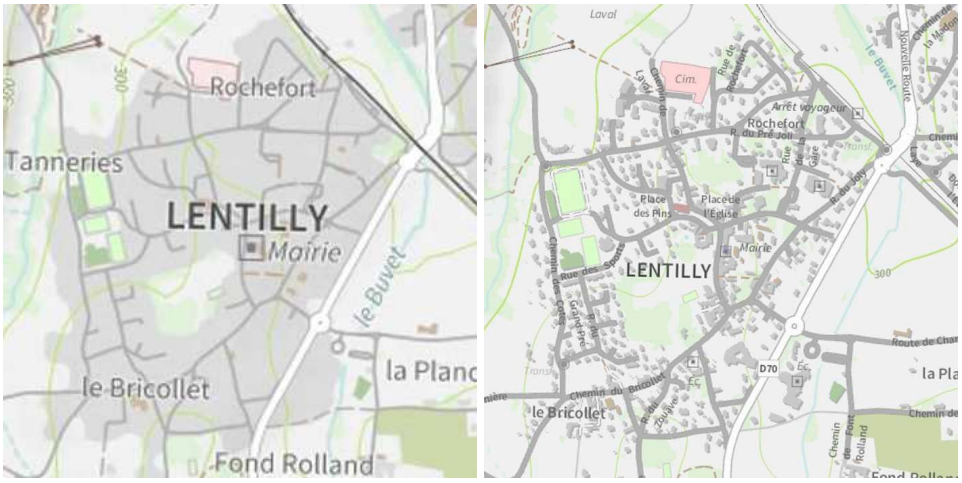

Figure 1.1 Deux cartes du même lieu à différentes échelles (petite échelle à gauche, grande échelle à droite). Plusieurs mécanismes de généralisation sont visibles, dont le remplacement des bâtiments par des tâches urbaines et le déplacement ou la suppression de certains labels. Cartes via IGN et Esri France.

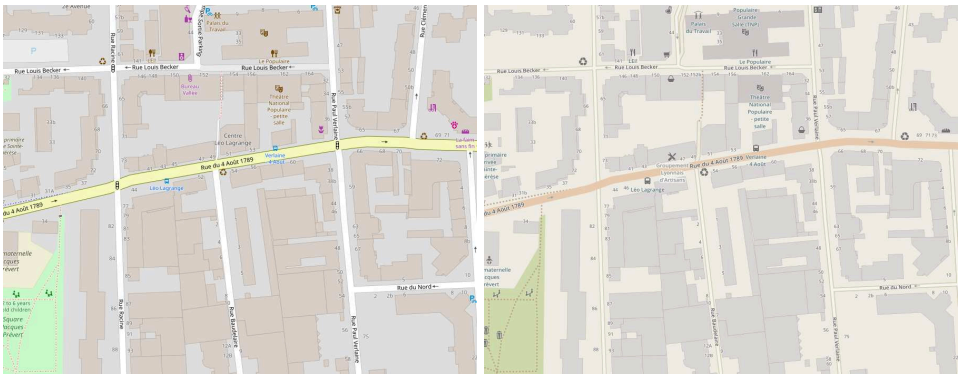

Figure 1.2 Deux cartes du même lieu représenté avec différentes symbologies : lignes de différentes épaisseurs, changements de couleurs et d'icônes. Cartes via Open Street Map.

selon les désirs de l'utilisateur. Ces couches sont habituellement thématiques. On peut, par exemple, avoir une couche pour les bâtiments, une pour la végétation, une pour le fond de carte, etc. L'aspect superposition et mélange de couches est emblématique de l'informatique géographique, où des données de différentes sources sont utilisées dans la même application. À noter l'importance des standards pour permettre l'interopérabilité entre l'application de visualisation et les sources de données.

Îlot urbain Un îlot urbain (plus communément appelé pâté de maisons), est un ensemble de parcelles de terrain délimité par des rues. C'est une structure qui est souvent utilisée comme plus petite subdivision de la ville, en dessous des quartiers et des arrondissements.

1.2.2 Système de projection

En informatique géospatiale, les objets que nous manipulons ont une existence et une position sur la surface de la Terre réelles. La position de ces objets dans nos applications doit donc être cohérente avec celle de la réalité et il faut avoir des modèles mathématiques de la Terre dans lesquels placer les objets. Plusieurs modèles existent pour représenter géométriquement la Terre [Bur08]. Les modèles les plus intuitifs sont ceux qui représentent la Terre d'une manière similaire à ce qu'elle est en réalité, c'est-à-dire comme une ellipsoïde, et où la position des objets est exprimée en mètre, relativement au centre de l'ellipsoïde. On parle dans ce cas de système géodésique, dont le plus connu est le WGS84, notamment utilisé par le système GPS (fig. 1.3).

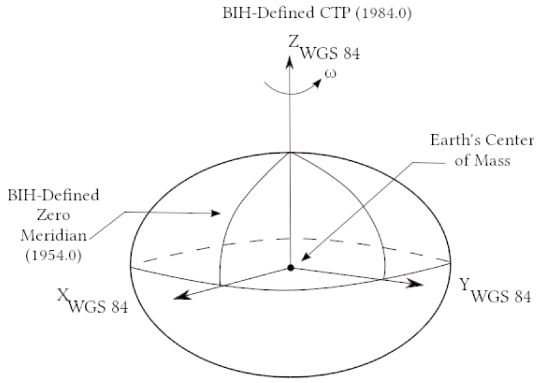

Figure 1.3 La définition du système géodésique WGS84. Crédit images : United States National Geospatial-Intelligence Agency.

Il n'est cependant pas toujours pratique d'utiliser un tel système, par exemple lorsqu'on travail à petite échelle et que le fait que la Terre ait une courbure n'est pas perceptible ou lorsqu'on souhaite visualiser des données en deux dimensions. C'est là que les projections cartographiques trouvent leur intérêt. Une projection cartographique est un ensemble d'opérations mathématiques qui permet de transformer les objets de la surface de l'ellipsoïde terrestre vers une surface plane.

Il existe une multitude de projections différentes, chacune ayant ses propriétés intéressantes, telles que la conservation locale des angles, des distances ou des surfaces. Les plus communes sont basées sur des projections cylindriques ou coniques (fig. 1.4) : les projections de Mercator, de Peters, équirectangulaire (cylindriques) et les projections coniques conformes de Lambert.

Que l'on travaille dans un système géodésique ou dans un système projeté, la taille des coordonnées rencontrées reste très grande, dépassant le million de mètres. Pour avoir une précision correcte, ces coordonnées doivent être encodées dans des nombres flottants à double précision, ce qui rajoute des difficultés : on est confronté

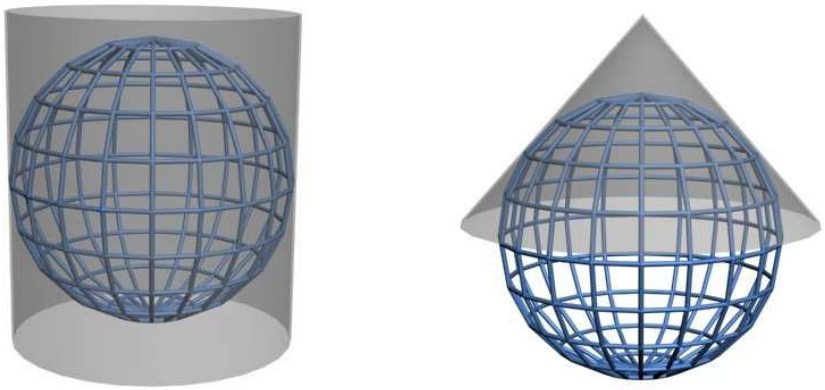

Figure 1.4 Les projections cylindrique (gauche) et conique (droite). L'ellipsoïde est projetée sur la surface grise. Crédit images : wikipedia, Traroth.

à des objets plus lourds en mémoire et compliqués à traiter au niveau de la carte graphique, qui ne sait gérer que des nombres flottants à simple précision.

1.3 Du réel à l'écran

De nombreuses étapes sont nécessaires à la production puis la consommation de données géospatiales (fig. 1.5). Ces étapes seront décrites en profondeur dans la suite de cette section. Tout d'abord, le monde réel est numérisé lors de l'étape d'acquisition. Les données numériques récupérées ainsi subissent alors un ensemble de traitements pour les rendre exploitables pour certains usages ou pour les enrichir. Les données géographiques sont ensuite échangées, transférées, stockées, éditeées ou visualisées selon l'application prévue.

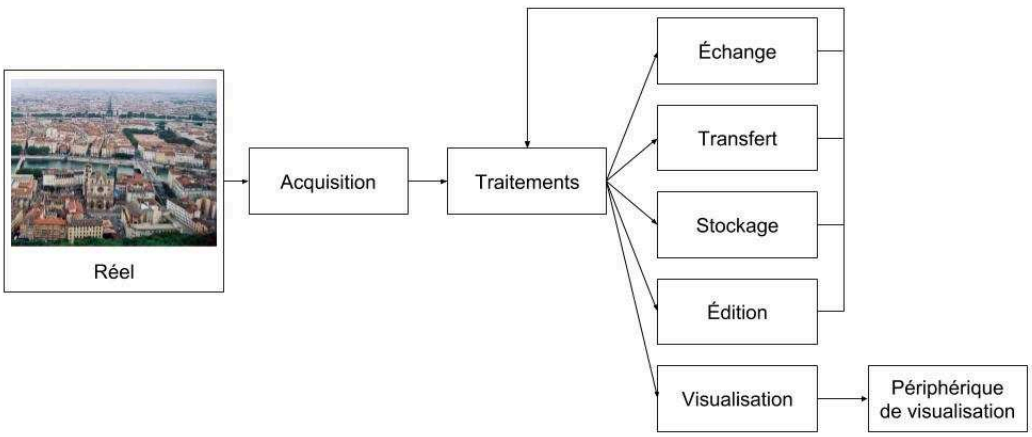

Figure 1.5 Les différentes étapes permettant l'exploitation de données géospatiales. Crédit photo : Patrick Giraud.

1.3.1 Acquisition

L'acquisition est le processus qui permet la numérisation du réel. Les outils permettant cette acquisition sont nombreux et produisent une variété de données de différentes natures. Cette section contient un rapide tour d'horizon du sujet, permettant de situer le type d'informations et le volume qui est recueilli aujourd'hui.

Commençons par la méthode la plus connue, la photographie, qui permet l'obtention d'images. Le procédé est connu, mais se décline sous beaucoup de formes : photographie aérienne, satellite ou terrestre, captant le spectre de la lumière visible, infrarouge... L'image obtenue n'a pas le même niveau de détail ou ne représente pas les mêmes données en fonction de la manière dont elle a été prise. Évoquons également l'ubiquité de ce moyen de captation, qui est à la disposition d'une majorité de la population et qui permet donc un *crowdsourcing* des données en plus des campagnes d'acquisition effectuées par les collectivités ou les entreprises.

Le LiDAR (*light detection and ranging*, ou *détection et estimation de la distance par la lumière* en français) est un procédé permettant la mesure à partir d'un laser. Les données LiDAR se présentent sous la forme de nuages de points. Cette technique s'est beaucoup développée ses dernières années en réduisant les coûts d'acquisition et en augmentant la quantité de points captés. Tout comme la photographie, le LiDAR est utilisé de différentes manières : il peut être embarqué par des drones, des avions, des voitures, ou bien effectuer ses relevés fixe.

L'acquisition ne se résume par à mesurer la couleur et la forme des objets. On trouve des dispositifs pour déterminer sa position sur la planète, notamment le GPS, mais d'autre types de mesures grâce aux capteurs. Que ce soit des informations de température, de pollution, d'humidité, ou de n'importe quelle autre grandeur mesurable et localisable, il est possible de les intégrer en tant qu'objets d'analyse dans un contexte géospatial. Finalement, on peut aussi citer les documents et données liées à des objets géographiques qui n'ont pas d'existence physique sur le terrain, mais qui les décrivent. Entrent dans cette catégorie, les cadastres et autres divisions administratives, ainsi que tout document lié à un espace en particulier.

Avec l'augmentation de l'accessibilité des appareils d'acquisition et les progrès techniques réalisés sur ceux-ci, l'acquisition résulte en la création de jeux de données extrêmement volumineux. Un jeu de données peut aisément peser plusieurs gigaoctets, voire téraoctets. Par exemple, le projet Stereopolis de numérisation mobile de l'IGN [Pap+14] collecte entre 3 et 4 terraoctets de données durant une campagne d'acquisition de 12 heures. La gestion de cette forte volumétrie est un défi majeur des recherches dans le domaine du géospatial.

Qualité des données Quelque soit la méthode d'acquisition ou le modèle de données utilisé, il y aura une erreur entre la donnée numérisée et la réalité. Cependant, la grandeur de ces erreurs varie fortement avec le procédé d'acquisition. Par exemple, la Métropole de Lyon garantit une précision au mètre pour les bâtiments[1], tandis que le projet Stereopolis atteint des précisions centimétriques [Pap+14].

Outre les erreurs de mesures, des erreurs humaines peuvent être introduites (lors de la collecte d'information par *crowdsourcing* notamment), de même que certains traitements algorithmiques peuvent amplifier les erreurs pré-existantes.

1.3.2 Traitement des données

Les données brutes acquises sont rarement exploitées en l'état. Elles vont généralement subir un ensemble de traitements avant d'être consommées, qu'on peut classer en différents types selon leur objectif. Ces traitements peuvent également survenir après d'autres étapes que l'acquisition. Par exemple, après que les données ont été transférées à un client web, celui-ci peut les modifier pour les adapter à un certain style de visualisation.

La première classe de traitement est celle des traitements post-acquisition qui transforment les données brutes acquises en données géolocalisées exploitables. Parmi elles, on trouve :

- les opérations de recalage qui consistent en la mise en correspondance de différentes acquisitions ;
- la géolocalisation : placer les données acquises dans un référentiel géographique, à l'aide de relevés GPS par exemple.

La deuxième classe de traitement est constituée de traitements dont le but est d'extraire des informations ou de transformer la donnée. Pour qualifier ces traitements, je parlerai dans ce manuscrit de traitement d'exploitation car ce sont ces traitements qui extraient la valeur des données. C'est la catégorie la plus variée et qui comporte le plus de traitements différents, dont voici un aperçu :

- la reconstitution d'objets 3D à partir d'un ensemble d'images (photogrammétrie) ;
- la construction de maillages à partir d'un nuage de points ;
- l'anonymisation des données, comme par exemple le floutage des visages ou des plaques d'immatriculation ;
- la généralisation ;

[1] https://download.data.grandlyon.com/files/grandlyon/localisation/bati3d/Maquette_3D.pdf

- la symbolisation ;
- les simulations physiques : propagation du son, ensoleillement, inondation, etc. ;
- le routage, c'est-à-dire le calcul d'itinéraire.

Finalement, la dernière classe concerne les traitements dont l'objectif est de transformer les données pour l'optimisation de certaines tâches. Ces traitements d'optimisation ne changent pas les données (aux erreurs induites par une éventuelle compression près), mais change la manière dont elles sont formatées ou organisées. On y trouve :

- l'indexation spatiale, pour permettre de retrouver plus rapidement les objets en fonction de leur position ;
- le transcodage des données dans une variété de formats optimisés pour un usage particulier (stockage, échange, affichage, etc.).

1.3.3 Échange et transfert

Dans le cadre d'une application web, les données seront dans la majorité du temps distantes, c'est à dire qu'elles sont stockées sur une (ou plusieurs) machine différente de celle de l'application. Il faut donc accorder une attention particulière à l'échange et au transfert des données. On parle d'échange pour le partage de données avec d'autres personnes. Le transfert fait référence à l'action spécifique de l'envoi des données d'une machine à l'autre. Deux problématiques se dégagent : la performance du transfert des données et l'interopérabilité de l'échange.

Afin d'améliorer la performance du transfert, il faut s'intéresser à la compression des données, mais aussi à la vitesse de compression et de décompression. Il est aussi pertinent d'étudier la progressivité du transfert, c'est-à-dire transférer les données de telles sortes qu'elles soient exploitables au fur et à mesure qu'elles parviennent à l'application.

L'interopérabilité désigne la capacité d'un système à interagir avec d'autres systèmes grâce à la connaissance de leurs interfaces respectives. En garantissant l'interopérabilité d'un système, il devient possible de diffuser beaucoup plus largement les données produites, et inversement, de disposer de beaucoup plus de données à intégrer à son application. Une manière de garantir cette interopérabilité est de conformer les protocoles et formats de données de son application à des normes ou des standards. En partie 2.4.1 des protocoles et formats standardisés pertinents à cette thèse seront présentés.

Un témoin de l'utilité des standards est la multiplication des ressources en *open data* : des données (statistiques, géométries géolocalisées, etc.) diffusées gratuitement par certaines communautés (principalement des villes). Les communautés y trouvent leur compte grâce aux applications développées par des tiers qui utilisent les données qu'elles mettent à disposition, applications qui peuvent être valorisées par la ville.

La totalité des données utilisées dans ma thèse sont des données *open data*. Parmi les villes dont les données ont été utilisées, on peut notamment citer les villes de Lyon[2], Montréal[3] ou Helsinki[4].

1.3.4 Stockage

Le stockage des données géospatiales peut prendre plusieurs formes selon l'usage qui sera fait des données par la suite. La manière dont les données sont stockées est cruciale lorsque l'on réfléchit aux moyens de les mettre à disposition.

Si l'objectif est la distribution, il est pertinent de stocker les données sous la forme de fichiers qui contiennent le maximum d'informations, même s'ils sont volumineux et complexes. Le format CityGML (voir partie 2.4.1) a été développé dans ce sens.

Dans le cas où l'interaction avec les données est importante, l'utilisation de bases de données, et particulièrement de bases de données géospatiales, offre la possibilité de manipuler facilement les données stockées.

Enfin, pour optimiser les performances, il peut être pertinent de créer un cache, c'est-à-dire de générer à l'avance le résultats de requêtes pour pouvoir les servir le plus rapidement possible.

1.3.5 Édition

Certains cas d'utilisation, comme la mise en place d'un plan d'urbanisme, nécessitent de pouvoir éditer les données. Cette édition prend plusieurs formes : l'ajout et la suppression d'objets géographiques, la modification de ces objets ou encore la modification des informations sémantiques qui leurs sont liées.

Dans la problématique d'édition, le temps nécessaire à ce que les modifications de l'utilisateur se répercutent sur les données stockées peut avoir son importance. Par exemple, le temps est un facteur crucial dans les applications d'édition collaborative

[2] https://data.grandlyon.com/
[3] http://donnees.ville.montreal.qc.ca/
[4] http://www.hri.fi/en/

ou qui incorporent des données temps réel (comme la position des transports en commun) mais il l'est beaucoup moins si l'objectif est simplement de mettre à jour les données après une nouvelle campagne d'acquisition.

1.3.6 Visualisation

Les applications de cartographie en ligne (*Web mapping*) ont montré l'adéquation du format web pour la mise à disposition d'informations géographiques. Google Maps[5], OpenStreetMap[6] ou les géoportails nationaux[7] en sont le témoin. Ainsi, il paraît intéressant de mettre à disposition les données géospatiales 3D à travers le même support.

La visualisation 3D sur le Web est devenue particulièrement accessible grâce à l'interface de programmation (API) WebGL, créée par le Khronos Group qui permet au navigateur d'accéder nativement à la carte graphique. Elle est basée sur OpenGL ES, une simplification d'OpenGL prévue initialement pour les systèmes embarqués. Malgré de grandes améliorations, le code JavaScript exécuté dans le navigateur a des performances médiocres (voir partie 2.3). Ainsi, lors de la conception de l'application, il faut s'assurer de minimiser les calculs effectués par le client web.

Des spécificités du domaine géospatial engendrent des besoins particuliers pour la visualisation des données. Par exemple, les données géospatiales sont d'échelles très hétérogènes, il faut donc être capable de gérer cet aspect multi-échelle lors de la visualisation, à travers la gestion de la généralisation par exemple. Un autre exemple est l'application d'une symbologie aux objets géographiques pour représenter l'information.

La visualisation n'est pas la seule finalité des applications géospatiales 3D. L'interaction avec les données a également son importance. Selon l'usage, il faudra alors permettre soit des interactions très simples comme pouvoir sélectionner des objets et obtenir les informations sémantiques liées, soit des interactions beaucoup plus complexes comme l'édition ou l'analyse des données. Il est difficile, voire impossible, de concilier visualisation optimisée et grand degré d'interaction. En effet, certaines interactions requièrent un formatage particulier qui n'est pas optimal pour le rendu.

[5] https://www.google.fr/maps
[6] https://www.openstreetmap.org/
[7] https://www.geoportail.gouv.fr/

1.4 Problématique et positionnement

La description du processus de production et de visualisation de données géospatiales 3D permet d'introduire le contexte de ma thèse. Mes travaux se concentrent sur une partie plus précise de ce processus (fig. 1.6). Les données initiales que je traite ont déjà été numérisées et transformées dans des formats facilement exploitables, notamment le format CityGML. Mes recherches se focalisent sur le transfert et la visualisation des données, ainsi que les traitements sur les données permettant de les organiser ou de les transformer pour améliorer l'expérience de l'utilisateur. Les notions de généralisation et de symbologie sont abordées, mais ne constituent pas le cœur de ma thèse.

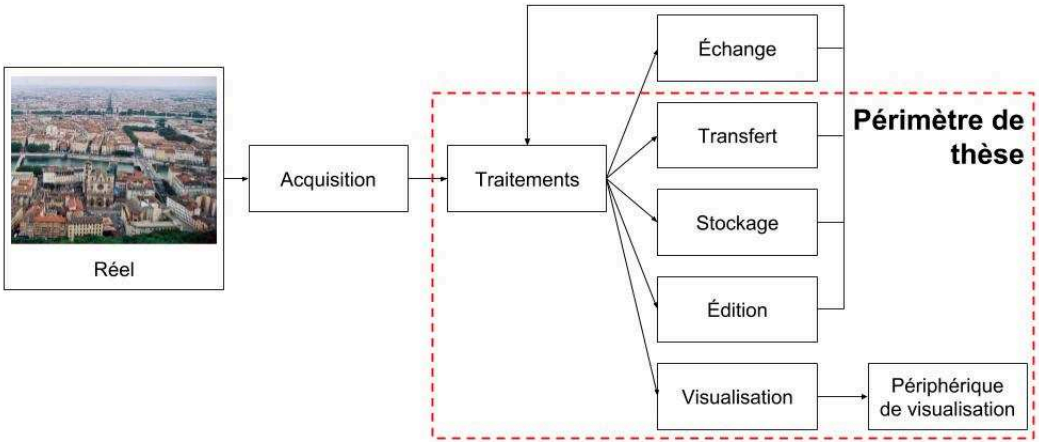

Figure 1.6 Le positionnement de ma thèse représenté dans le schéma de la fig. 1.5.

Les lecteurs de cette thèse seront probablement familiers avec des applications telles que Google Maps — déjà évoqué précédemment — (fig. 1.7), voire Cesium[8] (fig. 1.8). Cette thèse n'a pas vocation à concurrencer les travaux de Google et d'AGI respectivement. Mes contributions s'inscrivent dans une démarche différente de ces produits.

Contrairement à Google, mon approche n'est pas purement visuelle. Il doit être possible d'interagir avec les données, que tous les objets soient des entités séparées. Cela n'est pas le cas dans l'application de Google, où les objets et le terrain sont fusionnés dans un grand maillage progressif. De plus, Google se soucie peu d'interopérabilité, et propose des données formatées pour sa seule application.

AGI propose une solution plus proche de nos intérêts. Chaque objet peut être sélectionné, des informations sémantiques peuvent lui être liées. Cependant, les données qu'AGI utilisent sont statiques : il s'agit de fichiers stockés dans le système de fichier sur lesquels il est peu efficace de faire de l'analyse ou des modifications.

[8] https://cesiumjs.org/

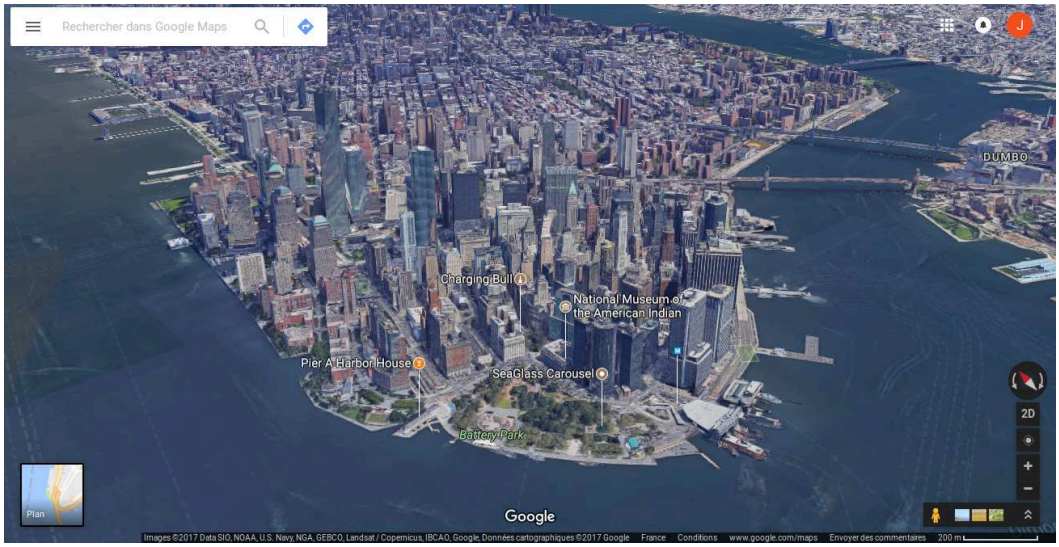

Figure 1.7 Une vue 3D de New York dans Google Maps.

Figure 1.8 Une vue 3D de New York dans Cesium.

Dans cette thèse, nous souhaitons tirer parti de la puissance des bases de données géospatiales, profiter de leur capacité d'édition et d'analyse. C'est un choix qui offre des performances sous-optimales pour visualiser des données, mais qui permet en échange de gagner une flexibilité bien plus grande dans l'utilisation et l'interaction avec les données.

Les problématiques notables de ma thèse sont :

- la prise en compte de l'aspect multi-échelle de la ville ;
- l'adaptation de la représentation de la ville à l'utilisateur ;
- la possibilité de modifier la ville et de changer sa représentation de manière interactive.

Pour répondre à ces problématiques, plusieurs verrous sont à surmonter :

- la volumétrie : on manipule des données très lourdes qui doivent transiter sur le réseau ;
- l'édition et l'interrogation : il faut être capable d'exploiter, voire d'éditer, les données géométriques et sémantiques en plus de les visualiser ;
- les représentations : un objet peut être représenté de manières différentes, que ce soit pour obtenir une vue métier ou différents niveaux de détails ;
- l'hétérogénéité : les données de la ville se présentent sous de multiples formes (maillages, nuages de points, informations sémantiques...) qu'il faut être capable d'exploiter au sein de la même application.

Dans la suite de ce manuscrit, je présenterai les méthodes développées pendant ma thèse qui répondent aux problématiques introduites précédemment. Nous verrons des méthodes qui permettent de visualiser de grandes quantités de données, tout en permettant d'éditer ou de personnaliser la représentation de la ville.

1.5 Plan de la thèse

L'organisation de ma thèse est la suivante. Après cette introduction, je présenterai un état de l'art des travaux des domaines traitées par ma thèse, le rendu d'images 3D, le Web et l'informatique géographique. Ensuite, j'aborderai le socle technique sur lequel sont construites les méthodes développées durant ma thèse. Les architectures définies et les logiciels utilisés ou conçus sont décrits dans ce chapitre. Le chapitre suivant est consacré à la structuration de données géographiques 3D dans le but de visualiser et d'interagir avec un large volume de celles-ci. Après cela vient un chapitre dédié à la personnalisation de la visualisation de données géographiques. Enfin, une synthèse et des perspectives sont proposées dans le chapitre de conclusion.

2 État de l'art

Sommaire

Dans l'introduction, nous avons vu que l'informatique géographique et les problématiques de ma thèse se situaient au carrefour de plusieurs domaines de recherches. Ces différents domaines sont abordés dans cet état de l'art.

Dans ce chapitre, nous commencerons par étudier les différents modèles de données rencontrés en informatique géographique afin d'avoir un aperçu des types d'objets à gérer.

Ensuite, nous étudierons les méthodes de visualisations de scènes complexes, c'est-à-dire des scènes dont les données sont volumineuses et le nombre d'objets est important. Je présenterai le pipeline usuel de rendu, ainsi que les manières de structurer et de transformer les données pour permettre une visualisation fluide.

Nous verrons alors quelles sont les problématiques du Web, et plus précisément de la 3D sur le Web, induites par la puissance limitée disponible et les données déportées sur des serveurs distants.

Enfin, les aspects spécifiques à l'informatique géospatiales seront abordés. Nous verrons les standards qui ont été définis pour échanger la donnée, les méthodes de généralisation et les applications qui ont été développées autour des modèles numériques de la ville.

2.1 Modèles de données

En information géographique, la nature des objets modélisés est variée : bâtiments, terrains, routes, végétations, mesures de vent, noms de lieux, etc. Cette variété ce reflète dans la nature de leurs modélisations, modélisations qui peuvent être contraintes par la méthode d'acquisition des données ou la manière dont on souhaite les représenter. Par exemple, un terrain mesuré par laser ou reconstitué à partir d'un ensemble de photos n'utilisera pas le même modèle ; une route pourra être représenté par une simple ligne pour une application de recherche d'itinéraire, mais nécessitera un modèle surfacique pour une application d'entretien de la voirie (fig. 2.1). La symbologie appliquée à ces objets influe également sur leur modélisation. Dans cette partie, je présenterai les principaux modèles de données utilisés en informatique géographique.

Il est important de savoir quels modèles existent car les méthodes conçues seront différentes selon qu'on choisisse un modèle à traiter plutôt qu'un autre, ou qu'on ait une volonté de généricité.

Ces modèles ne sont pas inédits en informatique graphique. Une nuance par rapport aux autres domaines est que les modèles sont géolocalisées, c'est-à-dire qu'ils

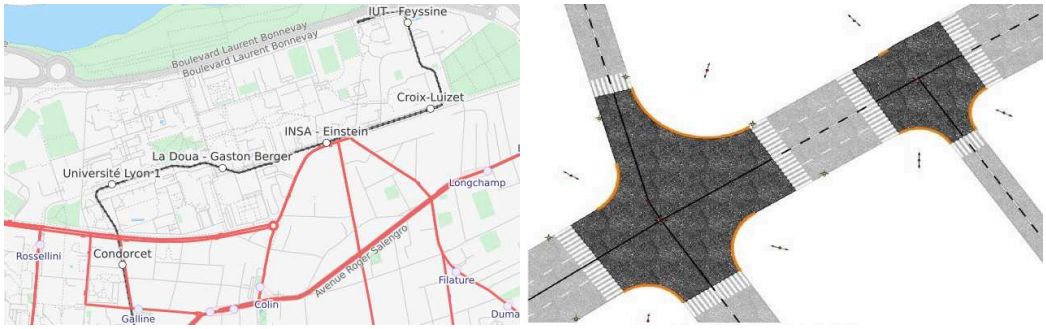

Figure 2.1 Les routes peuvent être représentées à l'aide de lignes (gauche) ou de modèles plus complexes, comme des surfaces (droite). Images extraites respectivement d'Open Street Map et de [Cur16].

ont une position définie et réelle sur la surface de la Terre. L'impact principal de cette géolocalisation est la taille des coordonnées des objets géographiques, dont l'ordre de grandeur est (selon le référentiel) la centaine de milliers ou le million de mètres.

Images En informatique graphique, une image est une matrice en deux dimensions où à chaque élément est associé une valeur, le plus souvent une couleur mais parfois un nombre entier ou flottant. L'image est utilisée pour les fonds de cartes (orthophotographies, plans), les cartes de hauteurs ou pour représenter des mesures spatiales diverses (comme le bruit ou la pollution). À chaque pixel de l'image correspond une position géographique.

Maillages Un maillage est un ensemble de points, arêtes et faces représentant la surface ou le volume d'un objet 3D et sa topologie. Les faces sont le plus souvent des triangles. Les maillages sont utilisés pour modéliser le terrain, les bâtiments, et d'autres objets 3D de la ville.

« Soupes de triangles » Une « soupe de triangles » est un ensemble désordonné de triangles. Il s'agit en quelque sorte d'un maillage sans topologie. On les retrouve essentiellement pour modéliser des terrains.

Nuages de points Un nuage de points est ensemble de points 3D auxquels sont associés diverses informations, typiquement une couleur. La quantité de données disponibles sous la forme de nuages de points est en forte croissante grâce à la baisse des coûts d'acquisitions. Les nuages de points ne sont généralement pas segmentés : les différents objets représentés (bâtiments, terrain, arbres, ...) ne sont pas individualisés.

Données vectorielles Les données vectorielles sont des points, des lignes ou des polygones en deux ou trois dimensions. Ces données sont utilisées pour représenter des routes, des limites administratives ou des flux par exemple.

Information sémantique liée Les objets géographiques n'ont pas qu'une composante géométrique. Ils possèdent également un ensemble d'informations sémantiques les décrivant. Ces informations sont multiples, il peut s'agir d'attributs simples, comme un nom, une adresse, une classe, ou bien plus complexes, comme la description de la décomposition d'un bâtiments en plusieurs pièces. Rendre ces informations accessibles est particulièrement important pour permettre une exploitation efficace des données géographiques.

2.2 Visualisation de scènes complexes

Une ville est composée de milliers d'objets et les jeux de données qui en sont issus peuvent aisément dépasser les centaines de gigaoctets. Il est clair qu'avec de telles volumétries, il n'est pas envisageable d'afficher l'ensemble de ces données telles quelles. Il existe des outils dans la littérature scientifique qui offrent des pistes pour la gestion de telles jeux de données. Après avoir présenté la méthode usuelle de rendu temps réel, nous verrons quels sont ces outils.

2.2.1 Rendu temps réel

Le rendu temps réel est le processus de génération d'images d'une scène à une fréquence suffisamment grande pour que l'observateur ne distingue pas les images individuelles. Le plus souvent, cette fréquence est de 60 images par secondes mais, selon le besoin de fluidité ou de réactivité de l'application, elle peut se trouver entre 15 et 120 ou plus images par secondes. Dans cette section, je présente le pipeline usuel de rendu temps réel (fig. 2.2), tel que décris par [AM+08].

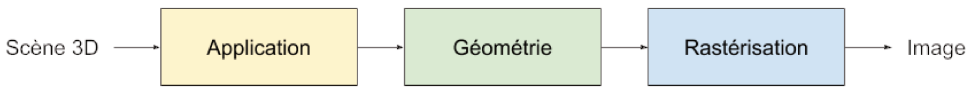

Figure 2.2 Les phases principales du pipeline de rendu tel que décrit par [Cla76]. La phase d'application sélectionne les primitives à afficher, la phase de géométrie calcul l'effet de la lumière sur les primitives et les projette dans l'espace écran et la phase de rastérisation détermine la couleur de chaque pixel.

Une scène 3D est composée d'un ensemble d'objets, constitués de primitives (points, lignes ou triangles : la géométrie de l'objet) et de matériaux (l'apparence de l'objet). À chaque *vertex* (sommet) d'une primitive sont associés des attributs :

position, normale, coordonnées de texture, couleur... Ces objets sont définis dans leur propre système de coordonnées, leur espace d'origine. Des matrices de transformations permettent de transformer ces objets de leur espace d'origine vers l'espace monde. La scène a également une caméra, dont le champ de vision est matérialisé par un tronc de pyramide (fig. 2.3) et des sources de lumières.

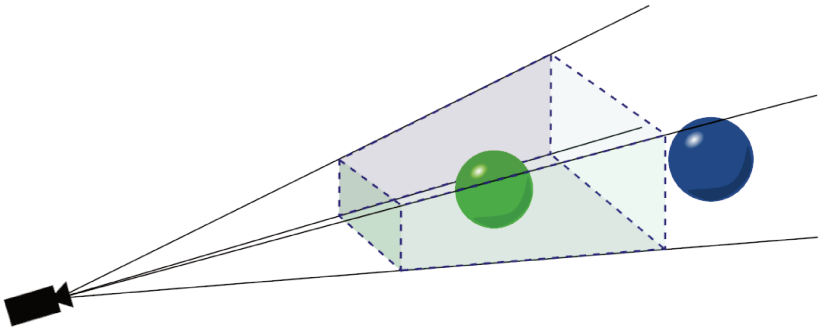

Figure 2.3 Le champ de vision d'une caméra pour une vue en perspective : un tronc de pyramide. La sphère verte est visible mais pas la sphère bleue. Image originale via wikipédia, Tetromino.

Le pipeline de rendu a trois phases : la phase *application*, la phase *géométrie* et la phase *rastérisation*.

La phase application comporte toutes les opérations spécifiques à l'application, telles que la simulation physique, le déplacement de la caméra ou la gestion des interactions, et est par conséquent la moins définie du pipeline. Le seul impératif est qu'à la fin de cette phase, l'application transmette l'ensemble des primitives qui devront être rendues. C'est une phase où des algorithmes d'accélération peuvent être implémentés pour réduire la charge du reste du pipeline : c'est précisément ce qui nous intéresse dans le cadre de cette thèse

La phase *géométrie* réalise la majorité des opérations sur les primitives et leurs vertices. Elle transforme les vertices de l'espace d'origine des objets vers l'espace caméra, détermine l'effet de la lumière sur chaque vertex (les données de *shading*) à partir de ses attributs et matériaux, projette les vertices dans l'espace écran et découpe les primitives qui débordent en dehors de l'écran. Ces opérations sont effectuées par la carte graphique, soit via des circuits électroniques dédiés, soit via des *shaders*, des programmes qui peuvent être exécutés par le processeur graphique.

La phase *rastérisation* détermine la couleur des pixels de l'image à partir des vertices transmis par la phase précédente. Lors de cette phase, les données de *shading* des vertices sont interpolées sur les pixels le long des primitives et les couleurs résultantes sont calculées. C'est aussi lors de cette phase que la visibilité de chaque pixel est résolue : à une même coordonnée sur l'écran, plusieurs pixels ont pu être générés (les projections de plusieurs primitives se trouvent à cet endroit), il

faut donc déterminer quel pixel est devant, ou mélanger les couleurs si la primitive au premier plan est partiellement transparente. Comme la phase précédente, celle-ci est effectuée sur la carte graphique.

Méthodes d'accélération Le moyen le plus simple pour accélérer le rendu est de limiter le nombre d'objets à traiter et le nombre de primitives envoyées à la carte graphique.

Les opérations de *culling* (élimination) sont une optimisation très commune dans les moteurs de rendu. Le but est de retirer les objets qui ne sont pas dans le champ de vision de la caméra des objets à dessiner, afin de ne pas envoyer de primitives superflues à la carte graphique. Une autre approche est de réduire le nombre de primitives par objet, par exemple en simplifiant leur géométrie quand les détails ne sont plus visibles. Ces deux méthodes seront étudiées dans la suite de cette partie.

Lorsqu'une scène dispose de nombreux objets, des problèmes de performances peuvent survenir, même si le nombre de primitives à afficher est modeste. En effet, chaque objet (et chacun des matériaux de l'objet) nécessite l'envoi d'une commande (*draw call*) à la carte graphique qui est très consommatrice de ressources. Réduire le nombre d'objets, et donc de commandes à envoyer, est nécessaire pour éviter ce goulet d'étranglement. Pour cela, on peut avoir recours au *batching*, c'est-à-dire fusionner plusieurs objets en un seul. Si cette fusion est triviale pour les informations géométriques (il suffit de concaténer des tableaux), elle implique de fusionner les textures des objets originaux, ce qui est plus complexe. Cependant, des algorithmes robustes ont été développées pour ce problème [Lév+02]. Ce procédé de batching est particulièrement utile en informatique géographique où les objets géographiques sont très nombreux. L'inconvénient de ce procédé est qu'une géométrie correspond alors à plusieurs objets. Pour que le programme puisse différencier les différents objets de la scène, il faut ajouter un attribut aux vertices des objets contenant un identifiant de l'objet.

2.2.2 Structures de données

Lors de l'exploration d'une scène 3D massive, l'ensemble des données n'est souvent que partiellement visible à l'écran. Ne pas dessiner les objets qui n'apparaissent pas à l'écran permet d'optimiser le temps de rendu sans aucune contre-partie visuelle. Cette opération de *culling*, évoquée dans la section précédente, est une optimisation qui est apparue très tôt dans la littérature scientifique, et qui a mené à la conception de nombreuses structures de données spatiales différentes pour rendre cette opération la plus efficace possible.

Une approche basique est la subdivision de l'espace en une grille régulière. Les géométries sont réparties entre les différentes cellules de la grille en fonction de leur position. Cette structure simple est parfois suffisante, lorsque l'on sait que seule une partie localisée de la scène sera visible à la fois, mais pose problème dans d'autres cas, par exemple lors d'une vue d'ensemble de la scène, où toutes les cellules de la grille sont visibles : le volume de données à gérer est alors trop grand.

Ce genre de structure rencontrant rapidement ses limites, la communauté 3D s'est tourné vers les structures hiérarchiques, notamment avec la conception d'un outil encore utilisé dans la vaste majorité des applications 3D : le graphe de scène [Cla76], qui permet — entre autre — de déterminer très rapidement quels objets sont visibles. Le graphe de scène se présente sous la forme d'un arbre au sein duquel se trouve l'ensemble des objets de la scène, répartis dans les nœuds de l'arbre . Selon l'application, les objets se trouvent soit uniquement dans les feuilles de l'arbre, soit dans n'importe quel nœud. Il existe de nombreuses méthodes pour répartir les objets dans les nœuds que nous verrons dans la suite de cette section. Dans tous les cas, la structure est construite de telle sorte que si un nœud n'est pas visible, alors tous ses fils sont également non-visibles. Traditionnellement, une boîte englobant l'ensemble des objets du nœud et de ses fils est définie et liée au nœud afin de tester sa visibilité rapidement (fig. 2.4). On appelle alors cette structure une hiérarchie de volumes englobante, ou *bounding volume hierarchy* (*BVH*).

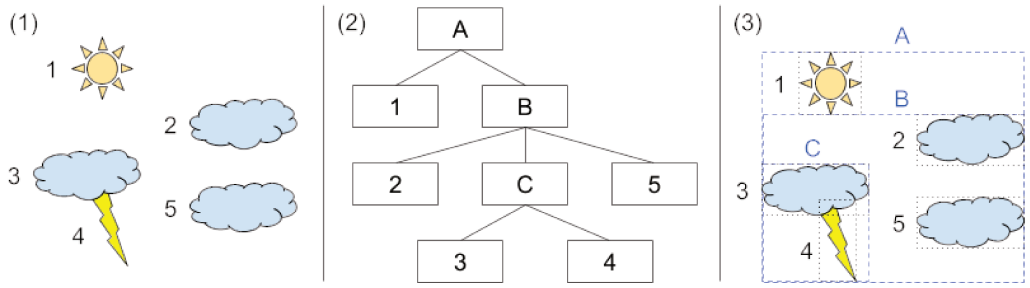

Figure 2.4 Organisation d'une scène simple (1) dans un graphe de scène (2). Le graphe de scène représentée ici est arbitraire, il est possible d'organiser les objets différemment dans l'arbre. En associant un volume englobant à chaque objet (pointillés noirs) ou nœud (pointillés bleus) du graphe, l'arbre de scène fait également office de hiérarchie de volumes englobants (3).

Comme évoqué plus tôt, la construction d'arbres de scènes peut se faire de multiples manières. Une solution simple est de baser la répartition des nœuds dans l'arbre sur un découpage régulier de l'espace, typiquement un octree [Mea80]. L'octree est un arbre où chaque nœud représente une portion d'espace, qui est subdivisée en parts égales par ses huit enfants 2.5. Cette structure à comme avantage sa simplicité et la possibilité de définir instantanément dans quels nœuds de l'octree se situe n'importe quel point de l'espace. Cependant, du fait de sa régularité, les scènes dont la densité d'objets n'est pas homogène formeront un arbre très déséquilibré —

c'est-à-dire que la profondeur des feuilles de l'arbre seront très variables. Comme la vitesse de certains algorithmes utilisant ces arbres dépendent de leur profondeur, cela entraîne des performances sous-optimales.

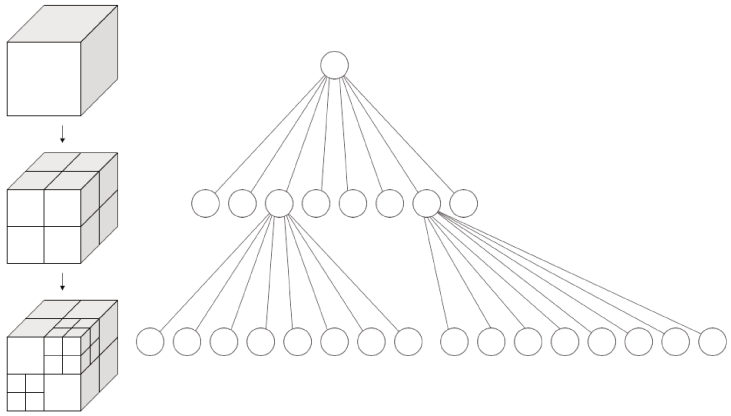

Figure 2.5 Subdivision hiérarchique de l'espace (gauche) et octree résultant (droite). Crédit image : wikipédia, WhiteTimberwolf.

D'autres structures existent pour garantir l'équilibre du graphe de scène résultant. On peut citer notamment les R-trees [Gut84] (fig. 2.6), surtout utilisés pour l'indexation spatiale dans les systèmes de gestion de bases de données. L'idée du R-tree est de minimiser la taille des boîtes englobantes de chaque niveau de l'arbre. Pour cela, lors de l'insertion d'un nouvel objet, ce dernier est insérer dans le nœud (dont le nombre d'objets est limité, 3 dans mon exemple) qui subirait l'élargissement le plus petit.

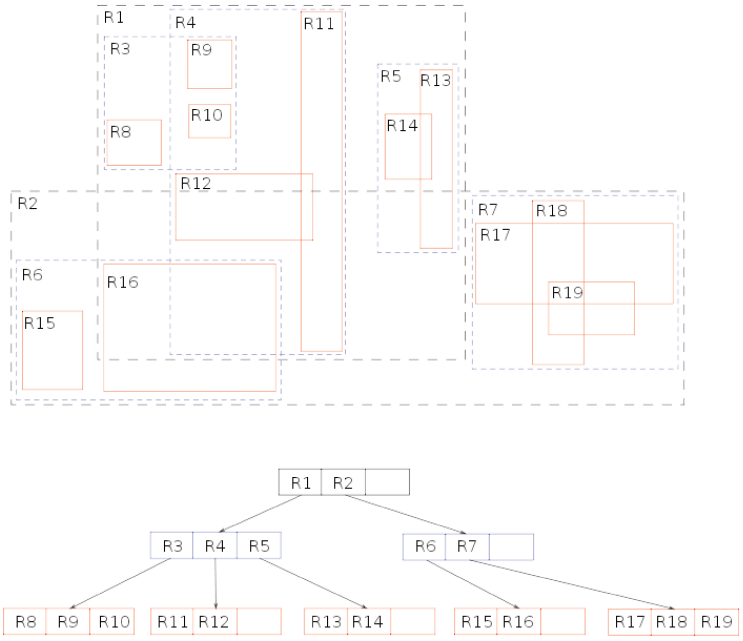

Figure 2.6 Répartition d'un ensembles d'objets (représentés par leur boîte englobante minimale en rouge) dans un R-tree. Crédit image : wikipédia, Skinkie.

Les k-d trees [Ben75] (fig. 2.7) sont aussi des arbres équilibrés : l'espace est récursivement divisé en deux parties, chacune disposant de la moitié des objets de l'espace. La séparation entre les deux espaces est un plan normal à une des directions (Ox), (Oy), (Oz), la direction choisie changeant à chaque récursion.

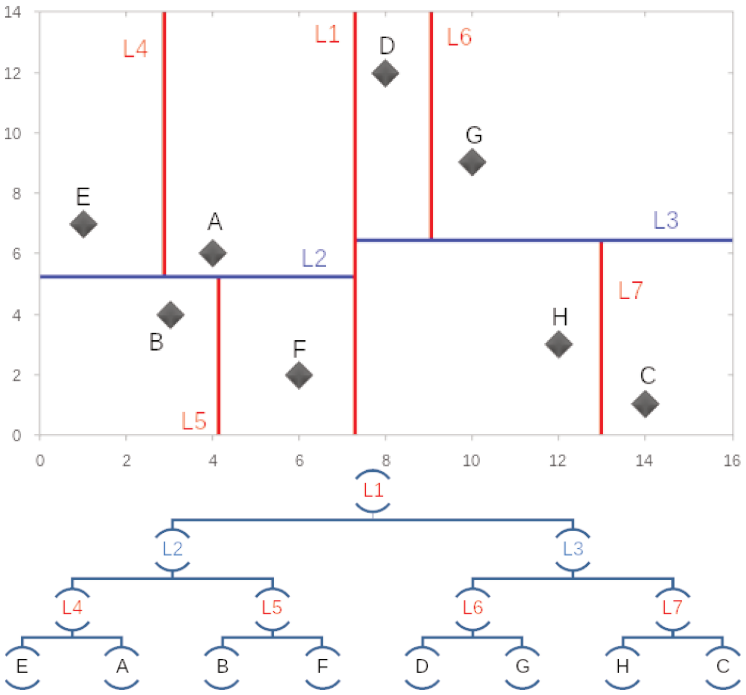

Figure 2.7 Répartition de points dans un k-d tree. Crédit image : wikipédia, Alexandre Delesse.

L'utilisation d'heuristiques pour construire des hiérarchies de volumes englobants est une autre approche pour la création d'arbre de scènes. Par exemple, dans le cadre du rendu par lancé de rayon, [Wal07] utilise une *surface area heuristic* (*SAH*) pour estimer le coût de la traversée de l'arbre et construire une BVH efficace.

2.2.3 Niveau de détail

À la place ou en complément de ces structures spatiales, il est possible d'enlever les détails superflus à la scène visualisée pour réduire le nombre de primitives à gérer. Le concept de niveau de détail (*level of detail*, ou *LoD*) désigne la simplification (ou complexification) de la géométrie d'un objet en fonction de sa distance d'affichage [Lue03] (fig.2.8). En effet, plus un objet est éloigné de la caméra, moins les détails de sa géométrie sont visibles. On peut réduire drastiquement le nombre de triangles à afficher dans une scène en ayant un impact visuel faible. La fig. 2.9 illustre ce phénomène, l'image de droite représente la différence entre le rendu avec ou sans gestion du niveau de détail. On constate une très légère différence (les pixels blanchâtres) dans le rendu en bordure des sphères.

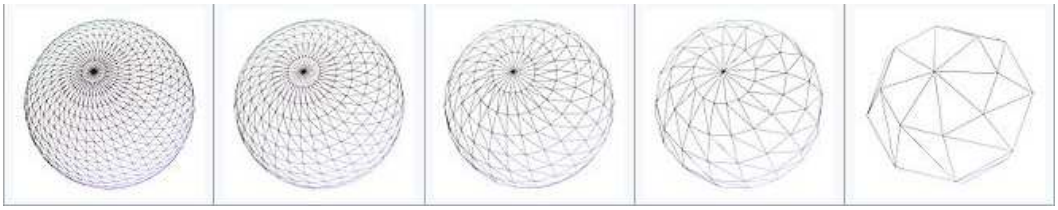

Figure 2.8 Une sphère représentées à plusieurs niveaux de détails. Crédit image : wikipedia, MaxDZ8.

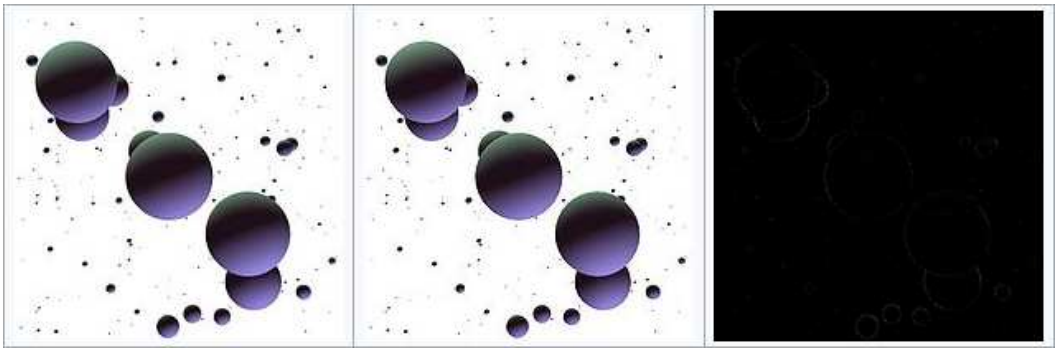

Figure 2.9 Un ensemble de sphères rendues sans gestion du niveau de détail (gauche, 2 328 480 points), avec gestion du niveau de détail (milieu, 109 440 points) et la différence entre les deux images rendues (droite). Crédit image : wikipedia, MaxDZ8.

Il existe plusieurs manières de gérer les niveaux de détails. Les trois principales sont présentées ci-après.

La première approche est le niveau de détail continu, où la géométrie est progressivement simplifiée et où il est possible de passer de la représentation la plus simple de la géométrie à la plus complexe un triangle (ou point ou autre primitive) à la fois. [Hop96] est un exemple notable de ce type de niveau de détail. Le maillage est progressivement simplifié par la fusion des sommets d'une arête. Cette opération est réversible (fission d'un sommet en deux), il suffit donc d'enregistrer les paramètres de la fusion pour pouvoir reconstruire le maillage à partir de sa version simplifiée en appliquant les opérations inverses de fission.

Ce type d'approches demande des ressources pour la reconstruction du maillage. Pour certaines applications, il est préférable d'utiliser des méthodes moins gourmandes : les niveaux de détails discrets permettent cela, en calculant *hors-ligne* (en dehors du fonctionnement régulier de l'application, potentiellement sur une autre machine) un ensemble de niveaux de détails pour la géométrie. On pourra alors afficher le niveau de détail qu'il convient sans avoir à le générer [FS93].

Ces méthodes sont adaptées à la réduction de la complexité d'une géométrie unique, mais dans certains cas, la scène à afficher comporte des milliers d'objets. Même si individuellement ces objets sont simples, des problèmes de performances se feront ressentir à cause de leur nombre. [Eri+01] présente une solution avec

l'introduction des niveaux de détails hiérarchiques, *HLoD*. Avec cette méthode, des niveaux de détails sont créés non seulement pour les objets individuels, mais aussi pour des groupes d'objets tels qu'ils sont groupés dans l'arbre de scène. Cela permet de réduire drastiquement le nombre d'objets à afficher ainsi que d'obtenir des meilleures simplifications de ces objets.

Contrôle du raffinement Pour contrôler le passage d'un niveau de détail à l'autre une solution commune est de calculer l'erreur dans l'espace écran, ou *Screen Space Error* (SSE). Elle se base sur l'erreur géométrique de l'objet considéré, qui peut se calculer comme la distance de Hausdorff entre la géométrie de la représentation simplifiée et la géométrie de la représentation originale. La distance de Hausdorff est la distance maximale entre deux ensembles (fig. 2.10).

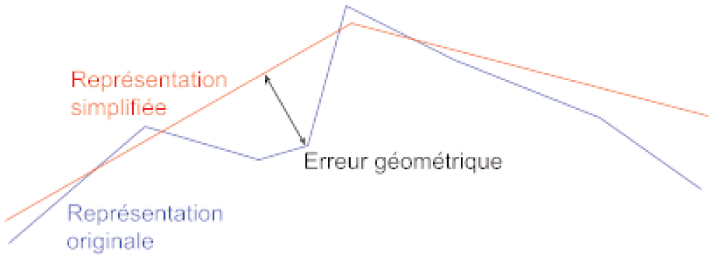

Figure 2.10 Erreur géométrique de la représentation simplifiée d'un objet (vue de coupe). La mesure de l'erreur correspond à la distance de Hausdorff entre la représentation simplifiée et la représentation originale.

La SSE découle de l'erreur géométrique, il s'agit de l'erreur perçue lorsqu'un objet est rendu à l'écran (fig. 2.11). Elle peut s'approximer avec la formule suivante :

$$SSE = \frac{erreur\_géometrique * x}{2d * tan(\theta/2)}$$

x étant la largeur de l'écran (en pixel), d la distance à l'objet et θ l'angle du champ de vision horizontal.

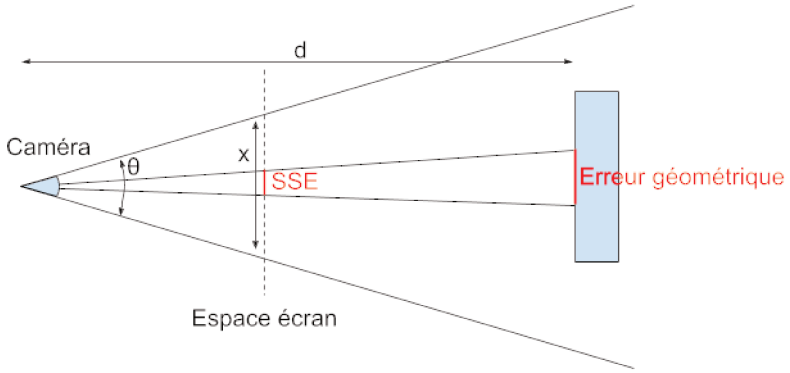

Figure 2.11 L'erreur dans l'espace écran est la projection de l'erreur dans l'espace objet vers le plan de l'écran.

Le raffinement est effectué lorsque $SSE > k$, k étant un facteur choisi à la discrétion de l'utilisateur (plus k est grand, plus on tolère de percevoir l'erreur). En d'autres termes, quand les approximations de l'objet simplifié deviennent apparentes ($k = 1$) ou trop apparentes ($k > 1$), une représentation plus précise doit être chargé.

2.3 3D sur le Web

Le Web est la technologie phare pour le partage et la diffusion de données. Autrefois limité au texte et aux images, les applications web peuvent désormais accueillir du contenu riche, tel que des vidéos ou de la 3D interactive. Ceci est encore plus probant depuis la dernière révision du langage HTML, HTML5[1], dont les spécifications sont progressivement implémentées dans les navigateurs modernes. Le standard HTML5 inclut notamment l'interface de programmation (*Application Programming Interface*, *API*) WebGL[2] qui permet d'accéder à l'accélération matérielle — en d'autres termes, à effectuer des calculs sur la carte graphique — pour effectuer des rendus 3D directement à partir du code JavaScript exécuté à l'intérieur du navigateur. WebGL est basé sur OpenGL ES, une simplification d'OpenGL destiné aux systèmes embarqués. La 3D sur le Web est donc plus limitée que sur une application de bureau[3] : une partie des fonctionnalités modernes des langages de programmation sur cartes graphiques n'est pas disponible dans un environnement web (les *geometry shaders* sont absents par exemple).

D'autres spécificités du développement web sont à prendre en compte pour la réalisation d'applications 3D. Tout d'abord, les performances du code JavaScript exécuté au sein du navigateur sont plus faibles que l'équivalent en C/C++ ou autre langage compilé haute performance sur bureau[4]. Cette assertion est tout de même à relativiser, des progrès majeurs ayant été faits grâce au récent développement de WebAssembly [Haa+17], un langage de bas niveau binaire généré à partir d'un langage haut niveau (C++ ou Rust actuellement) qui peut être exécuté par le navigateur et qui obtient des performances proches du code natif C++ (de l'ordre de 1.1 à 2 fois le temps d'exécution).

En attendant la généralisation de WebAssembly dans le développement web, il reste prudent de faire particulièrement attention au code exécuté par le navigateur, d'autant plus que la moitié des pages web sont accédées depuis un appareil

[1] https://www.w3.org/TR/html5/
[2] https://www.khronos.org/webgl/
[3] On considère comme application de bureau toute application prévue pour s'exécuter directement sur un ordinateur fixe, par opposition aux applications web qui sont exécutées à l'intérieur d'un navigateur web.
[4] https://kripken.github.io/mloc_emscripten_talk/gindex.html#/17

mobile[5] (téléphone ou tablette) à la puissance limitée. Il est préférable d'avoir un comportement très simple pour le code client, de se rapprocher le plus possible d'un comportement « passe-plat » — c'est-à-dire de transférer directement les données reçues à la carte graphique, sans traitement JavaScript au préalable.

Un deuxième élément crucial à prendre en compte est le fait que les données ne sont généralement pas sur la même machine que celle qui exécute l'application web. Cela signifie que les données — potentiellement volumineuses — doivent transiter sur le réseau : il faut prendre en compte la taille et la compression des données dans la mesure de performances. Cela signifie aussi qu'en fonction de la qualité du réseau et de la puissance des terminaux, certaines solutions seront préférables à d'autres : avec un débit faible, une compression forte est préférable, même si les données sont plus longues à décoder côté client [Lim+13].

Du fait de la faible puissance de certains clients, il peut être intéressant de déporter une partie des tâches côté serveur, d'autant plus que le serveur n'est pas concerné par des restrictions sur le choix du langage et peut donc être plus performant. Cependant, comme de nombreux clients se connectent habituellement au même serveur, le serveur pourrait ne pas passer à l'échelle s'il a à sa charge trop de calculs à effectuer sur les données. Il faut trouver le meilleur compromis entre ces contraintes en tenant compte de l'usage qui sera fait de l'application.

Comme les données prennent du temps à transiter sur le réseau, il peut être intéressant de mettre en place une progressivité du chargement des données : formater ou organiser les données de manière à ce qu'elles apparaissent au fur et à mesure du téléchargement. Bien que cela puisse rendre le chargement complet de la scène plus lent, il est plus agréable pour l'utilisateur de voir ses données arriver rapidement et se raffiner ensuite que d'attendre longtemps avant que quoi que ce soit ne soit visible.

Pour approfondir le sujet de la 3D sur le Web, [Chu12] est dédié au compte-rendu d'un des premiers projet web 3D et fait un panorama des problématiques techniques du sujet. [Eva+14a] propose un état de l'art des différentes méthodes de rendus, des méthodes de compression et des cas d'utilisations de la 3D sur le Web. La suite de cette section s'attardera sur des éléments choisis concernant la compression / formatage de données et les méthodes de rendu.

2.3.1 Compression et formatage

Pour transporter les données 3D sur le réseau dans l'optique d'une visualisation web, deux familles de techniques existent.

[5] https://www.statista.com/statistics/241462/global-mobile-phone-website-traffic-share/

La première vise à réduire la taille des données en les compressant. [Mag+15] présente un état de l'art très complet sur les techniques de compression de maillage. Les auteurs précisent là encore l'importance d'adapter les algorithmes de compressions aux spécificités du Web, notamment en évitant ou désactivant certaines étapes de compression qui sont trop complexes à décompresser. [Chu12] recours à la quantification, au delta encoding, au zigzag encoding. En outre, il exploite au maximum les spécificités du Web en encodant les données en UTF-8 (efficace pour le transfert en HTTP) et en compressant les données encodées avec l'algorithme gzip (implémenté nativement dans le navigateurs, très rapide à décompresser car ne dépendant pas de JavaScript). [Lav+13] propose une compression progressive — c'est-à-dire qui génère plusieurs niveaux de détails pour les maillages — et un algorithme de décompression basée sur les cartes combinatoires. [Gob+12] et [PD15] présentent des solutions de rendus adaptatifs : le maillage est plus ou moins détaillé en fonction de la distance des triangles à la caméra.

La seconde vise à réduire le temps de calcul du client web en formatant les données de telle sorte qu'elles n'aient pas besoin d'être transformées avant d'être visualisées. glTF[6] (GL Transmission Format) est un format conçu par le Khronos Group pour le transfert de maillages 3D. Il est composé d'une partie métadonnées en JSON décrivant la composition et la structure des objets stockés sous forme binaire. L'attrait principal de ce format est que les données binaires sont directement interprétables par la carte graphique : les tableaux d'indices, de positions, de normales sont déjà dans la forme attendue. PoTree [Sch16], une application web de visualisation de nuages de points, et [Eva+14b] utilisent tous les deux une structure binaire similaire, cette fois pour les points, où les données (positions, couleurs) sont stockées dans des tableaux binaires.

Suivant ce même modèle (métadonnées en JSON, géométries en binaire), le futur *community standard* 3D Tiles[7] définit un ensemble de formats pour divers modèles de données : nuage de points, maillages, données vectorielles... L'objectif de 3D Tiles est de gérer des jeux de données vastes. Ainsi, ces formats permettent de décrire un ensemble de tuiles qui sont organisées dans un *tileset*, un fichier spécifiant un graphe de scène et les informations nécessaires pour naviguer dans la scène (qui sera décrit plus précisément en section 4.2). En pratique, une application 3D commence par télécharger le tileset puis, en fonction de la position de la caméra et la description de la scène contenue dans le tileset, télécharge les fichiers correspondants aux tuiles dans le champ de vision de la caméra à un niveau de détail adapté. i3s[8] (Indexed 3D Scene Layer) est un standard proposé par ESRI très similaire à 3D Tiles qui se concentre plus sur l'aspect géospatial.

[6] https://www.khronos.org/gltf/
[7] https://github.com/AnalyticalGraphicsInc/3d-tiles
[8] https://github.com/Esri/i3s-spec

Citons également Blast [Sut+14], un format binaire dont l'objectif est de réduire le nombre de requêtes à envoyer par le client en offrant la possibilité de transmettre plusieurs types de données dans la même réponse à une requête. Blast est un conteneur générique, pouvant contenir n'importe quel type de données binaires et permet le chargement progressif des données (*streaming*).

Le format X3D[9], successeur de VRML[10], est un standard ouvert créé par le Web 3D Consortium basé sur le formalisme XML. Le format est extensible et a notamment intégré des concepts liés au géospatial et au Web. Il est très utilisé dans une approche déclarative au rendu sur le Web (voir section suivante).

2.3.2 Méthodes de rendu

Il existe deux approches pour le rendu sur le Web : l'approche déclarative et l'approche impérative [Jan+13].

L'approche déclarative est intégrée au DOM (*Doculent Object Model*, la hiérarchie d'objets du HTML décrivant le document de la page web), l'ensemble de la scène est décrite en rajoutant des nœuds spécialisés dans celui-ci, formant ainsi un arbre de scène. X3DOM [Beh+09] et XML3D [Son+10] sont les deux représentants de cette approche. L'avantage de ces méthodes est qu'elles restent très proches des principes du Web, rendant leur utilisation simple d'accès, et qu'elles fonctionnent nativement sur la plupart des plateformes. Leur principal défaut est un manque de flexibilité, il est notamment impossible d'agir sur le pipeline de rendu.

L'approche impérative consiste à programmer les instructions nécessaires pour effectuer le rendu. Précédemment réalisé avec des plugins tels que Unity3D[11], l'arrivée de WebGL a permis des solutions basées uniquement sur du code JavaScript fonctionnant nativement sur les navigateurs modernes sans le besoin d'installer un plugin. Des bibliothèques 3D telles que Three.js[12] ou Babylon[13] reprennent ce paradigme et offrent une abstraction de WebGL pour simplifier le travail des développeurs.

Une autre possibilité est de faire du rendu distant, c'est-à-dire de faire le rendu sur un serveur distant plutôt que sur le navigateur. Au lieu de transmettre les éléments de la scène, le serveur envoie l'image résultant du rendu de la scène au client. Cela allège considérablement la charge du client, mais la reporte sur le serveur. Cela pose des questions sur le passage à l'échelle de telles techniques, les opérations

[9] http://www.web3d.org/x3d/what-x3d
[10] https://web.archive.org/web/20140304011557/http://www.web3d.org/x3d/vrml/
[11] https://unity3d.com/fr/webplayer
[12] https://threejs.org/
[13] https://www.babylonjs.com/

de rendus étant habituellement assez lourdes. Pour éviter ce problèmes, certaines solutions[14] limitent le nombre de points de vues disponibles et génèrent les images correspondant à chaque point de vue à l'avance, faisant ainsi un compromis entre la performance d'un côté et la liberté de déplacement et le coût mémoire (pour stocker les images) de l'autre. Un autre problème de cette méthode est l'ajout d'un temps de latence conséquent sur les interactions de l'utilisateur. Pour plus de détails sur cette approche, [Eva+14a] dédie une section de son état de l'art sur la question.

2.4 3D et géospatial

Comme évoqué dans l'introduction, il existe de nombreux cas d'utilisations pour les maquettes 3D de villes [Bil+15]. Cette diversité requiert donc d'étudier les différents modèles de données que nous pouvons rencontrer et quels standards ont été mis en place pour favoriser l'interopérabilité et la réutilisation de données. Nous verrons aussi comment l'aspect multi-échelle de la ville peut être pris en compte à travers les techniques de généralisation et comment tous ces éléments permettent la création de maquettes numériques.

2.4.1 Standards

Dans le domaine de l'informatique géospatiale, il est courant d'utiliser des données issues de sources différentes et que l'on a pas produit soi-même. On comprend alors la nécessité de concevoir des solutions interopérables et de définir des standards pour communiquer les données. Deux sortes de standards peuvent être identifiées : les protocoles standards et les formats standards. Certains formats standards 3D génériques ont déjà été présentés dans la section 2.3.1, dans cette section nous verrons les standards spécifiquement liés à l'informatique géospatiale.

CityGML CityGML [Kol+05] (*City Geography Mark-up Language*) est un standard proposé par l'OGC (*Open Geospatial Consortium*, principal organisme de standardisation du domaine géospatial) d'échange et de stockage de modèles virtuels de la ville. Basé sur le formalisme XML, il permet de modéliser à la fois la géométrie et la sémantique d'une ville virtuelle 3D. CityGML spécifie cinq niveaux de détails pour représenter la ville (fig. 2.12) :

- le LoD 0 constitué uniquement du modèle numérique de terrain en 2.5D ;
- le LoD 1 qui représente les bâtiments comme des boîtes avec un toit plat ;
- le LoD 2 qui représente les bâtiments avec leur toit détaillé et distingue sémantiquement les différentes parties du bâtiment ;

[14] https://www.3dcontentlogistics.com/en

- le LoD 3 qui représente les bâtiments avec davantage de détails, tels que la modélisation des balcons et des corniches, et introduit le mobilier urbain ;
- le LoD 4 qui ajoute au LoD 3 la modélisation de l'intérieur des bâtiments.

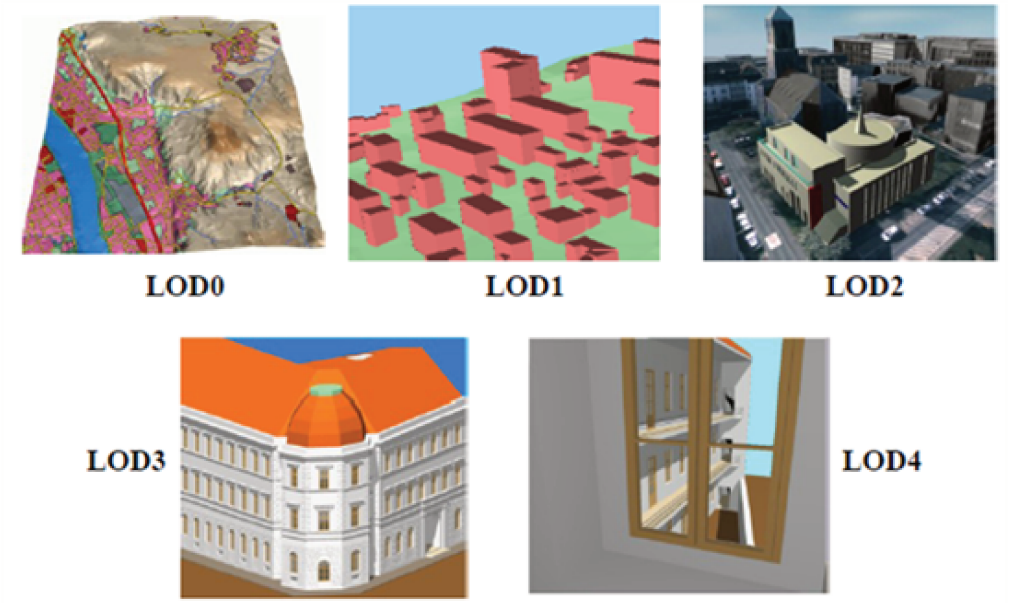

Figure 2.12 Les différents niveaux de détails du standard CityGML. Image extraite de [Kol+05].

CityGML est extensible au travers d'ADE (*Application Domain Extensions*) qui enrichissent le modèle CityGML avec des informations spécifiques à un domaine. On peut citer par exemple UtilityNetworkADE [Bec+11] qui permet la modélisation des réseaux de services d'utilité publique (réseaux hydraulique, électrique, etc.) ou Energy ADE [Nou+15] qui ajoute des propriétés nécessaire aux simulations énergétiques.

CityGML est un format extrêmement complet mais également lourd et complexe. Ainsi, il est rarement utilisé en tant que format de transfert pour les applications web, en faveur d'autres formats plus légers [Cha14]. Cependant, comme les données originelles sont en CityGML, il reste structurant et les concepts qu'il introduit restent pertinents.

Protocoles L'OGC a développé un ensemble de standards pour spécifier les échanges des différents types de données utilisés en informatique géospatiale. WMS (*Web Map Service*) [Bea06] est un protocole pour l'échange d'images de terrains (fond de plan, carte d'élévation, etc.). WMTS (*Web Map Tile Service*) [Mas+10] sert la même fonction que WMS, mais fonctionne avec un ensemble de tuiles prédéfinies au lieu de coordonnées arbitraires, dans l'objectif d'être plus performant. WFS (*Web Feature Service*) [Vre14] est un protocole permettant de récupérer des données vectorielles contenues dans une boîte englobante définie.

Ces trois protocoles ont été développés initialement dans un contexte 2D, même s'ils sont encore parfois utilisés pour charger des géométries 3D. Cependant, un protocole dédié à la 3D à ses avantages pour gérer les spécificités apportées par la troisième dimension. 3DP [Hag+17] (*3D Portrayl*) est ce protocole. Il offre deux modes de fonctionnement : le mode *scène* et le mode *vue*. Le mode *scène* permet l'échange de données géométriques 3D entre le client et le serveur, le rendu de la scène étant réalisé sur la machine du client. Le mode *vue* est un mode de rendu à distance : le rendu est calculé sur le serveur distant et le client reçoit l'image générée (comme expliqué en section 2.3.2). Récemment, [Gut+16] a proposé une architecture serveur implémentant 3D Portrayal.

2.4.2 Généralisation

La généralisation est le processus qui consiste à sélectionner et à représenter l'information d'une carte (en 2D) ou d'une scène (en 3D) de manière à l'adapter à l'échelle de la visualisation ou à l'importance qui lui est accordée. Ce processus a été très recherché en 2D pour la génération automatique de cartes à plusieurs échelles [Rua04] et a été étudié de manière plus limité en 3D, notamment à travers les thèses de Mao [Mao11], He [He12] et Glander [Gla13]. D'un point de vue d'informaticien graphiste, la généralisation permet la production de niveaux de détails. La thèse de Biljecki[Bil17] comprend un état de l'art très complet de méthodes de génération de niveaux de détails dans le contexte de la ville.

Dans sa thèse, Shuang He identifie des défis pour la production de modèles généralisés d'une part, et de la visualisation de ces modèles d'autre part. Pour la production :

- la conformité : la position, les mesures et la topologie de l'objet doivent être conforme à la réalité ;
- la diversité : des objets de natures radicalement différentes (terrain, bâtiment, eau, végétation...), pouvant avoir plusieurs représentations (maillage, image...), nécessitent des traitements différents ;
- le détail : limitation de la quantité de détails captables par les appareils de mesures, difficulté de combiner des modèles de niveaux de détails différents ;
- l'utilisabilité : les contraintes sont différentes en fonction du cas d'utilisation (stockage, échange, modification, extension, rendu...), il faut veiller à l'interopérabilité et aux standards.

Pour la visualisation :

- visualisation multi-échelle : définir des niveaux de détails en fonction de la distance de vue pour surmonter les limitations matériel. Cela nécessite de produire des niveaux de détails à partir du modèle initial ;
- lisibilité : améliorer la lisibilité d'une scène en n'affichant les détails que là où cela est nécessaire ;
- performance du rendu : transitions "douces" entre niveaux de détails, faible performances des clients mobiles ;
- information hétérogène : incompatibilité et complexité des différentes sources de données.

Il existe de multiples classes de méthodes de généralisation, certaines s'appliquant à un objet unique, d'autres à des groupes d'objets. Voici les principales.

Simplification Cette catégorie de généralisation consiste en la suppression de détails d'un objet. Il existe de nombreuses méthodes de simplification de bâtiments. [Thi02] segmente le bâtiment et en déduit un arbre de construction. On peut alors obtenir une version simplifiée du bâtiment en supprimant les branches de l'arbre de construction qui forment les détails. [Kad02] utilise la méthode des moindres carrés alliée à la détection de caractéristiques pour simplifier ses géométries. [For07] utilise des opérations morphologiques (dilatation, érosion, ...) à différentes échelles pour générer des niveaux de détails. [Kad07] découpe l'espace par des plans verticaux suivant les façades du bâtiment, formant un ensemble de cellules. Les cellules qui recoupent fortement la géométrie du bâtiment sont conservées, un toit leur est ajouté, et ainsi une version simplifiée est formée (fig. 2.13). Les méthodes précédentes se basent uniquement sur la géométrie des bâtiments. D'autres approches se servent des informations sémantiques liées pour affiner leurs résultats. L'algorithme présenté dans [Fan+09] utilise les informations stockées dans les fichiers CityGML pour généraliser des bâtiments. [Ver+15] génère un ensemble de niveaux de détails à la suite d'une reconstruction de maillages à partir de nuages de points.

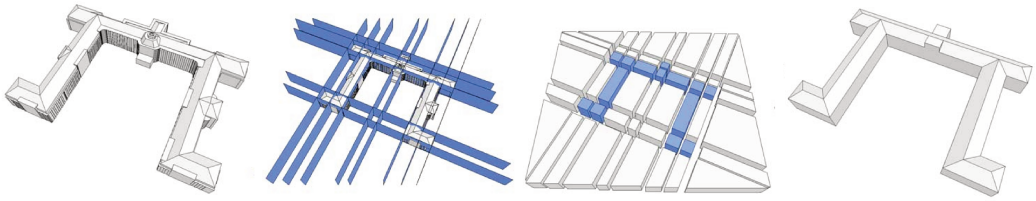

Figure 2.13 Plusieurs étapes de l'algorithme de simplification de [Kad07]. De gauche à droite : le modèle original, la définition des plans de coupes, les cellules résultantes et le modèle simplifié.

Agrégation L'agrégation consiste en la fusion d'un ensemble d'objets en un seul objet les représentant tous. [And05] réalise l'agrégation de groupes de bâtiments alignés en les projetant selon trois axes orthogonaux et en reconstruisant une géométrie unique à partir de ces silhouettes. [Gue+11] modélise l'agrégation de bâtiments sous la forme d'un problème de *Mixed Integer Programming* ce qui lui

permet de le résoudre efficacement grâce aux nombreux outils existant. [He+12] se base sur la décomposition des bâtiments en deux parties (*Top* et *Body*) et leurs empreintes pour générer des généralisations à différents niveaux de détails (fig. 2.14). [Cha+08] présente une méthode s'inspirant du concept de « lisibilité urbaine » défini par l'urbaniste Lynch [Lyn60] pour créer des représentations généralisées texturées de la ville plus claires pour l'observateur.

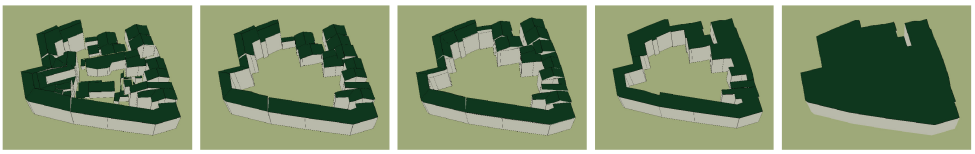

Figure 2.14 Différentes représentations d'un groupe de bâtiments créées par l'algorithme de généralisation par agrégation de [He+12]. À gauche, le modèle original, ensuite, de gauche à droite, des représentations de plus en plus simplifiées.

Typification La typification est la réduction de la densité et du niveau de détail d'un ensemble d'objets en conservant la structure de l'ensemble ou le motif de distribution des objets originaux. [And05] propose une typification fonctionnant par la détection d'un motif de grille, puis par un sous-échantillonnage des bâtiments répartis au sein des cellules de la grille. En utilisant une triangulation de Delaunay sur les objets de la ville, [BC07] parvient à remplacer ces objets par un plus petit ensemble d'objets et à déterminer quelles formes et tailles leur donner. [MB10] utilise des *Minimum Spanning Trees* (*MST*) pour grouper des bâtiments. Les bâtiments les plus petits sont ensuite peu-à-peu supprimés, ceux restants étant déplacés et agrandis pour compenser visuellement cette disparition. [Mao+11] se base sur le réseau routier pour déterminer à quel groupe de bâtiments le processus de typification doit être appliqué.

Classification et symbolisation Ces classes de méthodes groupent des objets similaires dans un objet de plus haut niveau et le représentent avec un nouveau symbole. [TS06] cherche parmi un ensemble de modèles génériques de bâtiments lequel convient le mieux pour le bâtiment à généraliser et adapte le modèle en fonction des dimensions du bâtiment. [GD09] introduit le concept de niveau d'abstraction, analogue au niveau de détail, qui propose de représenter les objets de la ville de manière plus ou moins abstraites en fonction de l'importance qui leur est accordé (fig. 2.15).

Sélection La sélection permet de représenter un groupe d'objets par un sous-ensemble de ces objets. [**DBLP:journals/corr/CuraPP16**] fait de la généralisation par sélection pour les nuages de points (fig. 2.16). L'espace est découpé en un ensemble de patchs, en se basant sur une grille 3D régulière, puis une sélection des points est effectuée à partir d'un octree pour chaque patch : à chaque niveau

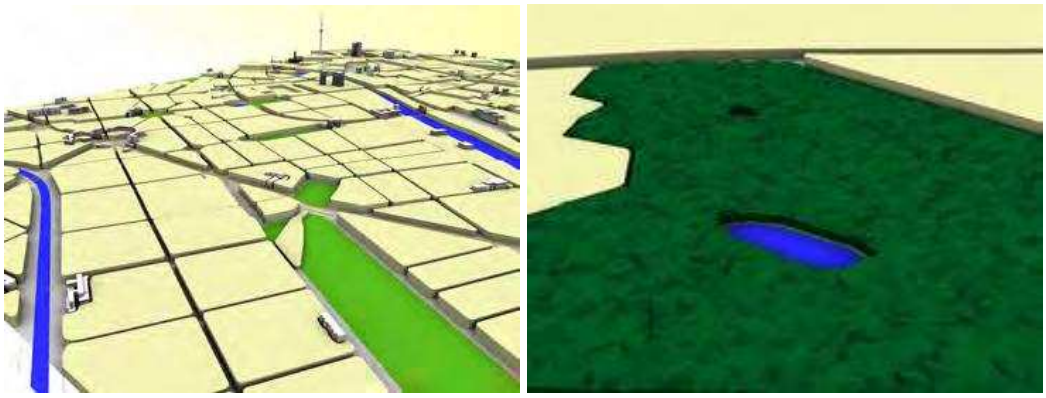

Figure 2.15 Représentations abstraites d'îlots urbains (gauche) et d'une forêt (droite) dans [GD09].

d'échelle, on conserve le point qui est le plus proche du centre de chaque cellule de l'octree.

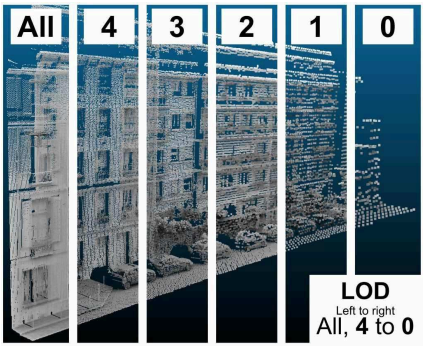

Figure 2.16 Niveaux de détails générés par la méthode de généralisation de [**DBLP:journals/corr/CuraPP16**].

2.4.3 Maquette numérique

Une maquette numérique est une représentation 3D d'un ensemble d'objets à des fins de visualisation et d'analyse. Elle peut être vue comme un « hub » où des données variées sont centralisées : terrain, bâtiments, végétation... Elle est utilisée comme outil d'échange et de compréhension de la ville. L'objectif est de pouvoir charger au sein d'une même maquette toutes les données pertinentes pour l'analyse envisagée.

Les technologies et méthodes présentées précédemment offrent différentes manières de créer ces maquettes.

Représenter les données géographiques à taille réelle implique un certains nombres de défis propres à la gestion de coordonnées de grandes tailles et la gestion d'objets d'échelles radicalement différentes. [CR11] traite ces problèmes dans un livre présentant les principes de conception d'applications de globes virtuels, principes qui ont été appliqués pour la création des applications NASA World

Wind et Cesium. Le livre contient également des indications sur la préparation et la visualisation de données vectorielles et du terrain.

Les nombreux modèles de données présents dans les maquettes numériques ont souvent été étudiées séparément.

[Ler09] s'intéresse au streaming de terrain, notamment via l'aspect d'adaptation du niveau de détail. [Cel14] étudie le problème d'affichage de terrain dans un contexte web, axant ses travaux sur le parallélisme et le Web haute performance. [Chr16] contrôle la quantité de données affichées simultanément sur une application de globe virtuel, ce qui est utile pour gérer efficacement sa consommation des ressources.

La visualisation des bâtiments de la ville, sous la forme de maillage le plus souvent, est un des aspects les plus étudiés dans l'état de l'art. De nombreuses applications de visualisation de données urbaines, généralement extraite de CityGML, ont été proposées. [MB11] et [Pri+12] présentent des applications basées sur une approche déclarative du Web 3D, X3DOM. [GM12] utilise WebGL pour son application. Pour pouvoir gérer des villes entières, celles-ci sont découpées selon une grille. [Mao+11] propose CityTree, une structure de données pour la visualisation de modèles 3D de bâtiments. Les feuilles sont constituées des bâtiments originaux, tandis que le reste des nœuds contiennent des représentations généralisées de groupes de bâtiments. [KG15] étudie plusieurs solutions, basées sur X3DOM, Three.js et Cesium, concluant qu'elles sont toutes viables et que le choix optimal dépend du cas d'utilisation. [EA14] réalise un ensemble d'optimisations pour la visualisation de la ville sur appareils mobiles. Cependant, les modifications apportées aux formats de données ne permettent pas de respecter les standards et ne garantissent pas l'interopérabilité de sa solution.

Le nuage de point devenant un modèle de données de plus en plus commun, l'intégration de ces données aux maquettes numériques a été étudiée. Le plus souvent le nuage de points est préparé en avance pour optimiser la vitesse de rendu [KG15] [Eva+15], mais d'autres solutions se base sur une base de données pour permettre une plus grande flexibilité à l'usage, comme [Cur+15].

Représentation de la donnée Nous avons vu précédemment qu'un même objet pouvait être représenté de manière différente selon le contexte et les besoins de l'utilisateur. Cette stylisation des données est appliquée dans les maquettes numériques de plusieurs manières.

Une symbologie peut être appliquée aux données pour présenter des informations sémantiques. [Gio+14] colore les bâtiments pour indiquer leur efficacité énergétique, [Red+17] pour indiquer le résultat de simulations de tremblements de terre.

La stylisation peut être utilisée pour simplifier ou mettre en valeur les données pour en améliorer leur compréhension. [Gla+10] transforme en temps réel le terrain de telle sorte que l'élévation soit discrétisée en un nombre fixes de valeurs, à la manière de lignes de niveaux. [HD07] réalise des vues en éclaté de bâtiments pour permettre aux utilisateurs d'en voir l'intérieur. La stylisation peut aussi être utilisé dans un but esthétique et de clarté. Par exemple, un rendu non-photoréaliste est obtenu par [Kra+17] pour le rendu de terrain. [Bra+15] propose une méthode pour classifier et constituer une base de connaissances de styles de rendu dans le but de mélanger différents styles entre eux.

La notion de *focus* permet de définir une partie de la scène comme étant une zone d'intérêt, le reste de la scène constituant le *contexte*. [Tra+08] propose une implémentation du focus avec sa « lentille de généralisation ». Des volumes sont définis et associées à des généralisations. Lors du rendu, si un pixel correspond à un de ces volumes, seule la généralisation correspondant est affiché (fig. 2.17, gauche). Un problème de cette méthode est sa grande consommation de ressources : même si une généralisation n'apparaît pas, elle passera par toutes les étapes de traitement du pipeline de rendu. [Bra+16] défini deux symbologies différentes entre les bâtiments du focus et du contexte pour une application de gestion de plan local d'urbanisme. [Fil+16] utilise cette même notion pour protéger les informations privées de particuliers (fig. 2.17, droite).

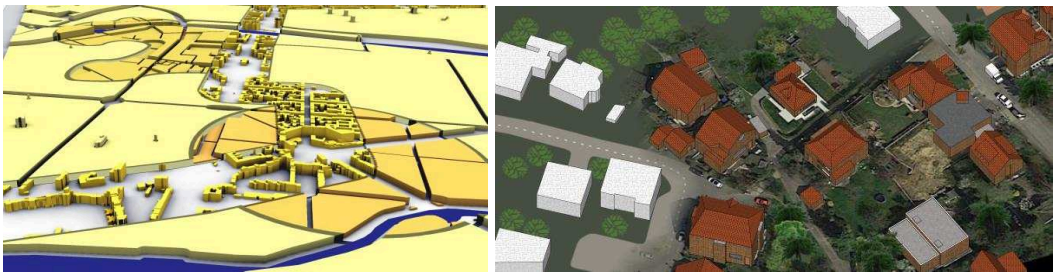

Figure 2.17 Les lentilles de généralisation introduites par [Tra+08] (gauche) et l'occultation de données privées de [Fil+16] (droite).

Interaction Dépasser la simple visualisation de données en ayant la possibilité d'interagir avec les données est un enjeu clef des maquettes numériques.

L'interaction la plus fréquente est le *picking*, la possibilité de cliquer sur un objet pour récupérer les informations sémantiques qui lui sont liées [Sch+16] [Pra+14]. [Pra+12] affiche l'historique de données de capteurs. [Cha+15] va plus loin dans cette veine, en permettant l'exploration de la structure des objets CityGML.

L'interaction peut aussi se matérialiser par des opérations géométriques réalisées sur les données. [Cha14] et [Wal+09] utilisent de l'analyse géométrique 3D pour faire une application simple de prévention des risques. [Pra+15] et [Dei+17] offrent des outils de mesures.

Des méthodes d'édition peuvent également être proposés. [Pra+15] décrit une application de gestion de forêt qui aide l'utilisateur à placer des téléphériques pour la collecte de bois, un des cas d'utilisation étudié par [KG15] permet de déplacer des bâtiments dans la scène.

Stocker les objets dans une base de données donne accès à beaucoup de possibilités d'interaction. [Cur+15], avec son serveur de points basé sur PostGIS, montre qu'il est possible d'effectuer une variété de traitements et de transmettre les résultats obtenus au client.

2.5 Synthèse

L'état de l'art fournit une grande quantité d'outils et de méthodes utiles pour appréhender les problématiques de ma thèse. Les techniques développées pour le rendu temps réel sont des sources d'inspiration pour le développement de structures de données efficaces adaptées aux données géographiques. La nécessaire création de niveau de détails est un sujet qui a déjà été particulièrement étudié à travers les multiples méthodes de généralisation, facilitant la visualisation multi-échelle de données géographiques. Dans un contexte web, nous avons vu que l'association du JavaScript et du WebGL offre la flexibilité nécessaire au développement d'applications 3D interactives, bien qu'il faille veiller à la charge de travail effectuée par le client pour éviter des problèmes de performances. L'interopérabilité des méthodes proposées dans ce manuscrit est garantie par l'existence de standards adaptés comme 3D Tiles.

Il reste encore des problématiques non résolues. La majorité des méthodes actuelles se limitent à la visualisation de données statiques. L'aspect dynamique des données (comme l'édition des données, ou la sélection d'un sous-ensemble particulier des données) est rarement abordé, et quand il l'est, c'est pour des situations très précises. La gestion des différentes représentations des objets géographiques est aussi limitée, elle est généralement étudiée seulement à travers le concept de couches de données : il n'y a pas de lien entre les représentations d'un même objet.

Dans mes travaux, je propose plusieurs approches pour permettre l'édition des données, la gestion de l'aspect multi-représentations des objets géographiques et la

personnalisation de la représentation de la ville tout en conservant des capacités de visualisation de la ville à grande échelle.

3 Socle technique

Sommaire

Les méthodes développées durant ma thèse se basent toutes sur le même socle technique : un ensemble d'outils, de logiciels développés par la communauté Open Source, les employés d'Oslandia ou moi-même. Ce chapitre est consacré à ces outils, leur agencement et aux problématiques qui ont entourées le développement de certaines solutions techniques.

Ces éléments seront abordés en trois temps. Tout d'abord, le processus menant à la visualisation sera étudié de manière globale, afin de définir quelles opérations effectuer côté serveur et quelles opérations effectuer côté client. Ensuite la partie serveur et la partie client seront abordées séparément. Pour chacune de ces parties, l'état de l'art technique sera présenté avant d'expliciter les choix techniques ou implémentations réalisées.

3.1 Architecture générale

L'architecture générale désigne l'architecture de la totalité de l'application de géovisualisation, du serveur au client, des données stockées à l'image rendue sur l'écran de l'utilisateur. Dans cette section, je présente le pipeline de visualisation tel que décrit par l'état de l'art, ainsi que l'implémentation de ce pipeline que j'ai choisie.

3.1.1 Architecture d'une application de géovisualisation

[DC98] propose un pipeline de visualisation (fig. 3.1) qui décrit les processus de traitement de données menant à la visualisation pour les applications géospatiales web. Ce pipeline est notamment utilisé dans des spécifications de l'Open Geospatial Consortium pour introduire leurs standards [Hag+17].

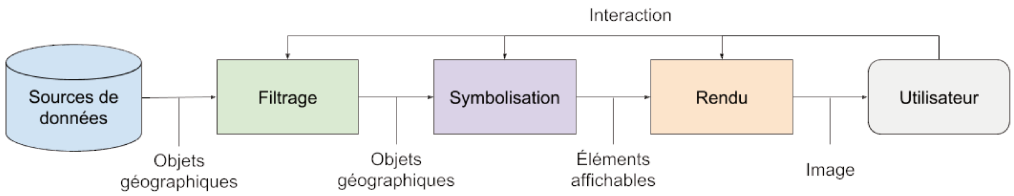

Figure 3.1 Présentation du pipeline de visualisation de données géospatiales. Si l'application le permet, l'utilisateur peut intervenir sur chacune des étapes pour modifier la manière dont les données sont affichées et leurs natures.

La fig. 3.1 introduit un code couleur qui sera repris tout au long de ce chapitre pour tous les schémas d'architecture. Les données sont en bleu, les étapes de filtrage en vert, les étapes de symbolisation en violet, les étapes de rendu en orange et les autres éléments neutres vis-à-vis de cette classification en gris.

Ce pipeline est composé de trois grandes étapes entre les sources de données et l'utilisateur : le filtrage, la symbolisation et le rendu. En fonction de la manière dont sont implémentées les étapes, l'utilisateur peut avoir un contrôle sur chacune d'entre elles.

Filtrage Le filtrage est la phase de sélection des données à visualiser. La sélection est double : à la fois sur les objets géographiques (quels objets faut-il visualiser ?) et sur les informations liés à ces objets (de quels attributs a-t-on besoin pour afficher cet objet géographique ?). Le filtrage peut se faire de multiples manières, en fonction de la position spatiale ou de la valeur des attributs des objets. Un exemple de filtre simple est une sélection de tous les bâtiments contenus dans un rectangle précis dont la taille dépasse un seuil. La phase de filtrage retourne un ensemble d'objets géographiques qui sera traité par la suite du pipeline.

Symbolisation La symbolisation (l'étape est appelée *mapping* ou *display element generator* en anglais) est une étape dont l'objectif est de générer des éléments affichables à partir d'objets géographiques. En d'autres termes, il s'agit de définir la représentation des objets géographiques à partir d'un style donné.

Rendu La phase de rendu consiste en la génération d'une image à partir d'un ensemble d'éléments affichables. Cette phase à déjà été présentée durant l'état de l'art (section 2.2), et possède son propre pipeline qu'il faut veiller à ne pas confondre avec le pipeline de visualisation qui l'englobe.

3.1.2 Architecture retenue

Dans un contexte web, le pipeline de visualisation est partagé entre le client et le serveur. Chaque étape peut-être effectuée d'un côté ou de l'autre. Par exemple, en ayant toutes les étapes côté serveur, on obtient du rendu distant (comme évoqué en partie 2.3.2). La répartition des étapes choisie dans la plupart des applications du marché est la suivante. Le serveur filtre les données le client effectue le rendu. La symbolisation est partagée entre les deux, le serveur générant les éléments affichables et le client gérant la modification d'apparence (coloration sémantique, etc.). Le choix de cette approche se justifie par la grande réactivité qu'elle offre au client lors des interactions avec la maquette et de la modification de certains éléments de symbologie. Elle a été retenue dans le cadre de cette thèse pour ces mêmes raisons.

Un schéma abstrait d'architecture pour ce type d'approche est présenté en fig. 3.2. L'utilisateur déplace la caméra, dont la position et l'orientation vont déterminer quelles parties de la scènes doivent être affichées. Des requêtes sont générées et

envoyées au serveur pour charger les données à afficher que le client ne possède pas déjà. Le serveur analyse ces requêtes, et sélectionne les données demandées, les transforme si besoin, puis les convertit dans un format adapté pour le transfert et la visualisation sur le Web. À la réception des données, le client applique un style selon les préférences de l'utilisateur, puis les affiche après une phase de rendu.

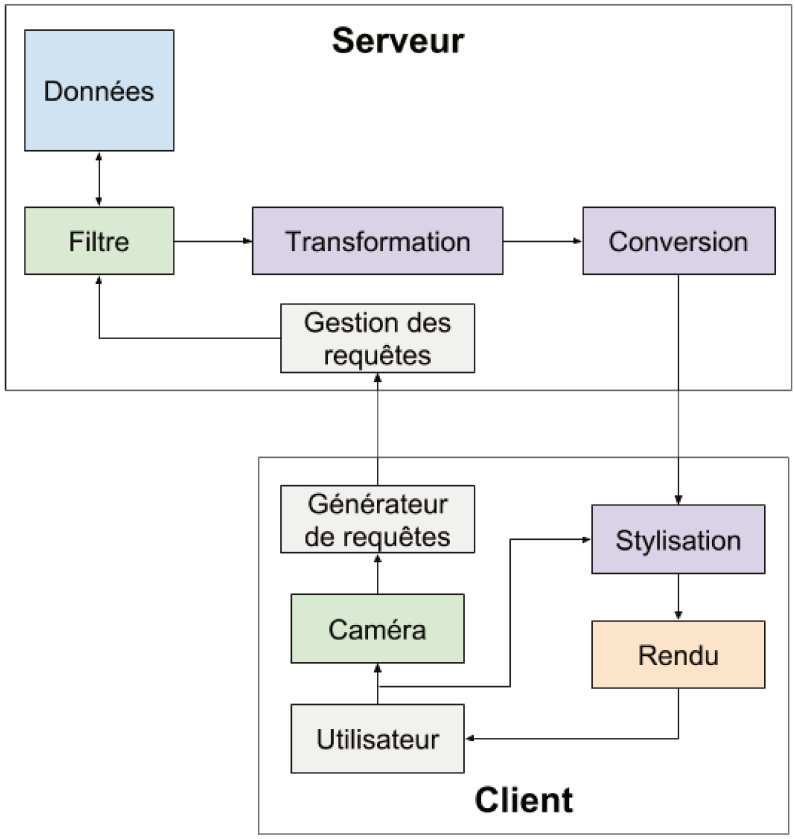

Figure 3.2 Schéma de l'architecture client-serveur abstraite du pipeline de visualisation.

3.2 Serveur

Les différentes méthodes qui seront présentées au cours de cette thèse n'utilisent pas toutes les mêmes processus de traitement des données côté serveur. Cependant, on peut tout de même extraire une architecture générique et des outils qui sont partagées. C'est l'objet de cette section, qui introduit des logiciels qui ont été développés au cours de cette thèse (py3dtiles) ou qui ont été structurants dans les approches conçues (PostGIS), ainsi qu'une architecture générique pour le traitement de données géospatiales 3D.

3.2.1 Techniques existantes

Il y a deux stratégies différentes pour implémenter la partie serveur de notre architecture de géovisualisation sur le Web.

La première, et plus commune, se retrouve chez les gros acteurs du marché, comme AGI (Cesium) ou Google. Il s'agit de préparer les données en avance, de générer un cache, dans un format très rapide à interpréter par le navigateur. Le but de cette approche est de minimiser le temps de réponse du serveur afin d'avoir l'application la plus rapide possible, en échange de la perte de la possibilité d'interagir significativement avec les données (modification ou analyse).

La seconde, qu'on trouve essentiellement pour les données 2D, consiste à générer les données à la volée, c'est-à-dire que les données sont choisies et converties dans un format adapté au transfert et/ou à la visualisation lorsque le serveur reçoit une requête. Bien que la réponse aux requêtes prennent alors plus de temps, il est cette fois possible de modifier les données sans avoir à régénérer un nouveau cache et d'interagir avec de manière bien plus poussée.

Ces deux stratégies ont leurs cas d'utilisation, c'est pourquoi le socle logiciel que j'ai produit est capable de supporter les deux. Je présente ci-après mon implémentation de ces deux stratégies puis en fais l'étude comparée.

3.2.2 Choix techniques et contributions

Afin d'aborder les problématiques de ma thèse qui n'ont pas été résolues par les applications de l'état de l'art, j'ai développé ou contribué au développement de diverses applications. Cette section explique les choix techniques qui ont été réalisés et décrit les divers développements qui ont été réalisés. Je présente successivement les travaux effectués sur la partie serveur puis sur la partie cliente de ma solution.

Dans mes implémentations de l'architecture serveur d'une application de géovisualisation, je me base sur deux modules, présentés en détail dans la suite de ce chapitre :

- PostGIS, un système de gestion de base de données géospatiales qui permet de stocker, filtrer et transformer des données géographiques 3D ;
- py3dtiles, un module python pour convertir les données en sortie de PostGIS dans le format 3D Tiles.

Les choix architecturaux et de standards ont été faits dans l'optique de garantir l'interopérabilité des méthodes développées pendant ma thèse. Il doit être possible

de changer certains modules sans remettre en cause le fonctionnement de toute la chaîne logicielle. Par exemple, PostGIS peut être remplacé par SpatialLite sans remettre en cause l'architecture globale.

3.2.2.1 Architecture

Le serveur dans le modèle d'application géospatiale web présenté précédemment a deux rôles : indiquer au client de quelles données il dispose et fournir au client le sous-ensemble de données qu'il demande, dans le format qu'il souhaite. Pour accomplir ces missions, il peut également être nécessaire d'effectuer un certain nombre de pré-traitements aux données initiales. Cela nous amène à proposer un découpage des tâches du serveur en trois phases :

- la phase de pré-traitement, la préparation des données ;
- la phase d'initialisation, où le serveur indique au client ce qu'il peut requêter et comment le faire ;
- la phase de transfert, où le serveur envoie au client les données qui lui sont demandées.

Ces trois phases ne sont pas sujettes aux mêmes contraintes temporelles, la vitesse d'exécution de certaines tâches est un facteur plus important que pour d'autres (fig. 3.3). En effet, le pré-traitement est réalisé une fois par jeu de données : il n'est pas crucial que son exécution soit rapide. Cette phase conditionne le temps nécessaire avant que les données puissent être servies. La phase d'initialisation peut se permettre quelques lenteurs, elle n'est exécutée qu'une fois par client. Cette phase détermine le temps nécessaire avant que le client puisse télécharger des données. La vitesse d'exécution de la phase de transfert est cruciale. Les opérations de cette phase vont généralement être exécutées de nombreuses fois pour chaque client, et sa vitesse va conditionner la réactivité de l'application du client, le temps avant qu'une nouvelle partie de la scène soit chargée.

La fig. 3.4 montre comment les données sont transformées lors des différentes phases. À partir des données initiales, on calcule les données qui seront stockées sur le serveur. La description de ces données est obtenue après la phase d'initialisation et la phase de transfert permet de construire les données transférées.

Les opérations réalisées au sein des phases d'initialisation et de transfert peuvent être déportées dans la phase de pré-traitement, de telle sorte que parmi les données stockées, on trouve la description des données et les données transférées. Cela assure un temps de réponse plus bas pour les requêtes du client, au détriment de la flexibilité du serveur : le serveur est statique et ne peut pas servir d'autres données que celles qui ont été pré-calculées. Au cours de ma thèse, je me suis beaucoup

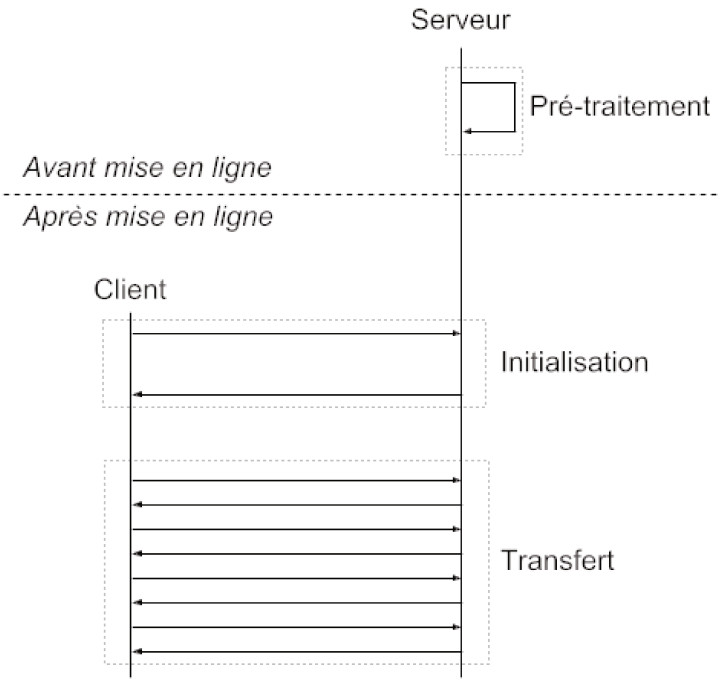

Figure 3.3 Diagramme de séquence illustrant la temporalité des différentes phases identifiées. La phase de pré-traitement est réalisée avant la mise en ligne du serveur (ou du service). Pour chaque client, le serveur répond à une requête d'initialisation contre de multiples requêtes durant la phase de transfert.

intéressé à la manière dont un compromis peut être créé entre le pré-calcul d'un côté, et le traitement à la volée de l'autre, afin d'obtenir une vitesse d'exécution correcte et de conserver au maximum l'aspect dynamique du serveur. Cela revient à se poser la question de quelles opérations peuvent être conservées dans les phases d'initialisation et de transfert sans impacter significativement les performances.

Nous verrons après la présentation des outils logiciels utilisés côté serveur deux stratégies différentes de conception de cette architecture que j'ai implémentées.

3.2.2.2 Base de données

PostGIS est une extension du système de gestion de base de données PostgreSQL permettant la gestion de données géospatiales. Plus précisément, les objets gérés par PostGIS sont ceux définis dans le standard *Simple Features* [Her11] [Her10] de l'OGC et de l'ISO TC/211. PostGIS offre un éventail de fonctionnalités particulièrement utiles pour le traitement d'objets géographiques.

Tout d'abord, PostGIS offre un système d'indexation des géométries basé sur les R-Trees [Gut84] qui permet d'accélérer significativement les requêtes spatiales.

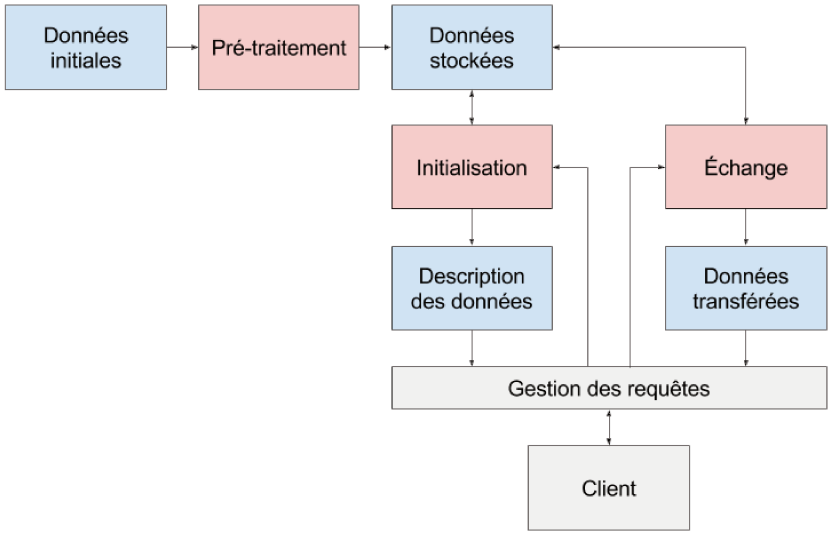

Figure 3.4 Schéma représentant les différentes phases que les données traverses dans l'architecture du serveur utilisée. En rouge sont représentées les phases, en bleu les données.

De nombreuses opérations géométriques sont disponibles dans PostGIS, parmi lesquelles on trouve notamment :

- des tests d'intersection ;
- des opérations ensemblistes comme l'union et l'intersection ;
- des translations, rotations et projections dans d'autres référentiels ;
- des opérations permettant la création de nouvelles géométries, comme l'extrusion ou la création de *buffers* (zones tampons autour d'un point, d'une ligne ou d'un autre type de géométrie) ;
- des traitements de géométries algébriques, tels que la triangulation de Delaunay, le calcul d'enveloppes ou la simplification de géométries.

PostGIS est un outil très puissant pour le traitement de géométries 2D. En revanche, le support des géométries 3D est plus limité, de nombreuses fonctions de PostGIS n'étant disponibles qu'en 2D. Pour étendre ses capacités dans le cas 3D, PostGIS peut-être compilé avec le support de la bibliothèque SFCGAL[1], un adaptateur de la bibliothèque de géométrie algébrique CGAL[2] qui lui ajoute le support des *Simple Features*.

Contrairement aux autres outils présentés dans cette section, PostGIS a été utilisée au cours de cette thèse sans apport de ma part au code source.

PostgreSQL et PostGIS ont été utilisés au cours de cette thèse mais ne sont pas essentiels aux méthodes présentées dans ce manuscrit. D'autres systèmes de gestion

[1] http://sfcgal.org
[2] https://www.cgal.org/

de base de données peuvent être utilisés comme SQLite et son extension SpatiaLite. Il serait également intéressant d'étudier des bases de données qui s'éloignent du modèle classique des bases relationnelles, par exemple MongoDB, pour évaluer la pertinence de cet autre paradigme dans le cas de la visualisation de données géographiques. À noter cependant que ces deux options ont pour le moment de sévères limitations : SpatiaLite ne supporte pas la 3D et MongoDB ne gère pas l'aspect géographique.

3.2.2.3 py3dtiles

Pour convertir les données issues de la base de données dans le format d'échange et de visualisation 3D Tiles, j'ai participé au développement du module python py3dtiles[3] (fig. 3.5) en collaboration avec Oslandia.

Comme nous l'avons vu dans la section 2.3.1, 3D Tiles est composé de deux parties : un ensemble de formats de tuiles qui décrivent différents modèles de données (*Batched 3D Models* pour les maillages, *Points* pour les points, etc.) et un *tileset* qui permet de décrire un graphe de scène (note : les termes anglophones sont conservés pour les termes directement issus des standards). La principale fonctionnalité de py3dtiles est de convertir des données dans les formats de tuiles 3D Tiles. En outre, py3dtiles implémente l'algorithme de génération de tileset présenté en section 4.4.

Pour s'interfacer facilement avec les systèmes de gestion de base de données géospatiale, py3dtiles est capable de lire des fichiers WKB (*Well-Known Binary*), qui décrivent les *Simple Features*.

3.2.2.4 Stratégies

L'architecture serveur peut être implémentée de différentes manières selon la répartition des tâches entre les différentes phases. Dans cette section, je présente deux implémentations possibles que j'ai réalisées dont je comparerais ensuite les avantages.

Stratégie basée fichiers Cette première stratégie (fig. 3.6) concentre la totalité de ses opérations dans la phase de pré-traitement. Les données initiales sont transformées en un ensemble de fichiers, qui représentent un ensemble d'objets géographiques regroupés dans une tuile. La structure des données est également stockée sous la forme d'un fichier.

[3] https://github.com/Oslandia/py3dtiles/

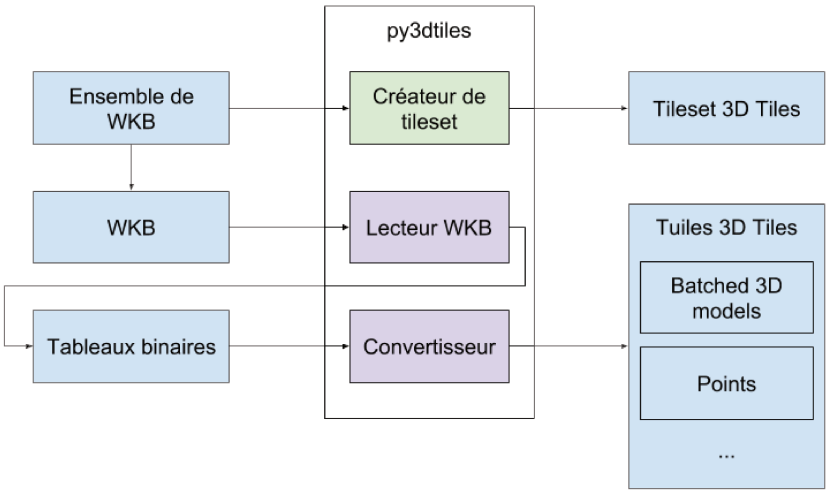

Figure 3.5 Les différentes fonctionnalités de py3dtiles. Le cœur de py3dtiles est la conversion de tableaux binaires en tuiles 3D Tiles. En outre, pour simplifier son utilisation avec des données formatées selon le standard répandu WKB, un utilitaire (lecteur WKB) permet de les transformer en tableaux binaires. Enfin, py3dtiles est capable d'organiser un ensemble d'objets dans un tileset 3D Tiles.

Stratégie basée base de données Cette stratégie (fig. 3.7) suggère de stocker et d'organiser les données au sein d'une base de données géospatiale. Les données ne sont pas stockées dans un format prêt à l'emploi, il faut qu'elles subissent des traitements avant d'être envoyées au client. Cette stratégie comporte tout de même une phase de pré-traitement permettant d'organiser les données dans un ensemble de tuiles structurées. L'idée est de trouver un compromis entre une solution base de données « pure » (sans pré-traitement) dont les phases de transfert et d'initialisation peuvent être excessivement longues et une solution analogue à la méthode basée fichier qui ne dispose d'aucune flexibilité. La manière d'organiser les données dans la base est l'objet du chapitre 4.

Le graphe de scène et les données à visualiser ne sont pas stockés dans leurs représentations finales, mais construites en fonction des requêtes de l'utilisateur. Il est intéressant de se demander comment stocker les données de telle manière que la transformation en descriptions des données et données à transférer soit le plus rapide possible.

Il est possible de passer de cette méthode à la méthode basée fichier en procédant à une mise en cache, c'est-à-dire en pré-calculant pour une requête initiale particulière la totalité des tuiles de la scène, voire d'avoir une stratégie hybride où un cache est maintenu pour les requêtes, de telle sorte qu'un élément souvent demandé ne soit pas constamment recalculé.

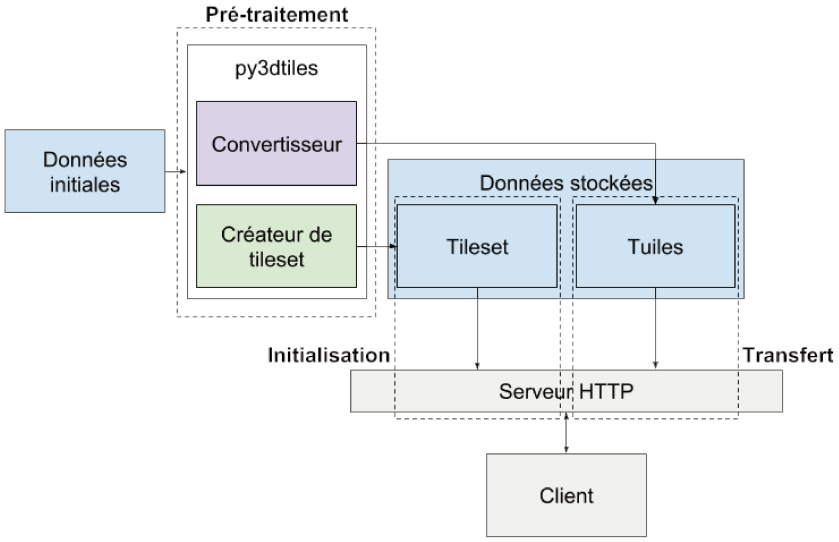

Figure 3.6 La stratégie basée fichiers de l'architecture serveur. L'ensemble des données nécessaires à la visualisation de la scène sont générées lors du pré-traitement avec l'utilitaire py3dtiles. Les phases d'initialisation et de transfert se résument à chercher le bon fichier sur le disque.

Comparaison Ces deux stratégies peuvent être comparés vis-à-vis d'un ensemble de critères étudiés ci-après : le temps de mise en ligne, le temps de traitement des requêtes, le passage à l'échelle, l'édition, l'interaction et la complexité.

Temps de mise en ligne La stratégie basée fichiers nécessite de calculer l'ensemble des tuiles du jeu de données, ce qui demande un temps de calcul conséquent. La stratégie basée base de données est plus rapide pour cette tâche, son format de stockage étant plus proche du format des données originales.

Temps de traitement des requêtes La stratégie basée fichiers à un temps de traitement quasi-nul : il suffit d'aller chercher le fichier à l'emplacement demandé. Utiliser la stratégie basée base de données nécessite de générer la tuile à partir des données de la base, ce qui demande un temps de calcul non-négligeable.

Passage à l'échelle Ce critère fait référence au passage à l'échelle du nombre de client du serveur. Les deux options permettent de paralléliser ou de multiplier les serveurs pour faire face à la charge, mais la stratégie basée fichiers a l'avantage sur la stratégie basée base de données : n'ayant pas de calcul à effectuer pour envoyer une tuile, chaque serveur peut bien plus facilement gérer un plus grand nombre de clients et la mise à l'échelle nécessitera donc un investissement moindre.

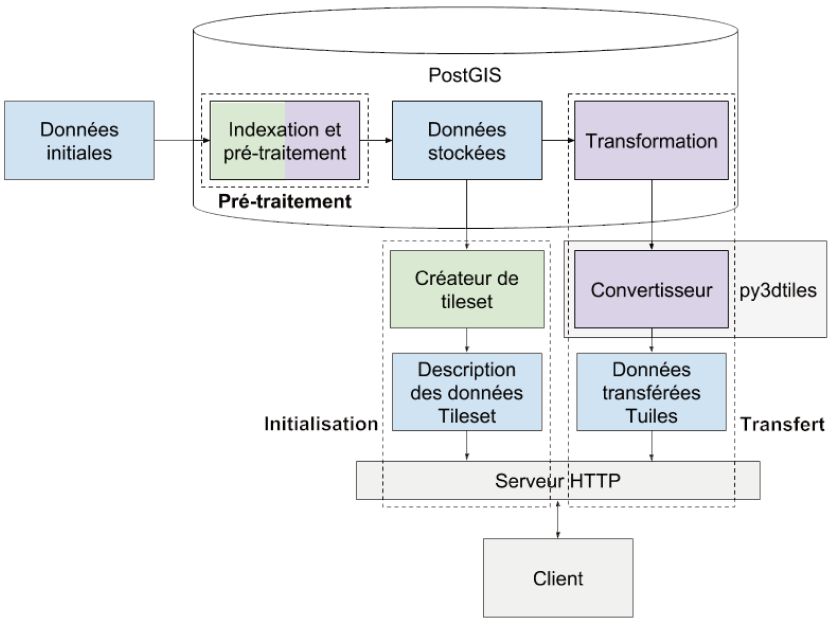

Figure 3.7 La stratégie basée base de données de l'architecture serveur. Les données initiales sont indexées et préparées à l'aide de PostGIS. Le graphe de scène (tileset) est générée à la volée, tout comme les tuiles qui sont préparées à l'aide de PostGIS et py3dtiles.

Édition L'édition consiste en la modification, l'ajout ou la suppression d'objets de la ville par n'importe quel outil et la réplication de ces changements sur les données stockées sur le serveur. Cette édition est bien plus facile pour la stratégie basée base de données. Le système de gestion de base de données est déjà capable de gérer des données dynamiques, contrairement à la stratégie basée fichiers dont l'édition requiert de regénérer tout ou une partie du jeu de données.

Interaction La stratégie basée fichiers n'admet aucune interaction avec le client : le jeu de données proposé est statique, immuable. La stratégie basée base de données propose au contraire un grand nombre d'opérations permettant de transformer les données avant leur envoie au client.

Complexité La mise en place et le développement de chaque solution n'est pas le même. La stratégie basée fichiers nécessite uniquement un serveur capable de servir des fichiers. La stratégie basée base de données nécessite l'installation d'une base de données ainsi que l'implémentation d'un code serveur faisant l'interface entre le client et la base de données, et potentiellement l'implémentation de processus plus complexes pour l'édition ou la transformation de données à la volée.

Le tableau ci-dessous résume la comparaison des critères.

Critère	Fichiers	Base de données
Temps de mise en ligne	-	+
Temps de traitement des requêtes	+	-
Passage à l'échelle	+	-
Édition	-	+
Interaction	-	+
Complexité	+	-

Une stratégie n'est pas meilleure que l'autre. Le choix de la stratégie va dépendre des cas d'utilisation visés. Par exemple, une application permettant la visualisation d'un jeu de données statique gagnera probablement à utiliser la stratégie basée fichiers, surtout si le nombre de clients est élevé. Une application de visualisation de données dynamiques nécessitera plutôt une solution basée base de données.

3.3 Client web

Dans cette section, je présente le choix du client web utilisé au cours de ma thèse, ainsi que mes contributions dans son développement et une méthode de projection à la volée.

3.3.1 Clients web considérés

Au commencement de ma thèse, plusieurs clients web avaient déjà été développées pour visualiser des données géographiques en 3D. Deux critères ont permis de choisir le projet le plus adapté aux problématiques de ma thèse et d'Oslandia : l'adéquation technique et l'ouverture du projet.

L'adéquation technique est mesurée selon les fonctionnalités qu'offre chaque projet. Les fonctionnalités qui nous intéressaient étaient la possibilité de travailler dans un repère local (un repère projeté, par opposition au repère global), le support des protocoles standards et les performances.

L'ouverture signifie que le projet devait être Open Source, ouvert aux contributions et, idéalement, avoir une communauté active. Du fait de cette exigence, les solutions comme Google Maps ou ArcGIS n'ont pas été considérées.

Urban Data Viewer Ce projet est un prototype interne au laboratoire LIRIS, développé pendant et à la suite de [GM12]. Urban Data Viewer (UDV) est basé sur la bibliothèque Three.js, un moteur de rendu 3D JavaScript très populaire, ce qui lui assure une certaine fiabilité. UDV gère les données sur un plan, mais ne gère pas les grandes coordonnées et nécessite donc de transformer les données au préalable. En

tant que prototype récent et développé avec peu de moyens, UDV possède peu de fonctionnalités et gère très peu de standards. Un avantage cependant est le contrôle total que nous avons sur son développement, mais l'absence de communauté autour du projet en fait une option peu attractive.

Cesium Cesium est un projet très mature. Il supporte de nombreux protocoles, dispose d'une grande communauté et est activement développé. Néanmoins, au début de ma thèse, il était très gourmand en ressources processeur. Cesium est un globe virtuel qui n'offre pas la possibilité de travailler dans un repère local : bien qu'une vue planaire existe, elle résulte d'une projection à la volée gérée par le moteur de Cesium, les données doivent tout de même être dans le repère global. Cesium est simple à utiliser tant qu'on reste dans le cadre prévu, mais est très complexe à manipuler si des fonctionnalités manquent et qu'il faut en modifier le code. Une dernière réserve est que, bien que le projet soit Open Source, la direction est totalement contrôlée par AGI, le développeur original de Cesium.

Malgré ces défaut, c'est la solution qui a été choisie avant la sortie d'iTowns en Open Source, d'autant plus qu'Oslandia avait déjà réalisé un prototype utilisant Cesium[4]. L'absence de mode planaire m'a amené a développer une méthode de projection à la volée présentée en section 3.3.3.

iTowns iTowns était un projet initialement interne à l'IGN permettant la visualisation d'une variété de données (maillages texturés, points, images orientées). Dans le cadre de la refonte du géoportail, le projet a été repris de zéro et publié en Open Source avec l'aide d'Oslandia. Comme UDV, iTowns se base sur Three.js. Il permet de gérer les données dans un repère local comme global. Bien que le projet soit jeune et ne soit pas aussi riche que Cesium, il est conçu pour être plus flexible et simple à modifier. L'équipe d'Oslandia fait partie de l'équipe de développement d'iTowns ce qui permet de contribuer bien plus facilement au code que sur un projet comme Cesium. Au final, iTowns est, sur le long terme, le projet qui convient le mieux à mes besoins et ceux de l'entreprise.

3.3.2 iTowns

iTowns est capable de charger des données de natures très différentes, comme des terrains issus de cartes de hauteur, des maillages, des nuages de points ou des images panoramiques (fig. 3.8). Pour cela, une architecture permettant d'ajouter facilement le support de nouveaux types de données et protocoles a été conçue. Cette architecture est le fruit de l'effort commun des contributeurs[5] du projet. Outre

[4] https://github.com/Oslandia/cesium-buildings/
[5] https://github.com/iTowns/itowns/blob/master/CONTRIBUTORS.md

mes contributions à cette architecture, mes travaux de thèses ont été l'occasion de corriger des bugs et d'ajouter le support de données 3D Tiles.

Figure 3.8 Images d'iTowns affichant différents types de données. De gauche à droite et de haut en bas : un terrain construit à partir de cartes de hauteurs, un maillage, un nuage de points et une image panoramique.

Ma description d'iTowns se fera en deux temps. Dans un premier temps, je présenterai l'architecture et les différents modules d'iTowns. Dans un second temps, je m'intéresserai aux mécaniques de mise à jour et de chargements de données.

Architecture iTowns est constitué d'un ensemble de modules qui seront présentés dans la suite de cette section. Les modules conservent leur nom anglais original pour que ce texte puisse servir comme outil de compréhension du code. La fig. 3.9 propose une vue d'ensemble de l'architecture d'iTowns.

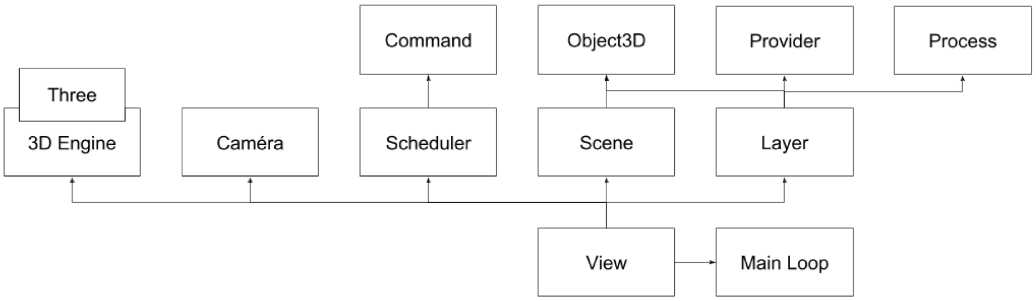

Figure 3.9 Les modules qui composent iTowns.

Object3D *Object3D*, ou objet 3D, est une classe de Three.js, la bibliothèque de rendu 3D utilisée dans le projet, qui défini un objet qui peut être affiché par Three.js.

View La *view* est une classe qui regroupe les modules nécessaires pour la visualisation et la mise à jour de la scène : la scène elle-même, la caméra, le *scheduler*, la liste des *layers*, le *3D engine* et la *Main loop*.

Main loop Cette boucle principale régit le fonctionnement général d'iTowns. C'est elle qui va demander à mettre à jour les objets ou les rendre à l'écran.

3D engine Le moteur 3D fait l'interface entre iTowns et Three.js et permet ainsi d'afficher la scène à l'écran, comme vue depuis la caméra.

Scene La scène regroupe dans un graphe de scène l'ensemble des objets 3D actuellement chargés dans iTowns. Tous les objets 3D contenus dans les *layers* sont dans la scène.

Camera La caméra permet de déterminer le point de vue de l'utilisateur et quels objets sont visibles.

Scheduler Le *scheduler* est un ordonnanceur qui permet de limiter le nombre de *commands* asynchrones exécutées parallèlement et de les prioriser.

Layer Un *layer* est une couche de données. Elle comprend un *provider*, un ensemble de *process* et les objets 3D lui correspondant.

Process Un *process* est un processus de traitement d'un *layer* qui permet de définir quels objets 3D sont visibles, s'il faut charger de nouveaux objets ou au contraire en décharger. Le *process* génère des *commands* lorsque de nouveaux objets doivent être créés. *Process* a plusieurs implémentations en fonction du types de *layer*.

Command Une commande contient toutes les informations nécessaires pour qu'un *provider* puisse télécharger et créer de nouveaux objets, tels que voulus par le *process*.

Provider Un *provider* permet la génération de nouveaux objets 3D à partir d'une commande, qui seront rajoutés au *layer* dont le *process* lié est à l'origine de la commande. Généralement, les données nécessaires à la création de l'objet 3D sont téléchargées sur des serveurs distants. Comme *process*, *provider* a plusieurs implémentations différentes, permettant de gérer différents protocoles et types de données.

Fonctionnement Deux mécanismes se déroulent en parallèle dans iTowns. Tout d'abord, il y a la boucle principale, qui met à jour les différents *layers*, génère de nouvelles commandes et affiche la scène mise à jour à l'écran. Ensuite, il y a

l'exécution des commandes générés par les *providers* qui engendrent la création de nouveaux objets 3D.

Boucle de mise à jour La *main loop* va successivement appeler pour chacun des *layers* les fonctions de mise à jour. Pour pouvoir gérer n'importe quel type de données et de structures de données, la manière dont les objets 3D sont parcourus et sont traités est déléguée au *process*. En pratique, un *process* doit implémenter une fonction *preUpdate*, *update* et éventuellement *postUpdate*. Les fonctions *preUpdate* et *postUpdate* sont exécutées une fois par *layer* (respectivement au début et à la fin de la phase de mise à jour) et la fonction *update* est appelée pour chaque objet du *layer* à mettre à jour. La fonction *preUpdate* retourne les premiers objets sur lesquels appeler la fonction *update* (dans le cas d'un arbre, ce serait la racine), puis *update* retourne les éventuels objets qu'il faut aussi mettre à jour (toujours dans le cas d'un arbre, les fils du nœud mis à jour).

La fonction *update* peut implémenter n'importe quel comportement, mais quelques actions particulières sont généralement attendues : déterminer si un objet est visible à l'aide de la caméra (et le cacher ou l'afficher selon le résultat), déterminer si un objet doit être raffiné (et créer de nouvelles commandes pour charger de nouvelles données) et décharger les objets qui ne sont plus visibles. Les commandes créées sont envoyées au *scheduler* pour une exécution différée.

À la fin de cette phase de mise à jour, le moteur 3D effectue le rendu de la scène, soit le rendu de l'ensemble des objets des *layers* dont la visibilité a été confirmée par le processus de mise à jour.

Exécution des commandes En parallèle de cette boucle principale, les commandes crées par les *processes* sont exécutées. Le *scheduler* décide en fonction de la priorité de chaque commande lesquelles il faut exécuter en premier. Lorsqu'une commande est exécutée, le *provider* lié à cette commande fait une requête au serveur contenant les données voulues, et lors de la réception de ces données, construit l'objet 3D correspondant et l'ajoute au *layer*.

3.3.3 Projection à la volée

Lors de la visualisation de données dans un système de projection particulier, il peut arriver que toutes les données importées ne soient pas dans ce système. Il faut alors reprojeter ces données afin que toutes géométries soient positionnées convenablement les unes par rapport aux autres. Ces opérations sont complexes : elles font intervenir des opérations trigonométriques sur des nombres très précis,

rendant leur parallélisation sur GPU difficile (les cartes graphiques grand public ne faisant des opérations que sur des nombres flottants de 32 bits). De plus, le nombre de points sur lesquels il faut appliquer cette transformation est souvent très grand.

Dans le cas où il n'est pas souhaitable de modifier les données du serveur (par exemple, si plusieurs applications utilisent les mêmes données et peuvent les modifier), il faut avoir la possibilité de projeter les géométries à la volée. Comme l'opération est coûteuse, il paraît intéressant de trouver une approximation rapide à calculer dont la précision est suffisante pour des applications de visualisation.

Le cas étudié ici est la transformation de géométries d'un repère planaire (repère local) vers un repère géocentrique (repère globale)[6]. Le postulat duquel je pars dans la conception de cette approximation est que l'échelle à laquelle se fait la visualisation, la courbure de la Terre est négligeable. Cela signifie que cette méthode d'approximation n'est fonctionnelle que jusqu'à une certaine taille d'objet (qui, mesurée expérimentalement dans la suite de cette section, s'avère être de plusieurs centaines de mètres). Si la courbure est négligeable, on peut alors considérer que les géométries se trouvent sur un plan (fig. 3.10). La projection devient donc une transformation d'un plan à un autre, qui peut s'exprimer sous la forme d'une matrice de transformation.

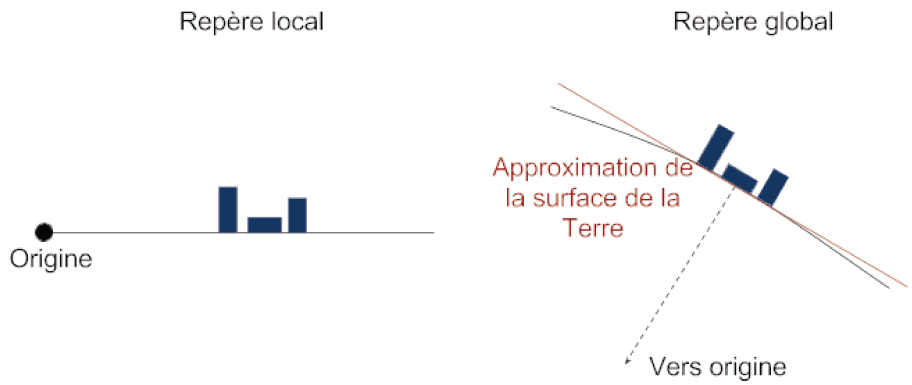

Figure 3.10 Illustration de l'approximation réalisée lors de notre projection. En bleu, les objets à projeter d'un repère local au repère global. Le plan rouge est une approximation locale de la surface très légèrement courbée de la planète.

L'attrait de cette approche, d'approximer la projection par une matrice de transformation, est qu'elle s'insère parfaitement dans le pipeline de rendu déjà existant. Il suffit de multiplier l'ancienne matrice de transformation des géométries par cette nouvelle matrice pour que les objets soient convenablement rendus à l'écran.

[6] La méthode présentée ici a été développée en début de thèse, lorsque j'utilisais Cesium. Cesium ne proposant pas un mode planaire, projeter les données d'un repère local vers le repère global était une étape obligatoire avant de pouvoir visualiser ces données. Cette méthode pourrait aussi être utilisée pour le mode globe d'iTowns.

Nous cherchons donc T, la matrice de transformation permettant la transformation d'un plan du repère local P_l au plan P_g approximant la surface du globe. Comme dit précédemment, cette approximation n'est valable que dans une zone réduite, une portion de plan que j'appelle tuile. t_l est la tuile dans le repère local, t_g celle du repère global. On peut définir t_l par trois de ses sommets, les points p_1, p_2 et p_3, et t_g par les trois points p_4, p_5 et p_6 (fig. 3.11) tels que :

$$\begin{cases} p_1 * T = p_4 \\ p_2 * T = p_5 \\ p_3 * T = p_6 \end{cases} \tag{3.1}$$

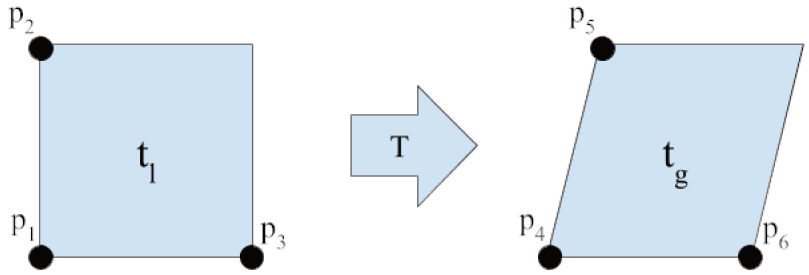

Figure 3.11 Une tuile t_l dans le repère locale transformée en tuile t_g dans le repère globale à l'aide d'une matrice de transformation T définie à l'aide des points p_1, p_2, p_3 de t_l et p_4, p_5, p_6 de t_g.

Connaissant p_1, p_2 et p_3, on peut calculer leurs projections exactes respectives p_4, p_5 et p_6, et résoudre le système linaire 3.1, obtenant ainsi T.

Résultats Un test de cette approximation a été réalisée sur des données de Lyon (fig. 3.12). Les données originales étaient dans le repère EPSG :3946 (projection conique conforme), le référentiel de l'application était le repère WGS84 (repère géocentrique). Fig. 3.13 montre l'erreur maximale mesurée due à l'approximation en fonction de la longueur du côté d'une tuile carrée. L'erreur croît au carré de la longueur de côté. On constate qu'avec une tuile de 500 mètres de côté on est bien en dessous du centimètre d'erreur, ce qui est inférieur à la précision de la plupart des relevés de données urbaines (comme nous l'avons vu en introduction).

Ces résultats nous montrent aussi qu'il faut limiter la taille des tuiles, et donc qu'il est nécessaire d'en utiliser plusieurs pour recouvrir toute la ville. L'économie de calcul reste substantielle : au lieu d'une projection par point des géométries, il suffit de trois projections et d'une multiplication de matrice par tuile.

Limites Cette méthode a cependant des limites. La plus évidente est celle de la taille des tuiles qui, une fois qu'elles dépassent un certain seuil, génèrent des erreurs visibles. Cela pose problème pour les organisation hiérarchique de données, où les

Figure 3.12 Un quartier de Lyon dont la projection dans le repère global a été approximée grâce à ma méthode.

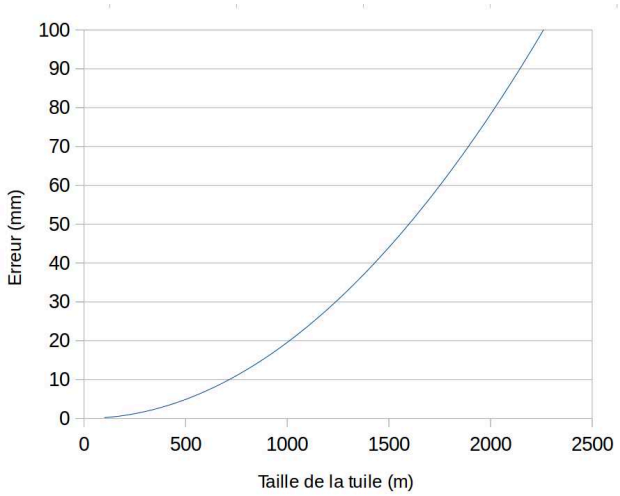

Figure 3.13 Erreur constatée en fonction de la taille de la tuile utilisée pour calculer l'approximation de la projection.

tuiles au sommet de la hiérarchie sont nécessairement très grandes. Ce type de structure est décrite en détail dans le chapitre 4.

Un autre problème est qu'il n'y a pas de continuité entre deux tuiles adjacentes. Si deux objets sont supposés être continus d'une tuile à l'autre (une route par exemple), leur raccordement ne sera pas parfait, et il est possible que l'utilisateur perçoive un léger décalage entre les deux géométries.

4 Structuration de données géospatiales

Sommaire

La forte volumétrie des données géospatiales requiert de prêter particulièrement attention à leur organisation. En effet, il est rarement envisageable de charger l'ensemble d'un jeu de données, d'autant plus dans un contexte web où les performances sont limitées et les données doivent transiter par le réseau. En structurant les données, il devient possible de charger uniquement, ou tout du moins majoritairement, celles qui sont pertinentes à un instant donné, d'offrir une gestion du niveau de détail et de permettre la progressivité du chargement.

Les données géospatiales sont de natures variées. Ainsi, dans un but de généricité, il est préférable d'éviter les méthodes reposant sur les spécificités de certains types de données, comme par exemple les maillages progressifs.

En outre, utiliser des structures qui sont compatibles avec les standards du domaine est particulièrement important pour permettre l'interopérabilité des solutions proposées.

Dans ce chapitre, je commence par contextualiser mes recherches en présentant la méthode proposée au tout début de ma thèse pour charger des données 3D, basée sur le protocole WFS. Puis, après avoir introduit plus en détail le cadre donné par l'utilisation du format 3D Tiles, je présente trois méthodes conçues au cours de ma thèse pour organiser des données géospatiales dans une structure de données. La première répartit les données dans une grille régulière, découpant les objets qui s'étendent sur plusieurs tuiles, la deuxième organise les données dans un quadtree et permet l'édition rapide des données, et la dernière place les géométries dans une structure hiérarchique basée sur un découpage sémantique de l'espace. Le chapitre se conclut par une synthèse comparant les différentes méthodes.

4.1 Limites du protocole WFS

Au début de ma thèse, la principale méthode utilisée (notamment dans les premiers prototypes d'Oslandia) pour charger des données 3D en utilisant des standards se basait sur le protocole WFS [Vre14], traditionnellement limité aux données 2D. Une grille était définie par le client, et dès qu'une tuile de la grille entrait dans le champ de vision de la caméra, une requête WFS était envoyée au serveur pour récupérer, puis afficher, les données 3D de cette tuile. Cette méthode est assez peu efficace, pour un ensemble de raisons que je vais détailler.

Vitesse Tout d'abord, le protocole WFS utilisé est un protocole non-tuilé, c'est-à-dire que le client requête une zone qu'il passe en paramètre et pas un identifiant de tuile. En conséquence, le temps de sélection des objets par le serveur est beaucoup plus lent : au lieu de récupérer quasi-instantanément les objets liés à l'index d'une

tuile, le serveur doit faire des intersections pour déterminer quelles géométries se trouvent dans la boîte englobante donnée par le client.

Redondance Deuxièmement, si on cherche à charger autre chose que des points, il y a un risque que ces géométries débordent de la tuile. Le protocole WFS offre deux options pour gérer ce cas là : soit charger toutes les géométries qui intersectent la zone demandée, soit charger toutes les géométries dont le centroïde se trouve à l'intérieur de la zone demandée. Dans le premier cas, il y a un soucis de redondance : une géométrie qui intersecte deux tuiles sera chargée deux fois, il faut donc filtrer les données reçues et gérer le fait qu'une géométrie appartient à plusieurs tuiles différentes (pour éviter qu'en déchargeant une tuile, on supprime des géométries qui ne devrait pas l'être). Même si cela est bien géré, il y a une perte d'efficacité, vu que certains objets transitent plusieurs fois sur le réseau. Dans le second cas, il n'y a pas de problème de redondance, mais un problème de visibilité des bâtiments appartenant à plusieurs tuiles : pour charger un objet, il faut impérativement que la tuile où son centroïde se situe ait été chargé. Ces deux problèmes sont illustrés par la fig. 4.1.

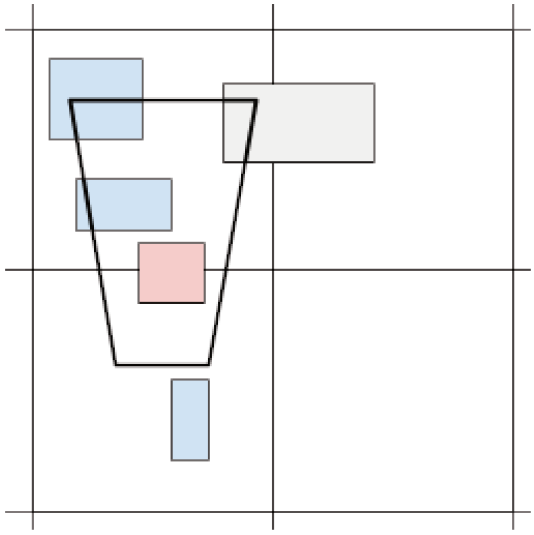

Figure 4.1 Illustration des problèmes de visibilité survenant avec le protocole WFS. Avec la sélection par centroïde, l'objet grisé ne sera pas affiché, bien qu'il se trouve dans le champ de vision de la caméra (trapèze), car la caméra ne voit pas la tuile de la grille auquel l'objet grisé appartient. Avec la sélection par intersection, l'objet rouge sera chargé deux fois, car les deux tuiles qu'il intersecte sont visibles par la caméra.

Niveau de détail Finalement, il n'y a pas de notion de niveau de détail dans le protocole WFS. En outre, quelle que soit la taille de la boîte englobante demandée, toutes les données situées à l'intérieur de cette boîte seront chargées, ce qui pose problème pour une vue d'ensemble d'une ville. À noter qu'il existe une proposition récente[1] qui s'attaque à ce problème en ajoutant la possibilité de spécifier un niveau de zoom lors des requêtes WFS.

[1] http://ogc.standardstracker.org/show_request.cgi?id=514

Ces problèmes illustrent bien l'insuffisance de la méthode existant au début de ma thèse, et la nécessité de développer de nouvelles structures de données et protocoles pour les données 3D.

4.2 Tileset 3D Tiles

J'ai évoqué en introduction l'importance de travailler avec des formats standards pour permettre l'interopérabilité des serveurs et des clients. Cela se reflète notamment dans mes travaux par l'utilisation du futur[2] *community standard* 3D Tiles déjà présenté précédemment. La spécification de ce standard limite ce qui peut être fait dans les méthodes que je développe, il est donc utile de regarder en détail quelles structures et mécaniques peuvent être décrites par ce format. Pour rappel, la spécification 3D Tiles est composée de deux parties (fig. 4.2) : les formats de données (maillages, points, etc.) et la description du tileset (littéralement *ensemble de tuiles*, qui décrit une hiérarchie spatiale). C'est cette seconde partie qui nous intéresse ici. Notons tout de même que les formats de données de 3D Tiles permettent de lier des informations sémantiques aux géométries, ce qui est une fonctionnalité nécessaire dans de nombreux cas d'utilisation.

Un tileset est la description d'un arbre en JSON. Chaque nœud de l'arbre est composé de plusieurs propriétés le définissant, dont voici les principales :

- une URL (facultative) qui pointe vers le contenu de la tuile ;
- la liste des enfants de la tuile ;
- un volume (boîte, sphère ou portion de globe) englobant le contenu de la tuile et de ses enfants ;
- une valeur d'erreur géométrique qui indique l'erreur qui subsiste si la tuile est chargée, mais pas ses enfants ;
- la méthode de *raffinement*, par ajout ou par remplacement.

Notons qu'une tuile n'a pas forcément de contenu et peut donc être juste utilisée pour structurer davantage ses tuiles filles.

Le raffinement est l'opération qui augmente le niveau de détail d'une tuile en chargeant ses enfants. Le « raffinement par ajout » ajoute le contenu des tuiles filles à la tuile à raffiner. Le « raffinement par remplacement » remplace la tuile à raffiner par ses tuiles filles.

[2] À l'heure où j'écris ce manuscrit, la procédure de validation du standard auprès de l'OGC est en cours de finalisation.

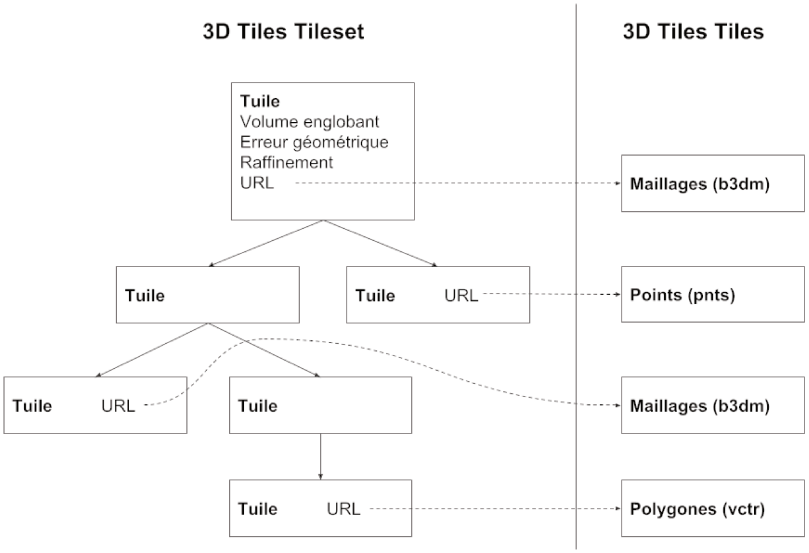

Figure 4.2 Exemple de tileset. Chaque tuile décrite par le tileset dispose d'un ensemble de propriétés, dont parfois une URL qui point vers la représentation géométrique de la tuile. Cette représentation peut prendre diverses formes, comme par exemple des maillages, des points ou des polygones.

L'erreur géométrique (présentée dans la section 2.2.3) est la métrique utilisée pour contrôler le raffinement des tuiles. Cette mesure fonctionne bien dans le cas où la méthode de généralisation utilisée pour générer les tuiles est une méthode de simplification, mais on peut questionner sa pertinence dans d'autres cas, comme la généralisation par symbolisation, où cette notion d'erreur géométrique n'a pas de sens, vu que l'on compare des objets de différentes natures.

4.3 Grille régulière

Principe Une première approche est de découper le jeu de données selon une grille 2D régulière. Ce choix a l'avantage de la simplicité et de la prédictibilité : il suffit d'avoir la connaissance d'un point de la grille et de la taille des tuiles pour pouvoir définir l'ensemble des tuiles. Cependant, les objets géographiques ne sont pas toujours contenus entièrement dans une seule tuile. Seule la partie de l'objet qui est contenue dans la tuile considérée doit être conservée en découpant les géométries le long de la grille (fig. 4.3), ce qui permet notamment d'éviter les problèmes rencontrés avec WFS.

Les motivations pour utiliser cette structure sont diverses. Elle permet d'obtenir une séparation « propre » des données : il n'y a pas de recouvrement entre les différentes tuiles, à chaque point de l'espace correspond une unique tuile. Cette structure permet également une certaine homogénéité dans les traitement des différents types de données, il s'agit dans tous les cas d'une découpe (si nécessaire)

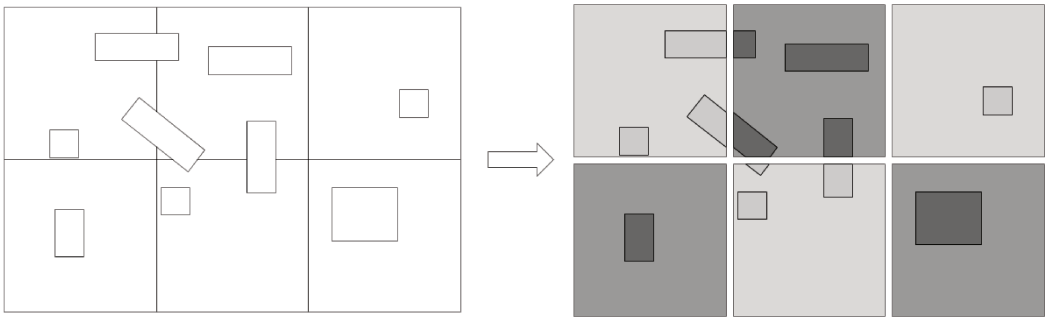

Figure 4.3 Un ensemble d'objets (représentées par des rectangles) répartis et découpés dans les tuiles d'une grille.

et d'une répartition des géométries dans les différentes tuiles. Ainsi, plusieurs couche de données de natures différentes peuvent partager la même grille, de telle sorte que les tuiles de chaque couche se superposent. Un dernier avantage notable de cette méthode est qu'elle gère sans soucis les objets anormalement grands. Leur taille n'importe pas car ils seront découpés par la grille.

La découpe est cependant problématique pour certains usages. Il y a une perte de l'unicité des objets géographiques qui se retrouvent répartis dans plusieurs tuiles. Un semblant d'unicité peut être retrouvé en marquant les géométries appartenant au même objet géographique, il demeure que l'objet aura perdu sa topologie. Cela peut, par exemple, poser problème lorsqu'un maillage fermé devient deux maillages ouverts et perd son étanchéité (ce qui est problématique pour certains algorithmes de simulation). Cette découpe est aussi une opération coûteuse en temps, d'autant plus si les données sont ensuite optimisées pour cette nouvelle découpe (recalcul des coordonnées de textures et des atlas de textures).

Application et résultats Cette méthode a été mise en œuvre à l'aide d'outils développés au sein du projet VCity[3]. La découpe a été réalisée avec le logiciel 3DUSE[4], qui implémente la découpe de fichiers CityGML en tuiles régulières. Fig. 4.4 montre une tuile découpée à l'aide de ce logiciel.

Le processus de découpe est long. Pour le troisième arrondissement de Lyon (420 bâtiments, 437 000 triangles, 6,4 km^2), la découpe en tuiles de 500 mètres de côté de la couche de bâtiments a pris 34 minutes sur un ordinateur équipée d'un Intel©Core™i5-4590 CPU @ 3.3GHz x 4.

Les données ont été séparées en plusieurs couches de données : bâtiments, terrain et nuages de points (représentés sur la fig. 4.5, gauche). Pour chaque couche de données, chaque tuile est stockée sous la forme d'un fichier, dont le nom est choisi en

[3] https://liris.cnrs.fr/vcity/
[4] https://github.com/MEPP-team/3DUSE

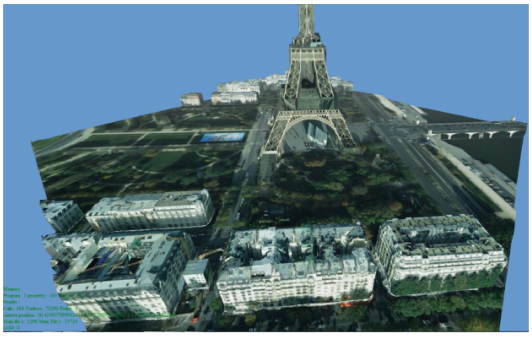

Figure 4.4 Une tuile découpée par 3DUSE. Jeu de données bati 3D de l'IGN.

fonction de ses coordonnées dans la grille. C'est la stratégie basée fichiers présentées dans le chapitre 3.

Notons également un autre aspect pratique de cette grille : elle permet de se caler exactement sur d'autres sources de données utilisant, par exemple, le protocole WFS (fig. 4.5, droite). Le protocole WFS et d'autres protocoles de l'OGC (comme WMS par exemple) permettent en effet de récupérer les données situées dans une boîte englobante 2D définie par le client, qui peut donc être définie comme étant la même que la tuile.

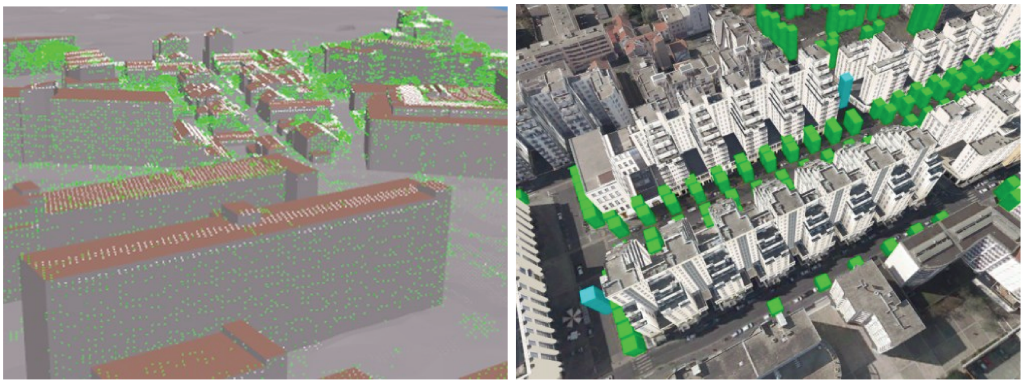

Figure 4.5 À gauche, différentes couches de données visualisées à partir d'une structure en grille. À droite, l'ajoute de données issues d'un flux WFS (arbres en vert et stations de vélos en bleu) à la maquette.

Limites Dans l'application cliente de test de cette méthode, seules les tuiles dans un voisinage défini autour de la tuile en dessous de la caméra sont chargées. Ajouter des tuiles plus éloignées cause de sérieux problèmes de performances. C'est là que nous observons les limites de cette organisation spatiale. Bien qu'elle atteigne ses objectifs de charger seulement un sous-ensemble pertinent d'un jeu de données, ce qui lui permet de gérer de hautes volumétries, il n'est pas possible d'avoir de vue d'ensemble de la ville. En effet, pour avoir une telle vue, il faudrait charger l'intégralité des tuiles, ce qui revient à charger la totalité du jeu de données ce qui est impossible avec la volumétrie envisagée. Cette structure de données est donc utilisable pour les applications de type « street view », où la caméra reste proche du

sol. Dans une application avec une vue plus libre, il faut avoir recours à d'autres structures, qui ne sont pas contraintes par une taille de tuile fixe qui force un certain niveau de détails : des structure hiérarchiques.

4.4 Hiérarchie de boîtes englobantes basée sur un quadtree

Principe Le développement de cette méthode a été articulé autour de trois objectifs : la visualisation de la ville à grande échelle, la progressivité du chargement des géométries et la possibilité d'éditer rapidement les données de la structure.

La solution employée est une hiérarchie de boîtes englobantes (ou de volumes englobants), basée sur un quadtree, qui emploie un raffinement par ajout (fig. 4.6). Ma hiérarchie de boîte englobante est un arbre dont chaque nœud est composé d'un ensemble d'objets et d'une boîte englobant tous ces objets ainsi que tous les nœuds fils. Le fait que cette hiérarchie se base sur un quadtree signifie qu'un quadtree est utilisé pour déterminer la manière dont les objets sont répartis dans les nœuds de la hiérarchie.

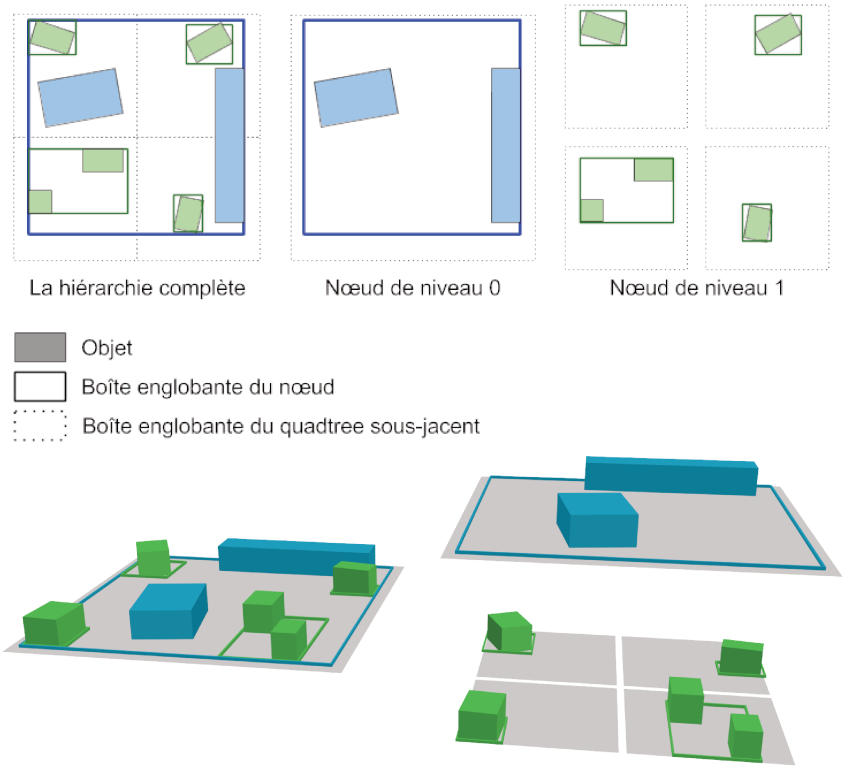

Figure 4.6 Un exemple de hiérarchie générée avec ma méthode et sa décomposition par niveaux. La vue 2D qui sera utilisée pour la suite des explications représente en réalité des objets 3D, comme ceux de l'image du bas.

Le nombre d'objets par nœud est limité. Les algorithmes présentés ici sont conçus de telle sorte que ce nombre soit fixe et choisi par l'utilisateur, afin qu'il puisse choisir le nombre le plus adapté à la précision et au type de ses données. Une variante est de limiter le nombre de primitives (triangles, points, lignes...) par nœud. Cela permet de définir une borne supérieur pour le poids des tuiles et, dans une moindre mesure, une certaine homogénéité des poids des tuiles. Cela permet à l'application cliente de gérer plus efficacement sa charge et de garantir une régularité dans le chargement des tuiles.

Le choix de cette structure se justifie pour plusieurs raisons :

- une structure hiérarchique permet de représenter des données à différentes échelles, et donc de visualiser la ville à grande échelle comme à petite échelle en ajoutant progressivement des détails ;
- un raffinement par ajout revient à faire de la généralisation par sélection (c'est-à-dire qu'on représente un ensemble d'objets par un sous-ensemble de ces objets). Ce type de généralisation a l'avantage de ne pas nécessiter la génération de nouvelles géométries, ce qui rend l'édition plus facile car il n'est pas nécessaire de recalculer de nouvelles représentations généralisées à chaque changement. La généralisation est faite en mettant les objets de plus haute importance dans les tuiles les plus haute dans la hiérarchie. La définition de cette importance sera étudiée dans la partie 5.1.2 et l'algorithme de classement dans la suite de cette section ;
- l'utilisation d'un quadtree comme base pour la construction de la structure hiérarchique permet, car c'est une structure régulière, de déterminer instantanément les tuiles dans lesquelles une géométrie se trouve, facilitant l'insertion de nouveaux objets.

Pour permettre la création et la modification de ma structure de données, trois opérations sont nécessaires : la classification (initialisation), l'insertion et la suppression d'objets (fig. 4.7). L'objectif est de pouvoir mettre à jour le jeu de données lorsqu'un changement survient. Il peut s'agir, par exemple, d'un bâtiment démoli qu'il faut supprimer de la base de données, ou des éditions fréquentes d'un projet urbain. Les algorithmes de ces différentes opérations sont présentés ci-après.

On suppose qu'à chaque objet du jeu de données est lié un poids (fixe) qui quantifie l'importance de l'objet dans la scène. La définition et la manière de calculer ce poids sont présentées dans la partie 5.1.2. Dans les exemples qui illustrent cette section, le poids est défini à partir de l'aire des géométries.

J'utilise les notations suivantes dans les algorithmes :

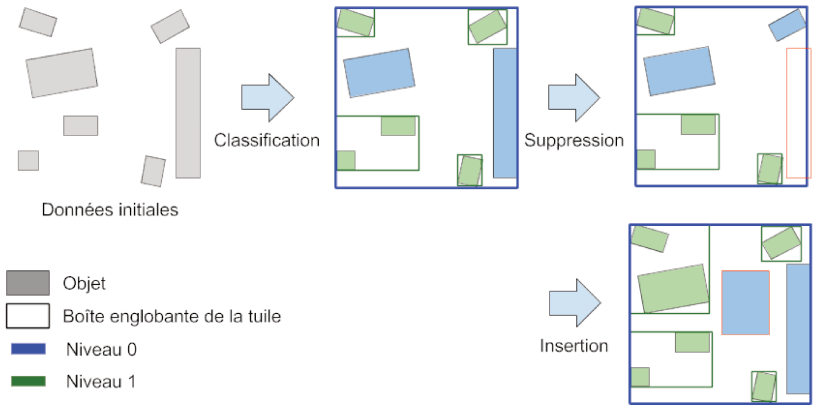

Figure 4.7 Les différentes opérations réalisables sur ma structure de données : la classification initiale des objets dans la structure et l'insertion ou la suppression d'objets. Notons que l'ajout ou la suppression d'objet peuvent entraîner la réorganisation des objets des tuiles.

- N un nœud de l'arbre ;
- f un objet géographique (*feature*) ;
- n le nombre maximum d'objets que peut contenir un nœud ;
- $B(N), B(F)$ la boîte englobante du nœud N ou de l'objet F ;
- $C(N)$ la liste des nœuds fils du nœud N ;
- $P(N)$ le nœud parent du nœud N ;
- $F(N)$ la liste des objets du nœud N ;
- F_T la liste de tous les objets géographiques ;
- $mf(N)$ l'objet de plus bas poids du nœud N ;
- $Mf(N)$ l'objet de plus haut poids du nœud N ;
- $QB(N)$ la boîte englobante de la tuile du quadtree correspondant au nœud N ;
- $W(f)$ le poids associé à l'objet F.

Les notations usuelles suivantes sont utilisées :

- $A \cup B$ union de A et B ;
- $A \setminus B$ soustraction de B à A ;
- $A \cap B$ intersection de A et B ;
- $A \in B$ A est à l'intérieur de B ;
- $A \subset B$ A est un sous-ensemble de B ;
- $\emptyset$ l'ensemble vide.

Classification Cette opération a pour but de créer la hiérarchie de boîtes englobantes et de répartir les objets dans ses nœuds. La fig. 4.8 illustre le déroulement de l'algorithme 1, détaillé ci-après.

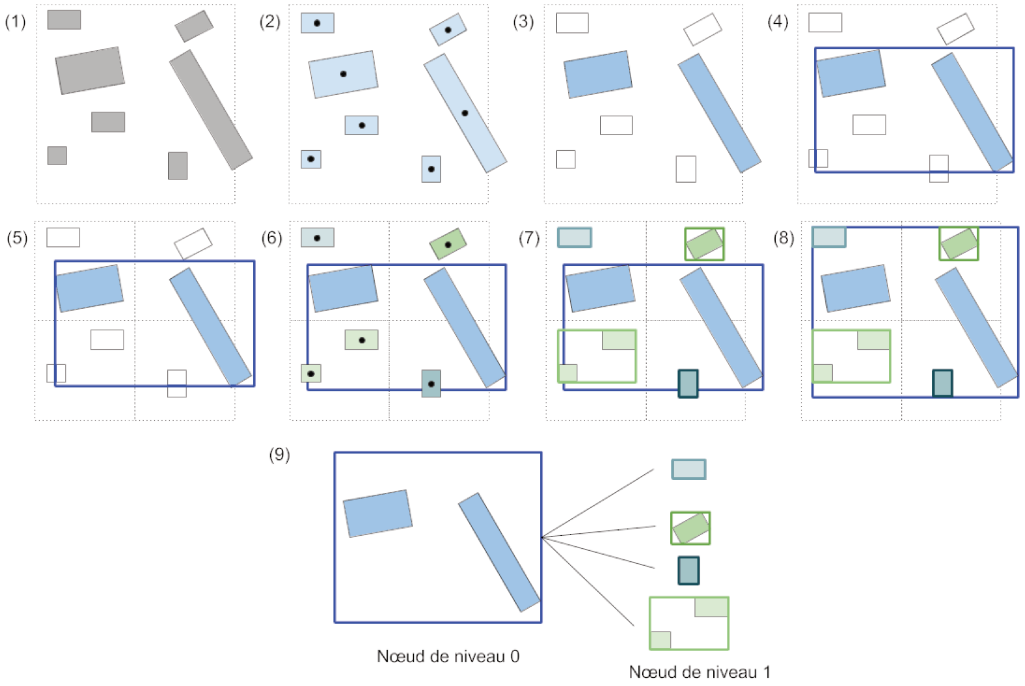

Figure 4.8 Algorithme d'initialisation pour $n = 2$. (1) État initial. Le carré en pointillé est la tuile initiale du quadtree qui couvre l'entièreté du jeu de données. (2) Sélection des objets par centroïde. (3) Assignation des objets de plus haut poids. (4) Calcul de la boîte englobante. (5-7) Récursion. (8) Correction des boîtes englobantes des parents. (9) Structure résultante.

L'initialisation (1) requiert plusieurs actions :

- la définition de l'étendue de la scène (une boîte englobante), qui servira également comme base pour la racine du quadtree sous-jacent ;
- le tri des objets par poids décroissants, de telle sorte que les objets auxquels on accède en premier sont ceux avec le plus haut poids ;
- le choix d'un nombre maximum d'objets par nœud n.

À chaque nœud de la hiérarchie de boites englobantes est associé une tuile du quadtree. Cette tuile sert à sélectionner les objets qui pourront potentiellement être assignés au nœud. Si le centroïde de l'objet considéré se trouve à l'intérieur de la tuile du quadtree, cet objet est sélectionné (2). Les n premiers objets sélectionnés sont assignés au nœud associé à la tuile du quadtree (3). Une fois ces objets assignés, la boîte englobante du nœud est calculée (4). S'il reste des objets non-assignés, la tuile du quadtree est subdivisée (5) et le processus est répété : assignation, sélection (6), calcul de boîte englobante (7) jusqu'à ce qu'il n'y ait plus d'objets non-assignés. À ce moment, les boîtes englobantes sont ajustées récursivement (8) pour que les boîtes englobantes des parents inclues les boîtes englobantes des enfants, résultant en la structure finale (9).

Algorithme 1 Algorithme de classification.

for f **in** F_T **do** ▷ Calculer tous les poids
    Calculer $W(f)$
Ordonner les objets géographiques selon un poids décroissant
Créer le nœud racine N_R
ASSIGNER(F_T,N_R)

function ASSIGNER(F, N)
    $compte \leftarrow 0$
    for f **in** F **do**
        if $f \in QB(N)$ **then**
            if $compte < n$ **then**
                $F(N) \leftarrow F(N) \cup f$
                $B(N) \leftarrow B(N) \cup B(f)$
                $F \leftarrow F \setminus f$
                $compte \leftarrow compte + 1$
            else ▷ Arrêter le processus quand le nœud est plein
                break
    if $F \neq \emptyset$ **then** ▷ Subdiviser le nœud s'il reste des objets géographiques
        DIVISER(F, N)
        for Ni **in** $C(N)$ **do** ▷ Ajuster la boîte englobante
            $B(N) \leftarrow B(N) \cup B(Ni)$

function DIVISER(F, N)
    Créer les nœuds fils de N
    for N_i **in** $C(N)$ **do** ▷ Assigner chaque objet géographique au bon fils
        $F_i \leftarrow \emptyset$
        for f **in** F **do**
            if $F \in QB(N_i)$ **then**
                $F_i \leftarrow F_i \cup f$
                $F \leftarrow F \setminus f$
        ASSIGNER(Fi, Ni)

Insertion Le processus d'insertion (algorithme 2, illustré fig. 4.9) nécessite de trouver où insérer le nouvel objet, puis de réorganiser localement les autres objets.

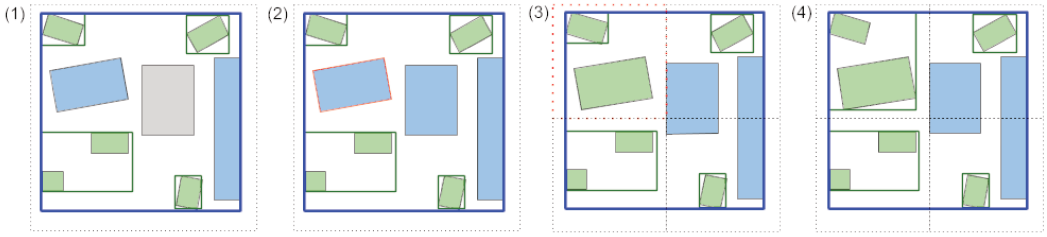

Figure 4.9 Algorithme d'insertion. (1) En gris, l'objet à ajouter. (2) Trouver l'objet de plus bas poids (contour rouge) de la tuile courante. (3) Retirer l'objet trouvé et l'ajouter au nœud fils dans lequel il se situe. (4) Recalculer les boîtes englobantes du nœud courant et de ses ancêtres.

On commence par essayer d'insérer le nouvel objet au sommet de la hiérarchie, au nœud racine (1). Si le nœud n'est pas saturé (son nombre d'objets est inférieur au nombre maximum), le nouvel objet est inséré dans le nœud. Si le nouvel objet à un poids supérieur à l'objet de plus bas poids du nœud, il remplace l'objet de plus base poids dans le nœud (2). La récursion de la procédure est faite : on essaie d'insérer l'objet de plus bas poids (celui remplacé ou celui qui était originellement en train d'être inséré selon le cas) dans le nœud fils qui correspond à sa position (3). Lorsque la récursion est terminée, les boites englobantes de tous les nœuds qui ont été modifiés ainsi que leurs ancêtres sont recalculées.

Algorithme 2 Algorithme d'insertion.

```
1:  function AJOUTER(f , N)
2:      if |F(N)| < n then
3:          F(N) ← F(N) ∪ f
4:      else
5:          if W(f) > W(mf(N)) then        ▷ Remplacer l'objet géographique de plus bas poids
6:              F(N) ← F(N) \ f
7:              F(N) ← F(N) ∪ f
8:              F ← mf(N)
9:          if C(N) = ∅ then
10:             Créer les nœuds fils de N
11:         for Ni in C(N) do              ▷ Ajouter l'objet géographique au bon fils
12:             if f ∈ QB(Ni) then
13:                 AJOUTER(f, Ni)
14:                 break ;
15:     Mettre à jour les boîtes englobantes
```

Suppression Le processus de suppression (algorithme 3, illustré fig. 4.10) requiert de rééquilibrer la hiérarchie après l'enlèvement de l'objet : tous les nœuds de la hiérarchie doivent être saturés, sauf les feuilles.

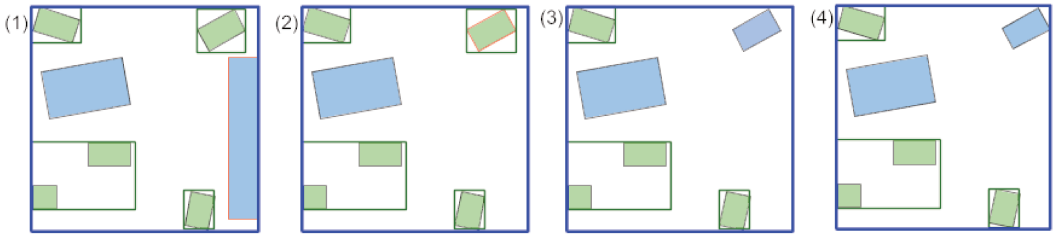

Figure 4.10 Algorithme de suppression. (1) Objet à supprimer (contour rouge). (2) Trouver l'objet de plus haut poids (contour rouge) parmi les nœuds fils. (3) Ajouter cet objet au nœud courant. (4) Recalculer les boîtes englobantes.

L'objet à supprimer est enlevé du nœud auquel il appartient (1). Si le nœud n'est pas une feuille, il faut remplacer cet objet pour que le nœud soit de nouveau saturé. On choisit pour cela l'objet de plus haut poids parmi les objets des fils du nœud (2). Le processus est appliqué au nœud fils dont l'objet de remplacement provient, et ce récursivement jusqu'à atteindre les feuilles de l'arbre (3). Enfin, les boites englobantes de tous les nœuds affectés ainsi que leurs ancêtres sont recalculés.

Algorithme 3 Algorithme de suppression.

```
1:  function SUPPRIMER(f, N)
2:      if f ⊂ F(N) then
3:          F(N) ← F(N) \ f
4:          if C(N) ≠ ∅ then
5:                  ▷ Trouver l'objet géographique de plus haut poids des nœuds fils
6:              for Ni in C(N) do
7:                  if W(MF(Ni)) > W(maxF) then
8:                      maxf ← Mf(Ni)
9:                      maxN ← Ni
10:             F(N) ← F(N) ∪ maxf
11:             SUPPRIMER(maxf, maxN)
12:         else
13:             if F(N) = ∅ then
14:                 Supprimer N
15:     else
16:         for Ni in C(N) do
17:             if f ∈ QB(Ni) then
18:                 SUPPRIMER(f, Ni)
19:                 break ;
20:     Mettre à jour les boîtes englobantes
```

Application et résultats Comme l'un des objectifs de cette méthode est de favoriser l'édition des données, celle-ci a logiquement été implémentée sur une architecture faisant usage d'une base de données. Le système de gestion de base de données PostgreSQL a été utilisé, avec son extension PostGIS pour la gestion des objets géographiques. Le reste du serveur, soit la gestion des requêtes et des opérations de classification, d'ajout et de suppression, a été codé en python. Côté client, Ce-

sium et iTowns ont été utilisés pour visualiser les données 3D, ce qui confirme l'interopérabilité de la solution.

Les résultats suivants ont été obtenus sur une machine équipée d'un Intel©Core™i5-4590 CPU @ 3.3GHz x 4 et d'une Nvidia GeForce GTX 970.

Deux jeux de données ont été utilisés pour évaluer cette méthode : les villes de Lyon (30 000 bâtiments) et de Montréal (60 000 bâtiments). La création de la structure initiale a pris respectivement 6,1 et 11,4 secondes. Les mesures des temps d'insertions et de suppressions sont indiquées dans le tableau 4.1. Ces temps sont en moyennes inférieur à 20 millisecondes, ce qui est suffisamment bas pour permettre une édition interactive des données.

	Moyenne (ms)	Écart type (ms)
Lyon		
Insertion	11.5	4.1
Suppression	10.4	3.9
Montréal		
Insertion	17.8	5.5
Suppression	16.1	5.6

Table 4.1 Performances des opérations d'insertions et de suppressions.

Fig. 4.11 montre le résultat de cette organisation des données pour la visualisation de bâtiments de la ville. Plus la distance à la caméra est grande, moins le nombre de bâtiments chargés est grand. Fig. 4.12 compare les résultat de cette méthode à une organisation en grille des données en limitant le nombre de triangles chargés simultanément. On voit clairement que ma structure hiérarchique permet d'avoir une vue généralisée de toute la ville, à la place de visualiser seulement la partie de la ville qui est proche de la caméra.

L'exploitation de données sémantiques liées est possible, comme le montre la fig. 4.13. Dans cette image, les bâtiments ont été colorés à partir du plan d'occupation du sol. Dans la base de données PostGIS, les géométries des bâtiments ont été croisées avec le plan d'occupation du sol, et un attribut décrivant leur fonction a pu être associé aux bâtiments. Cet attribut a ensuite été transmis au client qui a pu styliser les objets en conséquence.

Limites Cette méthode a plusieurs limitations. Tout d'abord, l'utilisation d'un quadtree induit des effets de grilles avec des chutes de densité de bâtiments affichés entre deux tuiles adjacentes (particulièrement visibles sur l'image de droite de la fig. 4.12 du fait de la limite imposée du nombre de géométries).

Ensuite, l'arbre généré par cette méthode peut être fortement déséquilibré (c'est-à-dire que la profondeur des feuilles de l'arbre varie grandement) en fonction de la

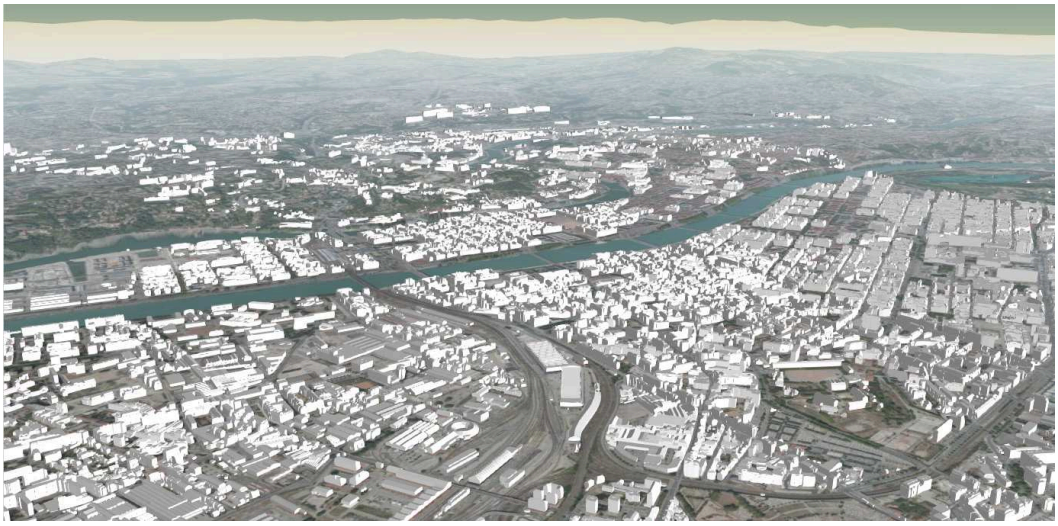

Figure 4.11 La ville de Lyon visualisée dans Cesium à l'aide de la méthode basée quadtree. On note la différence de densité des bâtiments affichés entre le premier plan et l'arrière plan.

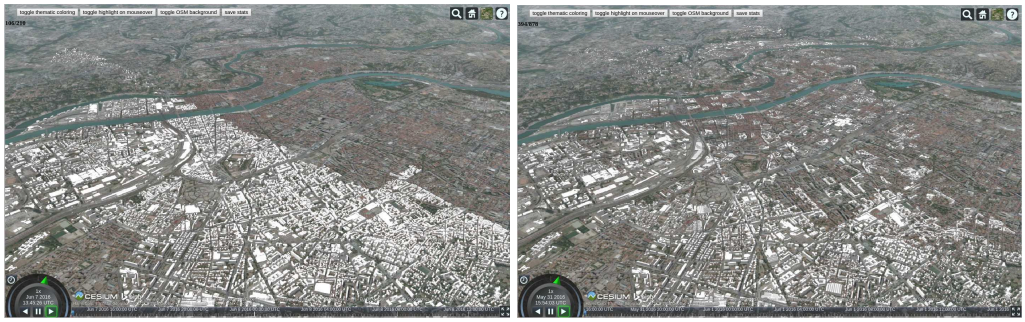

Figure 4.12 Vues de la ville de Lyon avec une limite d'un million de triangles chargés. À gauche, les bâtiments sont organisés dans une grille, à droite, dans ma structure hiérarchique. On observe les limitations de la grille, où seuls les bâtiments proches de la caméra sont chargés, alors qu'avec ma méthode, des bâtiments sont chargés dans toute la ville.

répartition des objets dans l'espace. Il faut attendre que les derniers niveaux de la hiérarchie soient chargés pour avoir une idée de la densité d'une partie de la ville. De loin, cela peut donner l'illusion que la ville est de densité uniforme. Cet arbre déséquilibré est aussi une structure sous-optimale pour certains algorithmes. En revanche, cela garantit des tuiles compactes, qui sont efficaces pour charger le moins de données superflues.

Utiliser d'autres méthodes de constructions d'arbres telles que le k-d tree par exemple diminuerait ces problèmes, mais compliquerait en contrepartie l'édition.

Support des niveaux de détails

Il est possible d'ajouter une gestion rudimentaire des niveaux de détails à la méthode précédente (et de manière générale, à toute méthode qui fait de la généralisation par sélection et du raffinement par ajout) tout en se conformant à

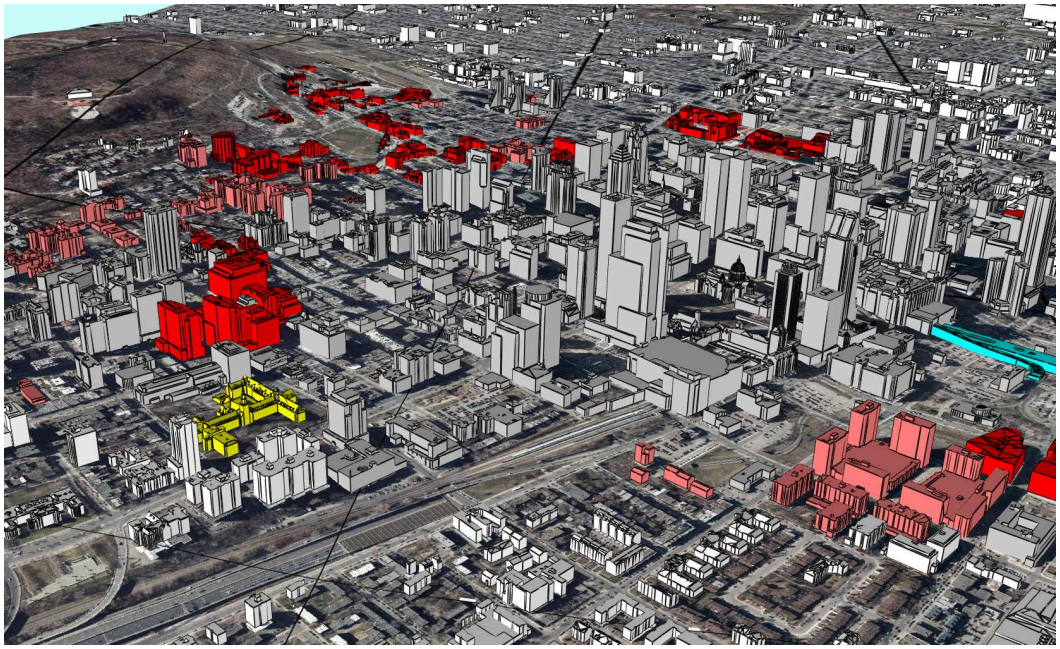

Figure 4.13 Les bâtiments de Montréal colorés selon leur fonction telle que prévue par le plan d'occupation du sol.

la spécification 3D Tiles. La difficulté vient du fait qu'on essaie de faire cohabiter deux comportements différents : d'un côté, l'ajout progressif d'objets, de l'autre le remplacement de ces mêmes objets par des versions plus détaillées. Une organisation naïve des objets dans l'arbre (« hiérarchie originale » dans la fig. 4.14) ne permet pas ce double comportement.

Les modifications qu'il faut apporter à la hiérarchie sont illustrées fig. 4.14. Soient une tuile T_1, son parent T_0 et ses enfants $\{T_2\}$. Soit T_1+ la tuile de niveau de détail supérieur de T_1. Tout d'abord, il faut s'assurer que T_1 n'ait pour enfant que sa tuile de niveau de détail supérieur T_1+. Cela permet de définir un comportement de raffinement par remplacement pour T_1 sans affecter $\{T_2\}$. L'ajout d"une tuiles intermédiaire vide au raffinement par ajout I permet de garantir l'isolation entre T_1 et $\{T_2\}$. I est ajouté comme enfant à T_0, aux côtés de T_1, et $\{T_2\}$ sont ajoutés comme enfants à I. Le seuil de raffinement de I est défini comme étant le même que le seuil de raffinement original de T_1. De cette manière, on conserve le même comportement additif de la structure originale, mais avec une séparation entre T_1 et $\{T_2\}$. T_1 peut ainsi être remplacés par des versions plus détaillés sans modifier le reste du graphe de scène.

Ce procédé est particulièrement utile pour les données texturées, où les objets doivent être individualisés (c'est-à-dire, ne plus être groupés avec un ensemble d'objets dans une tuile) sans quoi la taille des textures de la tuile dépasserait les limites des cartes graphiques côté client. On peut se permettre d'individualiser les bâtiments sans grand risque pour les performances (voir justification pour le *batching*

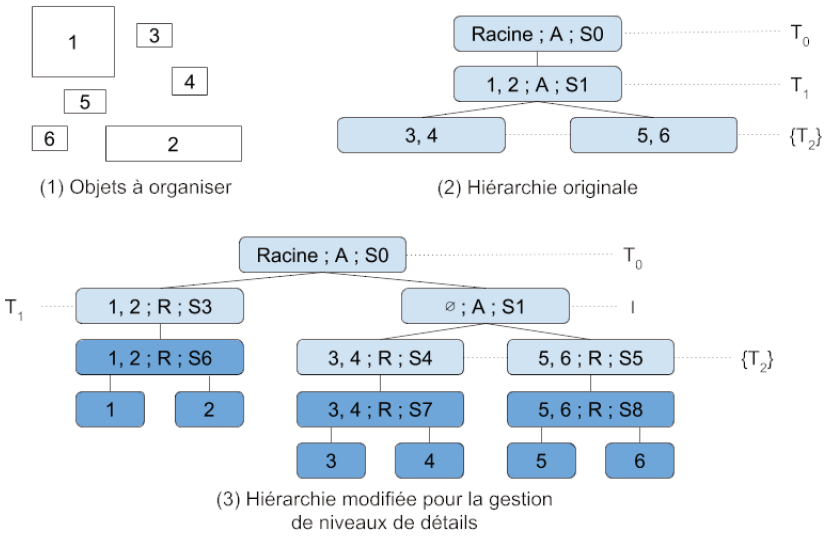

Figure 4.14 Un ensemble d'objets (1) organisé dans une hiérarchie utilisant un raffinement par ajout sans gestion du niveau de détail (2) puis avec (3). Chaque nœud de l'arbre est décrit avec les objets qu'il contient, suivi (si pertinent) du type de raffinement (A pour ajout, R pour remplacement) et du seuil (erreur géométrique) à partir duquel le raffinement a lieu. En bleu foncé, les nœuds ajoutés qui comportent les objets avec un plus haut niveau de détail.

dans la partie 2.2) car cette individualisation ne concerne que les objets proches : ceux pour lesquels un niveau de détail plus élevé est nécessaire.

Cette variante a été spécifiée de façon formelle au cours de ma thèse, mais n'a pas encore pu être mise en pratique au moment de la rédaction de ce manuscrit.

4.5 Découpage sémantique

Principe Bien que la précédente méthode permette la visualisation de la ville à plusieurs niveaux d'échelle, il est difficile de percevoir à grande échelle la manière dont la ville est structurée. En effet, à grande échelle, on visualise un sous-ensemble des bâtiments de la ville de densité homogène. Afin d'améliorer ce point, un autre type de découpage a été conçu, basé sur des caractéristiques sémantiques plutôt que géométriques. Un raffinement par remplacement a aussi été privilégié, afin que les tuiles aient une représentation généralisée qui puisse être plus représentative des bâtiments qu'elles contiennent.

En outre, la structure proposée ici est capable de gérer efficacement les différentes représentations (niveaux de détails, types de données différents, etc.) des objets géographiques : chaque objet et chaque tuile peut être associé à un nombre quelconque de représentations différentes. L'utilisateur peut alors choisir parmi l'ensemble de ses représentations lesquels il souhaite visualiser dans son application (voir section 5.2 pour la présentation d'une méthode de sélection), par exemple il

peut définir dans une zone d'intérêt une représentation en nuage de points et dans le reste de la scène des maillages peu précis. Fig. 4.15 présente le modèle de données de cette structure et fig. 4.16 un exemple de hiérarchie qui applique ce modèle. On note une séparation dans le modèle de données entre les métadonnées de l'objet ou de la tuile (boîte englobante, informations sémantiques) et leurs représentations géométriques.

Les objets géométriques sont considérés individuellement que lorsque l'on souhaite avoir une représentation très détaillés de ceux-ci. Généralement, pour éviter les problèmes de performances liés à la multiplication de géométries individuelles lors du rendu (voir partie 2.2), tous les objets d'une tuile feuille seront regroupés. Par exemple, dans la fig. 4.16, on regroupera les objets $F2$, $F3$ et $F4$.

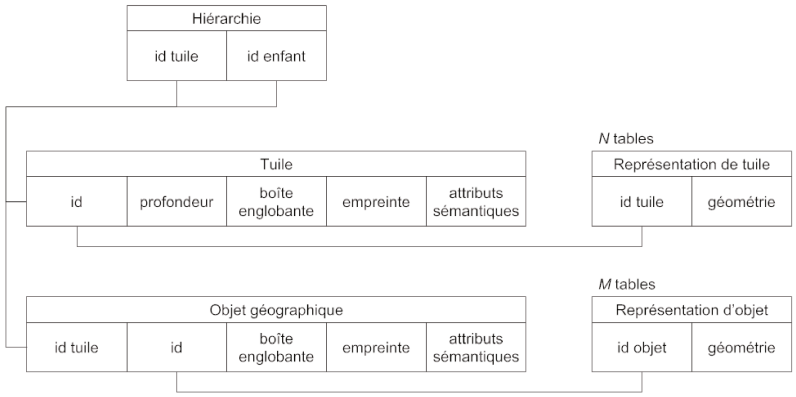

Figure 4.15 Le schéma de données de ma structure hiérarchique. Chaque tuile et chaque objet géographique peut avoir plusieurs représentations. La sémantique et la géométrie sont séparées.

Deux approches ont été étudiées pour construire cette structure. La première méthode cherche à trouver des similarités entre tuiles voisines pour les grouper et former une hiérarchie. La seconde se base sur des partitions de l'espace mises à disposition en Open Data. Il s'agit principalement de découpages administratifs ou routiers : arrondissements, quartiers, îlots urbains...

Dans les deux cas, on obtient des arbres parfaitement équilibrés, mais dont le nombre d'objets par tuiles varie grandement, comme on peut le voir sur l'exemple de la fig. 4.16.

Fusions par similarité Afin de former des groupes de bâtiments pertinents pour la généralisation, il paraît intéressant de chercher à créer des tuiles dont les bâtiments ont des propriétés similaires.

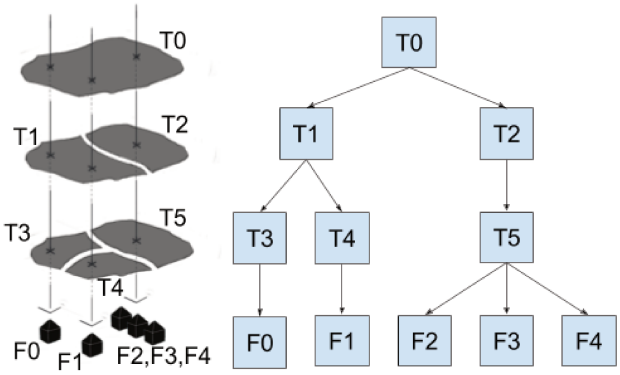

Figure 4.16 Un exemple de hiérarchie créée par cette classe de méthode. Seul le dernier niveau de tuile contient des objets.

On part d'un découpage de l'espace fin, par exemple l'ensemble des îlots urbains, qui peut être obtenu par polygonisation[5] du réseau routier. Mon modèle ne supporte pas les cas où un objet est partagé entre plusieurs tuiles. Il est donc nécessaire de fusionner les îlots urbains qui partagent un même objet.

La similarité entre les tuiles est mesurée à l'aide de plusieurs métriques, comme la taille moyenne des bâtiments ou le taux d'occupation du sol. On peut aussi imaginer utiliser d'autres indicateurs non géométriques, comme le type d'occupation (résidentiel, industriel...).

Bien que la piste semble intéressante, les résultats obtenus (fig. 4.17) n'étaient pas suffisamment bons pour être utilisés en pratique et je n'ai pas eu le temps d'améliorer la méthode. Les défauts notables sont la grande disparité dans la taille des tuiles et une très forte sensibilité aux variations locales (un défaut dans le jeu de données ou un îlot un peu inhabituel peut dégrader fortement le résultat).

Figure 4.17 Fusion sur trois niveaux (du plus fin au plus grossier) de tuiles basée sur la similarité de leur contenu.

[5] Transformation d'un graphe ou d'un ensemble d'arêtes en un ensemble de polygones.

Découpages administratifs De nombreuses ressources sont mises à disposition en ligne par les collectivités qui peuvent nous permettre d'organiser nos données. L'objectif ici est de récupérer des partitions de l'espace de différentes échelles pour créer une hiérarchie de tuiles qui contiendra les objets de la ville. Il n'est cependant pas possible d'utiliser directement ces partitions, il faut les retravailler pour échapper à deux problèmes : les objets qui se trouvent dans plusieurs tuiles et les bordures des tuiles des différentes partitions qui diffèrent le long d'une frontière commune.

Des partitions administratives (arrondissement, quartier, etc.) peuvent être utilisées, mais aussi des partitions basées sur le réseau routier. Comme précédemment, cela peut être utilisé pour trouver les îlots urbains, mais aussi d'autres partitions de plus basse résolution : le réseau routier est lui-même hiérarchique, comportant des routes de différentes importances, on peut donc obtenir d'autres partitions en filtrant les routes par leur importance. Les routes étant des éléments particulièrement structurants pour la ville, les exploiter pour créer la hiérarchie semble aussi pertinent.

Fig. 4.18 est un exemple du traitement effectué sur les partitions pour créer la hiérarchie. Soit un ensemble de partitions $\{P_0, P_1, ..., P_n\}$, (P_0 étant la partition de plus petite échelle, P_n celle de plus grande échelle). Une première étape de nettoyage a lieu : pour chaque objet F du jeu de données, on cherche les tuiles de P_n avec lesquelles F intersecte. Si plusieurs tuiles sont intersectées, celles-ci sont fusionnées. F est affecté à la tuile résultante.

La seconde étape, l'étape de fusion, a pour but de redessiner les bordures des tuiles des partitions $\{P_1, ..., P_n\}$ pour garantir l'adéquation des tuiles d'un niveau à l'autre de la hiérarchie. Il s'agit de s'assurer que l'union des empreintes des tuiles filles soit égale à l'empreinte de la tuile mère. Pour cela, on commence par créer le lien de parenté entre tuile mère et fille : les tuiles $\{T_{i-1}\}$ de P_{i-1} dont le centroïde est dans l'empreinte de la tuile T_i de P_i sont les tuiles filles de celle-ci. Finalement, l'empreinte de la tuile mère est remplacée par l'union des empreintes de toutes ses tuile filles.

Fig. 4.19 montre les résultats de cette méthode sur un jeu de données réel.

Application et résultats Cette méthode a été testée avec les données de Lyon en utilisant le réseau routier, les quartiers et les arrondissements comme partitions de l'espace. Trois types de représentations des données ont été utilisées : des maillages texturés et non texturés et des polygones extrudés. Tous les bâtiments ont ces trois représentations, mais les tuiles n'ont que la représentation par polygones extrudés.

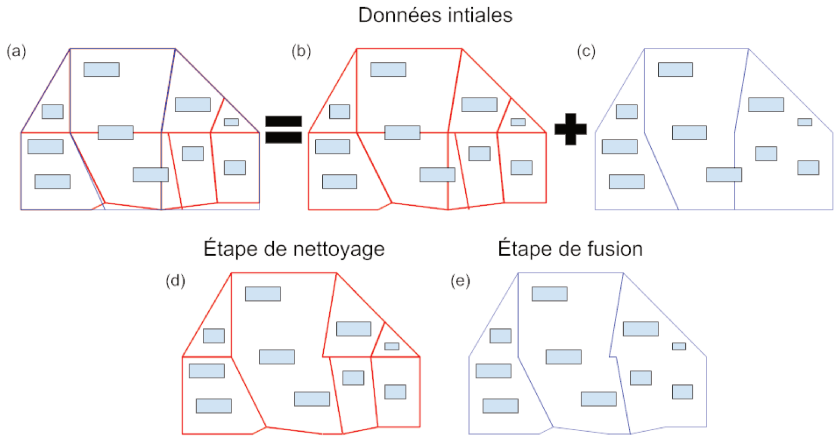

Figure 4.18 Illustration des étapes nécessaires à la création de la hiérarchie de tuiles. (a) est le jeu de données initial, un ensemble d'objets et deux partitions de différentes échelles (en rouge, grande échelle, en bleu, petite échelle). (b) et (c) sont ce même jeu de données mais avec uniquement la partition de grande échelle ou de petite échelle, respectivement. (d) est le résultat de la fusion des tuiles de grande échelle possédant un objet en commun. (e) est le résultat de la fusion des tuiles filles pour créer les tuiles mères.

Les maillages sont extraits de fichiers CityGML fournis par la Métropole de Lyon. La représentation par polygones extrudés résulte d'un processus de généralisation simple : la génération d'une enveloppe concave 2D (via PostGIS) du ou des bâtiments, assorti de la hauteur moyenne du ou des bâtiments. Il existe de bien meilleures manières de généraliser des bâtiments individuels ou des groupes de bâtiments, mais ce n'est pas l'objectif de ces travaux (des méthodes de généralisations de l'état de l'art sont présentées en section 2.4.2).

Le résultat est visible sur les fig. 4.20 et 4.21. L'initialisation de cette structure de ces données a pris 51 secondes (pour 14851 bâtiments et 3796 tuiles) à partir de partitions déjà réalisées. Le temps de génération des formes généralisées pour les tuiles varie énormément en fonction de la méthode utilisée. Pour référence, notre méthode naïve (calcul d'une enveloppe concave en utilisant la fonction native de PostGIS) a pris 24,7 secondes pour générer ces représentations.

Le raffinement par remplacement utilisé ici cause le téléchargement de données redondantes, c'est-à-dire que toutes les données téléchargées ne sont pas visibles, il y a donc une perte d'efficacité. Cette perte est difficilement quantifiable avec la méthode générique présentée ici, car elle dépend de l'algorithme de généralisation, du nombre de niveaux de la hiérarchie et de la manière dont l'espace est découpé.

En basant les regroupements de bâtiments sur un découpage sémantique de la ville, on parvient à faire apparaître la structure de la ville à travers les généralisations à différents niveaux d'échelles. De plus, la séparation entre les objets ou tuiles et leurs représentations géométriques permet bien de gérer les multiples représentations des

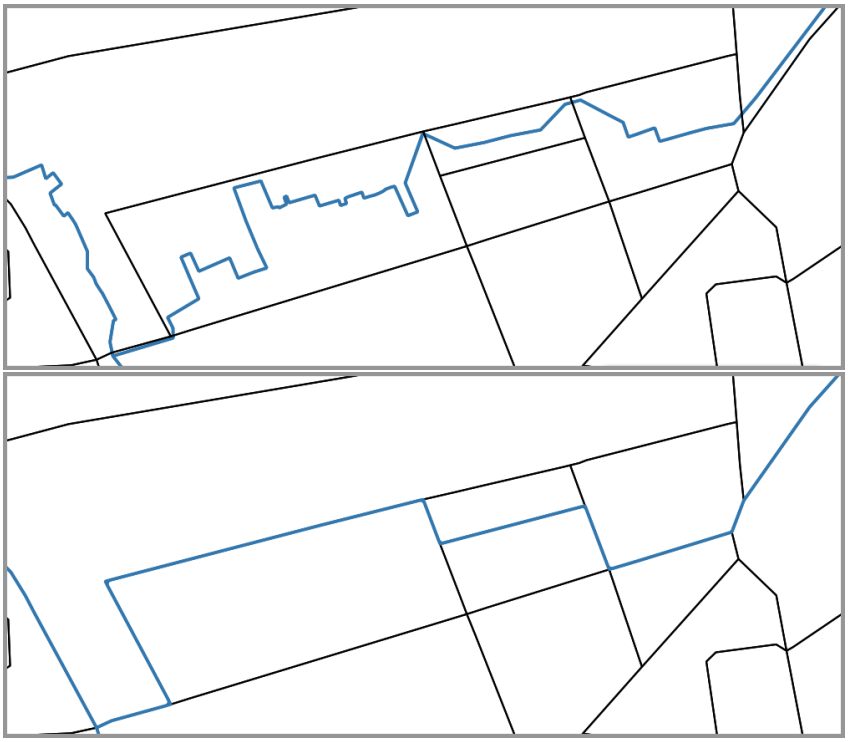

Figure 4.19 Une partition de plus basse résolution (définition des quartiers, en bleu) est recalée sur une partition de résolution supérieure (réseau routier, en noir). L'image du haut contient les bordures originales des quartiers, tandis que l'image du bas contient les bordures recalées. Données du premier arrondissement de Lyon.

données. Dans nos exemples, une représentation 2.5D et une représentation basée sur des maillages ont été utilisées pour représenter les mêmes objets géographiques.

Limites Un problème de cette méthode est l'absence de contrôle sur la taille de chaque tuile, résultant en la création de tuiles parfois très grandes ou, au contraire, très petites. Les algorithmes de découpe gagneraient à être améliorés dans ce sens. J'utilise les îlots urbains comme partition de plus haute résolution, mais ce n'est pas toujours adapté : dans certaines configurations (rivière avoisinante, zone rurale...) les routes ne forment que rarement des cycles et génèrent des îlots extrêmement grands. Cette grande taille devient un problème si elle s'accompagne d'une grande quantité de données, lorsque l'îlot est étendu et dense. Découper ces îlots anormalement grands donnerait une plus grande robustesse à ma méthode et éviterait des temps de chargement anormalement longs qui nuisent à l'expérience utilisateur. Au contraire, en centre ville, où le maillage routier est très dense, les îlots sont trop petits et ne permettent pas de réduire significativement le nombre de géométries à traiter par la carte graphique. Dans ce cas, c'est la fusion de tuiles voisine qui améliorerait les performances.

Figure 4.20 Application de ma méthode de découpage sémantique aux données de la ville de Lyon. Quatre niveaux de de zooms successifs, faisant apparaître des tuiles de plus en plus détaillées. Les tuiles de niveau 0 (arrondissements), 1 (quartiers) et 2 (îlots) sont représentées par des formes généralisées, des polygones extrudés. Les bâtiments sont représentés par des maillages.

4.6 Synthèse

Les différentes structures présentées dans ce chapitre peuvent être évaluées selon un ensemble de critères :

- le caractère hiérarchique de la structure ;
- le type de raffinement ;
- la possibilité d'éditer interactivement les données ;
- le temps de préparation des données ;
- la gestion des niveaux de détails et des représentations des données ;
- la robustesse de la méthode ;
- la préservation des données originales.

Le tableau 4.2 présente les résultats de cette évaluation, dont le détail est disponible ci-après.

Hiérarchique Le caractère hiérarchique d'une structure permet la gestion de l'aspect multi-échelle des données. Parmi mes trois méthodes, seule la méthode basée grille n'est pas hiérarchique.

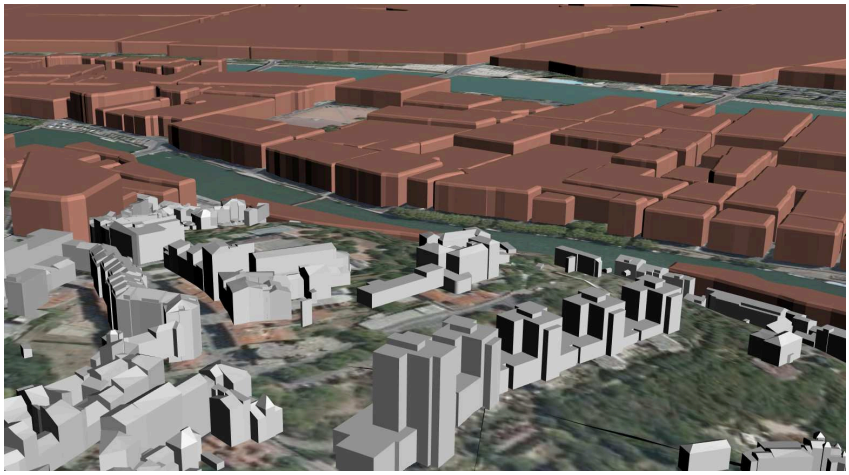

Figure 4.21 Application de ma méthode aux données de la ville de Lyon. Vue oblique permettant d'observer le mélange des différents niveaux de tuiles et de représentations. Au premier plan se trouvent les représentations les plus détaillées, et plus les éléments s'éloignent, plus leur représentation est grossière.

	Grille	Quadtree	Découpage sémantique
Hiérarchique	-	+	+
Raffinement	-	+ (ajout)	+ (remplacement)
Édition	-	+	-/+
Temps de préparation	-	+	-/+
Représentations	-	-	+
Robustesse	+	+	-
Préservation	-	+	+

Table 4.2 Évaluation de mes méthodes par rapport à une liste de critères définis.

Raffinement La méthode basée grille n'a pas de notion de raffinement car ce n'est pas une structure hiérarchique. La méthode basée quadtree a un raffinement par ajout, ce qui a l'avantage de ne pas charger de données redondantes, contrairement à la méthode basée sur un découpage sémantique qui utilise un raffinement par remplacement.

Édition L'édition n'est pas possible dans la méthode basée grille. La méthode basée quadtree permet une édition interactive. La méthode basée sur un découpage sémantique supporte certains types d'éditions seulement. L'ajout de nouvelles représentations est facilement faisable, mais l'ajout de nouveaux objets peut nécessiter de reconstruire des généralisations ou de modifier la structure (dans le cas où le nouvel objet s'étend sur plusieurs tuiles).

Temps de préparation La méthode basée sur une grille nécessite un long temps de préparation, de l'ordre de l'heure. La méthode basée sur un quadtree est très rapide, l'initialisation de la structure prenant un temps de l'ordre de la dizaine de secondes. La méthode basée sur un découpage sémantique prend un temps qui est

très variable en fonction des algorithmes de découpe et de généralisation utilisés. Dans mon cas (où les algorithmes utilisés sont de faible complexité), ce temps était aux alentours de la minute.

Représentations Seule la méthode basée sur un découpage sémantique permet la gestion des multiples représentations des objets. Les autres méthodes doivent se contenter d'un système de couches de données, ce qui revient à dupliquer les structures pour différents types de données. Dans ce cas, aucun lien n'existe entre les différentes représentations du même objet géographique.

Robustesse La méthode basée sur une grille est particulièrement robuste. Le fait de découper les données selon la grille permet d'être sûr que n'importe quel objet pourra être géré, quelque soit sa taille ou sa forme. La méthode basée sur un quadtree est plutôt robuste. Il faut seulement veiller à ce que les très gros objets soient placés dans les premiers niveaux de la hiérarchie pour éviter de déformer trop les tuiles du quatree. La méthode basée sur un découpage hiérarchique manque de robustesse, et c'est d'ailleurs une piste de recherche mentionnée précédemment.

Préservation Seule la méthode basée sur une grille ne préserve pas les données originales. Cela peut résulter en une perte de la sémantique liée aux objets géographiques ou de certaines propriétés de ces objets (comme l'étanchéité évoquée précédemment).

En résumé, trois structures ont été présentées qui possèdent chacune leurs avantages et leurs limites. Il n'y a pas de solution universelle, et le choix de la structure va dépendre des cas d'utilisations qui devront être gérés. Quelle que soit la structure utilisée, leur interopérabilité est garantie par leur compatibilité avec les standards de description de scènes 3D, notamment 3D Tiles qui est utilisé dans le cadre de ces travaux.

5 Personnalisation de la visualisation de données géographiques

Sommaire

Les cas d'utilisations des applications géospatiales 3D sont variés et nombreux. Comme nous l'avons vu en introduction, cela va de la visualisation de données à des fins touristique à la simulation ou la prévention des risques. Dans ces cas d'utilisations, l'information recherchée par l'utilisateur est différente. Pour prendre un exemple concret, dans une application de recherche d'itinéraire, la position des centres de transports est particulièrement importante. Pour une application touristique, ce sont les monuments et lieux historiques qui sont à mettre en avant. Parfois, une partie spécifique de l'environnement va être plus intéressante. Une agence immobilière voudra détailler les environs des appartements qu'elle propose, sans pour autant cacher le contexte que constitue le reste de la ville.

Adapter la représentation des données à un usage est le principal objectif de la personnalisation. Elle a cependant d'autres vertus. Du fait du contexte web et de la volumétrie des données, le délai avant l'obtention ou l'affichage des données est conséquent. En connaissant le but de l'utilisateur ou de l'application, on peut prioriser le chargement des données de telle sorte que les objets qui sont le plus porteurs d'informations pour l'usage défini sont chargés en premier. De manière similaire, les données importantes peuvent être représentées avec davantage de détails et les données auxiliaires de manières plus grossières. De cette façon, on peut réduire la quantité de données à afficher, sans pour autant réduire la quantité d'informations. Au contraire, cela peut même mettre en valeur les informations importantes.

Dans le cadre de ma thèse, je me suis demandé quelles mécaniques de priorités pouvaient être mise en place dans des applications de géovisualisation sur le Web et comment gérer la personnalisation d'une scène sans avoir à matérialiser toutes les vues.

Ce chapitre est organisé en deux parties. La première étudie les mécanismes de priorisation développés pendant ma thèse, que ce soit pour le client ou pour le serveur. La seconde partie décrit une méthode permettant à un utilisateur de personnaliser une scène en choisissant parmi les différentes représentations des objets à l'aide d'un langage de règles. Le chapitre se clôt par une synthèse.

5.1 Gestion des priorisations

La priorisation est le premier mécanisme de personnalisation présenté dans ce chapitre. La priorisation prend deux formes différentes selon qu'elle est effectuée côté client ou côté serveur. Comme les données sont généralement regroupées sous la forme de tuiles qui rassemblent plusieurs objets géographiques, le client ne peut pas agir sur les objets individuels, mais seulement sur quelles tuiles il veut charger.

De son côté, le serveur peut choisir la manière dont les objets sont répartis dans les tuiles.

Nous verrons donc séparément ces deux manières de prioriser les données, en commençant par la priorisation côté client, avant d'étudier la priorisation côté serveur.

5.1.1 Priorisation côté client

Le choix des tuiles à charger en priorité côté client à un intérêt particulier lorsque le temps de chargement est long ou lorsque les capacités du client sont limités : dans le premier cas, l'attente paraît d'autant plus longue si les données chargées ne sont pas pertinentes, dans le second cas, prioriser revient à sélectionner les données qui seront affichées avant que les limites de l'appareil utilisé soient atteintes.

5.1.1.1 Stratégies de chargement de couches de données

La classification des données géospatiales en différentes catégories permet de les séparer en plusieurs couches de données. Par exemple, on peut avoir une couche de terrain, une couche de bâtiment, une couche de végétation, une couche de noms de lieux... Traditionnellement, dans un contexte 2D, l'utilisateur choisi parmi ces différentes couches lesquelles il souhaite afficher ou cacher, voire combiner par transparence.

Des comportements plus complexes peuvent être implémentés pour gérer les différentes couches de données. Je présente ici la manière dont un ensemble de règles peut être utilisé pour définir plus finement l'affichage et le chargement de ces couches.

Ces ensembles de règles sont appelées *stratégies*. Une stratégie permet de déterminer quelles données charger et avec quelle priorité.

Soit $T_{candidates}$ comme l'ensemble des tuiles (non-chargées) visibles par la caméra. La stratégie est une fonction S qui, appliquée à l'ensemble des tuiles candidates, renvoie l'ensemble des tuiles à charger $T_{retenues}$, classées dans n sous-ensembles $T^i_{retenues}$, $i \in [1...n]$, où i est la priorité des tuiles du sous-ensemble. n est le nombre de priorités différentes définies par la stratégie, 1 étant la plus basse priorité et n la plus haute.

À chaque priorité est associée une file d'attente FIFO (*First In First Out*, le premier élément inséré est le premier sortie de la file) Q^i dans laquelle les tuiles de $T^i_{retenues}$ sont insérées. On a donc une liste de files Q^i, $i \in [1...n]$. Toutes les tuiles de Q^{i+1}

doivent être chargées avant que les tuiles de Q^i soient chargées, de telle sorte qu'une tuile moins prioritaire ne sera jamais chargé avant une tuile plus prioritaire.

Ci-après, je présente une implémentation de cette solution pour la structure de données en grille présentée en partie 4.3.

Dans le cas de figure illustré par la fig. 5.1, deux niveaux de priorités sont définis, haute priorité (file Q^2) et basse priorité (file Q^1). Deux couches de données sont à considérer : le terrain et les bâtiments. La stratégie utilisée est la suivante :

- pour la couche de bâtiments, la tuile est chargée en haute priorité $T \in T^2_{retenues}$ si la caméra est directement au dessus de la tuile et en basse priorité $T \in T^1_{retenues}$ si la caméra est dans un rayon de une case de la tuile ;
- pour la couche de terrain, la tuile est chargée en haute priorité $T \in T^2_{retenues}$ si la caméra est dans un rayon de une case de la tuile et en basse priorité $T \in T^1_{retenues}$ si la caméra est dans un rayon de deux cases de la tuile ;
- dans tous les autres cas, la tuile n'est pas chargée $T \notin T_{retenues}$.

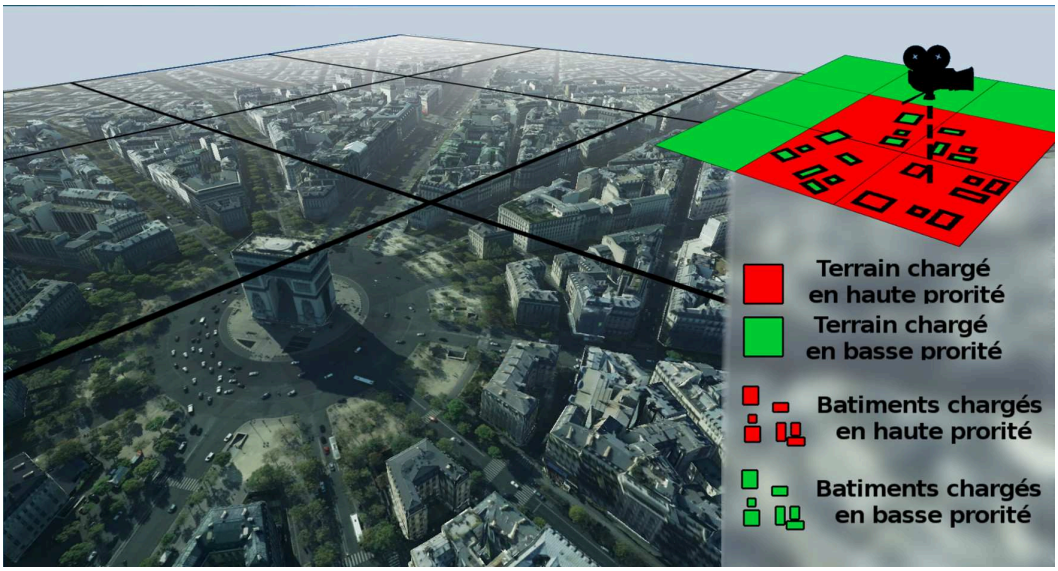

Figure 5.1 Stratégie de chargement pour une couche de terrain et une couche de bâtiments.

5.1.1.2 Priorité spatiale

Une autre approche à ce problème de priorisation se base sur une file d'attente unique, une file de priorité, c'est-à-dire une liste dont les éléments sont ordonnés selon les priorités. Chaque élément inséré dans la file possède sa propre priorité et le premier élément à sortir de la file est celui dont la priorité est la plus haute.

Le choix de la méthode de calcul de cette priorité dépend du comportement voulu. Plusieurs métriques sont pertinentes pour déterminer la priorité :

- la distance à la caméra ;
- le niveau de détail de la tuile considérée ;
- la distance à un point d'intérêt.

5.1.2 Priorisation côté serveur

La méthode présentée en section 4.4 nécessite de classer les objets géographiques en fonction de l'importance de ceux-ci. Pour rappel, cette méthode répartie des objets géographiques dans une hiérarchie de boîtes englobantes construite à partir d'un quadtree. Le nombre d'objets par tuile étant limité, il faut décider de la manière dont les objets sont mis dans la structure hiérarchique, sachant que plus l'objet est haut dans la hiérarchie, plus il sera chargé tôt. Pour effectuer cette décision, un poids est associé à chaque objet géographique, de telle sorte que plus le poids est grand, plus l'objet aura tendance à être haut dans la hiérarchie. Je présente ici la manière de calculer ce poids. Une fonction de poids est définie pour associer un poids à chaque objet géographique F. Cette fonction de poids W peut se définir de la sorte :

$$F \mapsto W(F) \in [0..1]$$

Cette fonction de poids est la même pour tous les objets à insérer dans la structure de données. La manière dont cette fonction de poids est construite dépends du cas d'utilisation. Par exemple, une application touristique définira un poids plus fort pour les monuments. Ci-après, des recommandations pour la définir.

Poids surfacique Une classification basée sur l'empreinte 2D d'un bâtiment (ou autre objet géographique) permet aux plus grands bâtiments d'être affichés en priorité. Ceci est un choix pertinent puisque ces bâtiments peuvent être vus de loin et peuvent servir de points de repères pour l'utilisateur. On s'intéresse à l'empreinte 2D du bâtiment plutôt qu'à son volume car lorsqu'on observe une ville de loin (et donc lorsque les bâtiments les plus prioritaires sont chargés), on a habituellement une vue plongeante sur la ville, et c'est donc la surface (et non la hauteur) du bâtiment qui importe.

La fonction de poids surfacique est définie comme une fonction qui renvoie 0 pour une surface nulle et 1 pour le plus grand bâtiment du jeu de données. Comme les villes ont souvent quelques bâtiments très grands en comparaison des autres bâtiments, il y a un risque que les poids se retrouvent concentrés près de 0. Pour éviter cela, je définis le poids du bâtiment de taille moyenne comme étant 0,5. La

fonction suivante répond à ces critères, avec S la surface du bâtiment, max la plus grande surface des bâtiments et moy la surface moyenne des bâtiments :

$$W_{poids\_surfacique}(F) = \begin{cases} \frac{S}{2*moyenne} & \text{si } S < moyenne \\ 1 - \frac{S-max}{2*(max-moy)} & \text{si } S >= moyenne \end{cases}$$

Poids attributaire Les informations sémantiques liées aux objets géographiques peuvent être mises à contribution pour déterminer la priorité d'un objet. Ces informations varient beaucoup dans leur nature, il est donc difficile de définir des bonnes pratiques dans leur cas. Pour le cas le plus commun, une classification des objets dans un nombre limité de classes, il suffit d'assigner un poids à chaque classe en fonction de l'importance qu'on souhaite leur donner. Pour reprendre l'exemple de l'application touristique, les objets géographiques classés comme monuments peuvent être assignés un poids de 1, les arrêts de transports en commun un poids de 0,5 et les autres bâtiments un poids de 0.

Combinaison de fonctions de poids Ces fonctions de poids peuvent être combinées pour créer des fonctions plus complexes. À chaque fonction W_i est assignée une importance w_i. La nouvelle fonction de poids est la moyenne pondérée de ces fonctions :

$$W(F) = \frac{\sum_{i=0}^{n} w_i W_i(F)}{\sum_{i=0}^{n} w_i}$$

Il faut cependant garder à l'esprit qu'on mélange des grandeurs qui n'ont pas forcément de rapports entre elles (par exemple, la taille d'un objet et sa classe énergétique), il n'y a donc pas de formule scientifiquement exacte pour combiner ces poids. Il est préférable de choisir les valeurs d'importance en fonction du résultat attendu, comme illustré par l'exemple ci-après.

La fig. 5.2 illustre l'incidence du choix de la fonction sur un même jeu de données. Deux fonctions sont utilisées puis combinées pour former la fonction de poids W_C, la fonction de poids surfacique W_1 (d'importance w_1) et une fonction de poids attributaire W_2 (d'importance w_2) qui assigne un poids de 1 pour les objets rouges et 0 pour les objets bleus. La première fonction composite W_{C1} utilise les poids $w_1 = 1$ et $w_2 = 0$, soit un classement basé purement sur la surface des objets. La seconde fonction composite W_{C2} utilise les poids $w_1 = w_2 = 1$, ce qui résulte en une fonction qui priorise les objets rouges et ensuite ordonne les objets par surface. La troisième fonction W_{C3} utilise les poids $w_1 = 1$ et $w_2 = 0.5$, la fonction ainsi créée garantit que tous les objets rouges plus grands que la moyenne seront chargés en premiers.

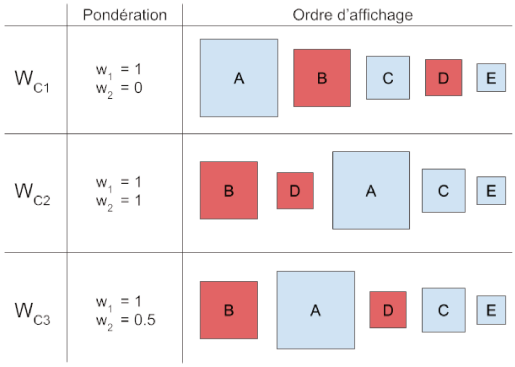

Figure 5.2 Classements obtenus par l'application de différentes fonctions de poids. W_{C1}, poids surfacique. W_{C2}, combinaison du poids surfacique ($w_1 = 1$) et du poids attributaire ($w_2 = 1$). W_{C3}, combinaison du poids surfacique ($w_1 = 1$) et du poids attributaire ($w_2 = 0.5$).

Des expérimentations ont été faites sur le jeu de données de la ville de Lyon. Les bâtiments de Lyon sont classées en catégories : remarquables et non-remarquables. La fonction de poids W_{C2} a été utilisée pour prioriser ces bâtiments remarquables. Le résultat est visible sur la fig. 5.3, où les bâtiments remarquables ont été colorisés en rouge. Le nombre de triangles chargés a aussi été limité à un total de 500 000 pour que la comparaison soit pertinente. L'image de gauche utilise la fonction de poids W_{C1}, l'image de droite la fonction de poids W_{C2}. On obtient bien le résultat espéré : avec la fonction W_{C2}, les bâtiments remarquables apparaissent en priorité.

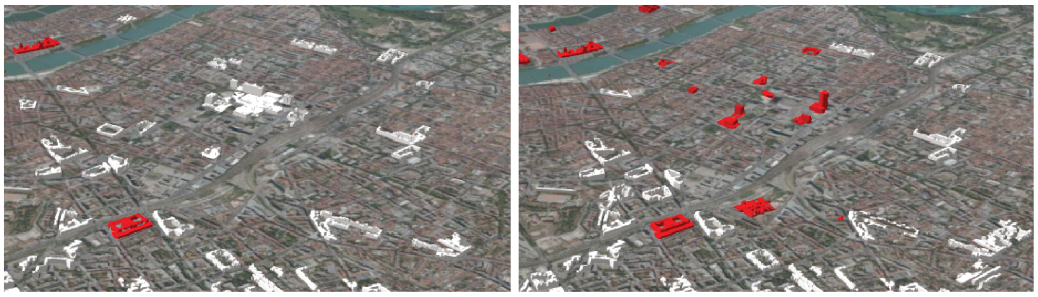

Figure 5.3 Les bâtiments de Lyon priorisés par leur surface (gauche) et par leur proximité au Rhône (droite).

La même expérience a été réalisée avec une fonction de poids favorisant les bâtiments proches du Rhône en fig. 5.4. La fonction est inversement proportionnel à la distance entre le bâtiment et la polyligne décrivant le tracé du Rhône. On note un changement de la répartition des bâtiments qui se rapprochent du fleuve. Cependant l'effet reste limité, et certaines berges restent vides. En effet, comme nous l'avons vu dans les limites de cette structure (section 4.4), la densité aura tendance à rester uniforme, ce qui empêche d'avoir une concentration trop forte de bâtiments autour du fleuve.

Figure 5.4 Les bâtiments de Lyon priorisés par leur surface (haut) et par leur proximité au Rhône (bas).

5.2 Génération personnalisée de graphe de scènes

Dans la partie 4.5, nous avons vu que la représentation d'un objet géographique n'est pas unique. Il peut être représenté à différents niveaux de détails, de manière abstraite, ou en utilisant des types de données particuliers (maillages, nuages de points...). Le choix de la représentation dépend de l'utilisation qui sera faite de cet objet.

Dans les applications de l'état de l'art, les données sont généralement statiques. Un jeu de données est transformé et structuré dans un format exploitable pour le Web (3D Tiles par exemple) et stocké sous la forme de fichiers. Cette manière de faire n'est pas adaptée à des jeux de données multi-représentations, où chaque objet peut prendre une variété de formes. Je propose dans cette section de générer dynamiquement l'arbre de scène (le tileset dans la spécification 3D Tiles) en fonction des préférences de l'utilisateur pour choisir les représentations adaptées aux besoins de celui-ci. L'idée est d'exploiter la séparation entre la structure (le tileset) et les géométries (les tuiles) de 3D Tiles. En conservant une structure (majoritairement) fixe et en changeant les géométries sur lesquelles pointe le tileset, on peut générer efficacement des scènes personnalisées, où l'utilisateur peut choisir les représentations en fonction de ses préférences.

Deux objectifs ont été identifiées :

- permettre le choix de la représentation des tuiles et des objets géographiques ;
- permettre de mettre en valeur des objets individuels.

Le second objectif consiste à donner la possibilité d'agir sur des objets individuels (par oppositions aux tuiles que l'on manipule habituellement) en choisissant leur représentation et en les faisant apparaître plus tôt dans la hiérarchie (fig. 5.5).

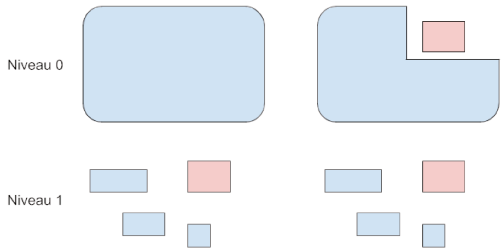

Figure 5.5 Mise en valeur d'un objet (en rouge) en le faisant remonter dans la hiérarchie. À gauche, la hiérarchie originale (une tuile et quatre objets fils) et à droite la hiérarchie avec l'objet mis en valeur : il est remonté dans la hiérarchie et la tuile est découpée de telle sorte que l'objet soit visible.

Le défi est de parvenir à combiner la vitesse d'une structure statique avec la flexibilité d'une base de données géospatiale. En effet, utiliser uniquement l'index spatial de la base de données causerait les temps de génération des tuiles bien trop longs. Une part de précalculs est nécessaire pour réduire l'attente. La solution présentée ici se base sur la structure de données présentée en section 4.5 qui permet la gestion des multiples représentations des tuiles et des objets et l'accès très rapide à ces éléments. Cette méthode nécessite d'aller un cran plus loin, c'est-à-dire de partitionner la ville en un ensemble de zones associées à chaque objet géographique, que l'on appellera empreintes. L'avantage de générer ces partitions est que cela définit l'appartenance de tout point de l'espace à un objet, ce qui permet par exemple de savoir à quel objet il faut associer quelle partie d'un nuage de points.

La première phase de la méthode présentée ici est la préparation des données. La structuration des données dans une base de données géospatiale évoquée à l'instant fait partie de cette préparation. La deuxième phase est la personnalisation du graphe de scène en fonction des préférences de l'utilisateur, exprimées sous la forme de règles. Finalement, la troisième phase est celle de transfert, où le client récupère les données décrites dans le graphe de scène. Ces deux dernières phases sont calquées sur le modèle de fonctionnement habituel du format 3D Tiles (envoie du tileset, puis récupération des géométries), mais ont été enrichies avec des options de personnalisation.

5.2.1 Préparation

À l'issue du processus de création de hiérarchie présenté en section 4.5, on obtient un arbre de tuiles. Chaque tuile possède un ensemble de représentations, et les tuiles qui sont les feuilles de l'arbre indexent les objets géographiques qui possèdent eux aussi leurs propres représentations. Il reste à définir une empreinte pour chacun des objets. Cette empreinte sera utilisée pour créer une zone vide autour d'un objet

qu'on souhaite mettre en valeur. Le processus de préparation est résumé par la fig. 5.6.

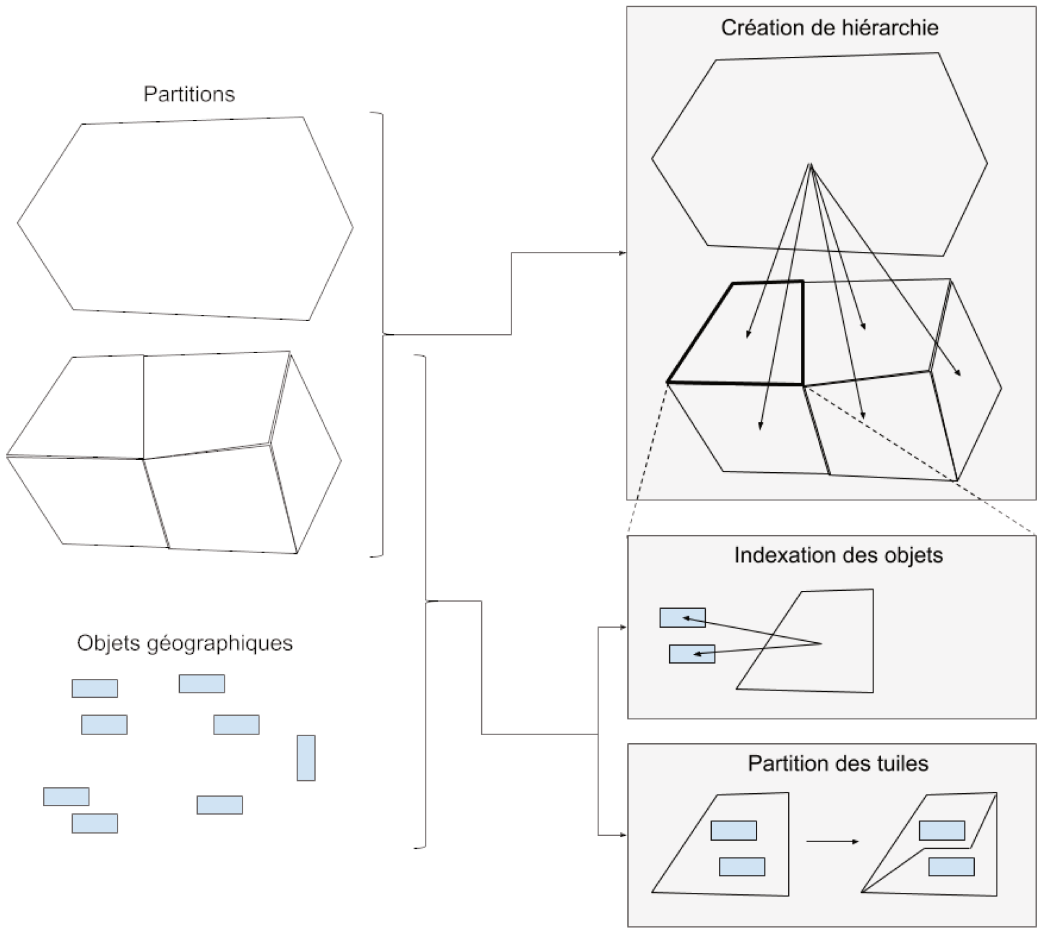

Figure 5.6 Les opérations de la phase de préparation.

La création des empreintes des objets revient à segmenter les tuiles qui contiennent des objets (les tuiles qui sont les feuilles de la hiérarchie). Le processus est similaire à celui utilisé par [Fil+16] (mais qui n'est pas explicitement décrit dans l'article) et se base sur une squelettisation [FO98] de la tuile. L'algorithme (algorithme 4, fig. 5.7) pour chaque tuile est le suivant. Soient $e(T)$ l'empreinte de la tuile T, $p(F)$ la projection de l'objet F sur le plan horizontal (1). Tout d'abord, $e(T)$ est dilaté d'une distance d égale à au moins la moitié de sa plus grande diagonale (explication de cette étape au paragraphe suivant) (2). Ensuite, on soustrait à $e(T)$ la projection $p(F)$ de tous ses objets (3). On calcule la squelettisation du polygone troué résultant $p_{troué}$: on obtient un découpage de $p_{troué}$ en un ensemble de polygones $\{p_{squelette}\}$ (4). Chaque polygone $p_{squelette}$ est affecté à l'objet dont il est le voisin et ajouté à son empreinte $e(F)$ (5). Finalement, $e(F)$ est découpé selon les contours originaux (avant dilatation) de $e(T)$ (6).

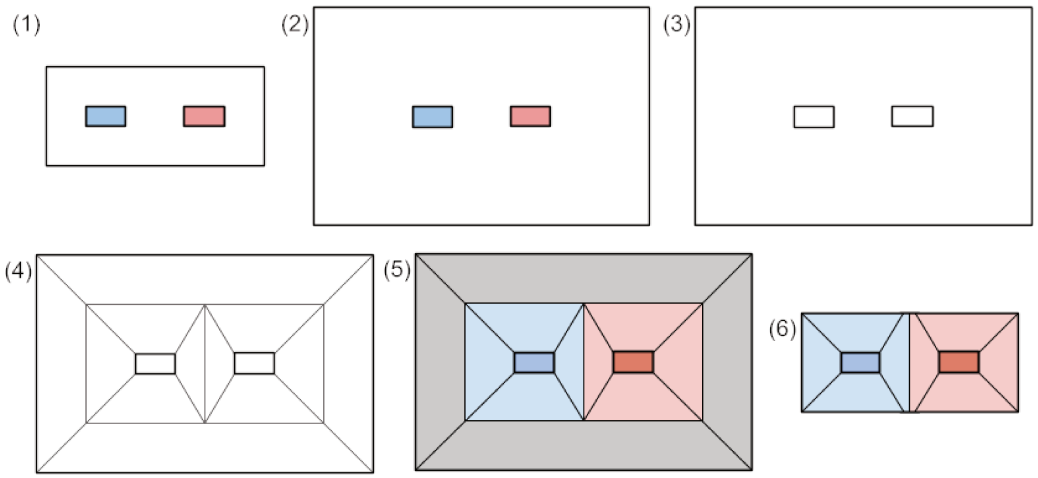

Figure 5.7 Segmentation d'une tuile pour calculer les empreintes de ses objets géographiques.

Algorithme 4 Algorithme de partition de tuile.

1: $e'(T) = dilater(e(T), d)$
2: $p_{troué} = e'(T) \setminus \{p(F)\}$
3: $\{p_{squelette}\} = squelettiser(p_{troué})$
4: **for** F **in** $\{F\}$ **do**
5: $\quad e(F) = p(F)$
6: $\quad$ **for** $p_{squelette}$ **in** $\{p_{squelette}\}$ **do**
7: $\quad\quad$ **if** $intersecte(p_{squelette}, p(F))$ **then**
8: $\quad\quad\quad e(F) = e(F) \cup p_{squelette}$
9: $\quad e(F) = e(F) \cap e(T)$

La dilatation est nécessaire à cet algorithme car les bords extérieurs de la tuile sont pris en compte par la squelettisation mais ne nous intéressent pas. En effet, le squelette nous permet de savoir pour tout point d'un polygone, de quel bord il se rapproche le plus. Mais comme les seuls bords qui nous intéressent sont ceux issus des objets (donc les trous du polygone), éloigner les bords extérieurs grâce à la dilatation permet de s'assurer que toute cette zone liée aux bords extérieurs se trouve en dehors de l'empreinte de la tuile originale. Ce problème est illustré par la figure 5.8.

La fig. 5.9 est un exemple de résultat obtenu avec cet algorithme.

Alternativement, pour éviter ce processus qui peut s'avérer très long, on peut générer une zone tampon autour de chaque objet géographique à l'aide d'une dilatation de la projection 2D de l'objet. L'inconvénient étant que le résultat obtenu ne forme pas une partition de l'espace.

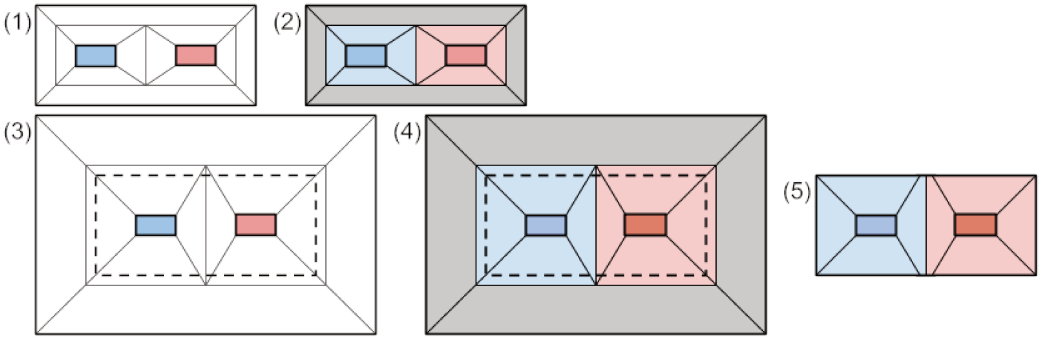

Figure 5.8 Explication de la nécessité de la dilatation pour mon algorithme de segmentation. Sans dilatation (1), les deux objets (en bleu et rouge) auront des empreintes qui ne couvriront pas la totalité de la tuile (2). Avec dilatation (3), on observe que la bordure originale de la tuile (en pointillés) est à l'intérieur des empreintes des objets (4). Après la découpe (5), on a bien une segmentation de la tuile.

Figure 5.9 Une tuile partitionnée en un ensemble d'empreintes. Exemple issu des données de la Métropole de Lyon. En rouge, les bâtiments et, en vert, leurs empreintes après squelettisation.

5.2.2 Génération du graphe de scène

Définition des règles L'utilisateur définit ses préférences à travers un ensemble de règles. Deux types de règles existent : les règles de tuiles qui indiquent comment une tuile doit être représentée à différents niveaux d'échelle, et les règles d'objets qui permettent de choisir des objets géographiques qui apparaîtront plus tôt, c'est à dire qui seront remontés dans la hiérarchie. Le terme *élément* est utilisé pour désigner indifféremment une tuile ou un objet géographique.

Les règles sont composées d'une condition C et d'une action A. La condition spécifie quels éléments sont affectés par la règle, en se basant sur leur position ou

leurs attributs. L'action définit comment l'élément doit être représenté à un certain niveau d'échelle.

Les règles sont appliquées les unes après les autres, les premières règles ayant la priorité en cas de conflit. Si une tuile ne valide aucune règle, une action par défaut lui est appliquée.

Ci-après, la définition de notre langage de règle sous la forme d'une grammaire formelle [Cho57]. Une grammaire formelle est composée d'un ensemble de règles de productions, où les symboles non-terminaux (en lettres capitales) sont transformés en symboles non-terminaux et terminaux (en lettres minuscules). L'étoile signifie que le symbole précédent peut être généré un nombre quelconque de fois, le signe plus signifie que le symbole précédent doit être généré au moins une fois.

$$
\begin{aligned}
&ENSEMBLE\_RÈGLES \rightarrow DEFAUT \quad RÈGLE*\\
&DEFAUT \rightarrow A\\
&RÈGLE \rightarrow (règle\_tuile \mid règle\_objet)\ C+\ A\\
&C \rightarrow condition\_spatial \mid condition\_attributaire \mid \ldots\\
&A \rightarrow (niveau\ représentation+)+
\end{aligned}
$$

En d'autres termes, un ensemble de règles est constitué d'un comportement par défaut (une action) et d'une liste de règles. Chaque règle est une liste de conditions qui déclenchent une action. Il y a plusieurs types de conditions, comme les conditions spatiales (par exemple, un élément est dans un cercle) ou attributaires (par exemple, un élément est plus haut qu'un certain seuil). Une action définit les représentations des éléments affectés à chaque niveau d'échelle. Pour l'action par défaut et les actions des règles de tuile, cela se traduit par le fait qu'une tuile d'un certain niveau d'échelle sera représentée par les représentations définies par l'action à ce niveau d'échelle. Pour les règles d'objet, cela signifie que les objets affectés seront affichés en même temps que les tuiles de ce niveau d'échelle (ces objets sont « remontés » dans la hiérarchie, voir fig. 5.5) avec les représentations définies.

Prenons un exemple concret (fig. 5.10) pour expliciter le fonctionnement de ces règles. Les règles sont les suivantes :

- par défaut, les tuiles de niveaux 0 sont représentées en vert, puis en rouge ; les tuiles de niveau 1 sont représentées en bleu ;
- règle de tuile : si une tuile se trouve dans le cercle rouge, elle doit être représenté en rouge au niveau 1 ;

- règle d'objet : si un objet est un triangle, le représenter au niveau 1 en vert.

La résolution de la règle de tuile fait que la tuile T2 sera représentée en rouge. La résolution de la règle d'objet fait que les objets F3 et F5 seront représentés en vert au niveau d'échelle 1. Le comportement par défaut fait que la tuile T0 sera représenté en vert, puis en rouge, et que la tuile T1 sera représentée en bleu.

Au niveau 0, on obtient donc : la tuile T0 représentée en vert, puis la tuile T0 représentée en rouge. Au niveau 1, on obtient : la tuile T1 représentée en bleu, la tuile T2 représenté en rouge et les objets F3 et F5 représentés en vert. Comme les objets F3 et F5 apparaissent au niveau 1 à la place du niveau 2, les représentations des tuiles T1 et T2 sont découpées pour permettre à F3 et F5 d'être visible.

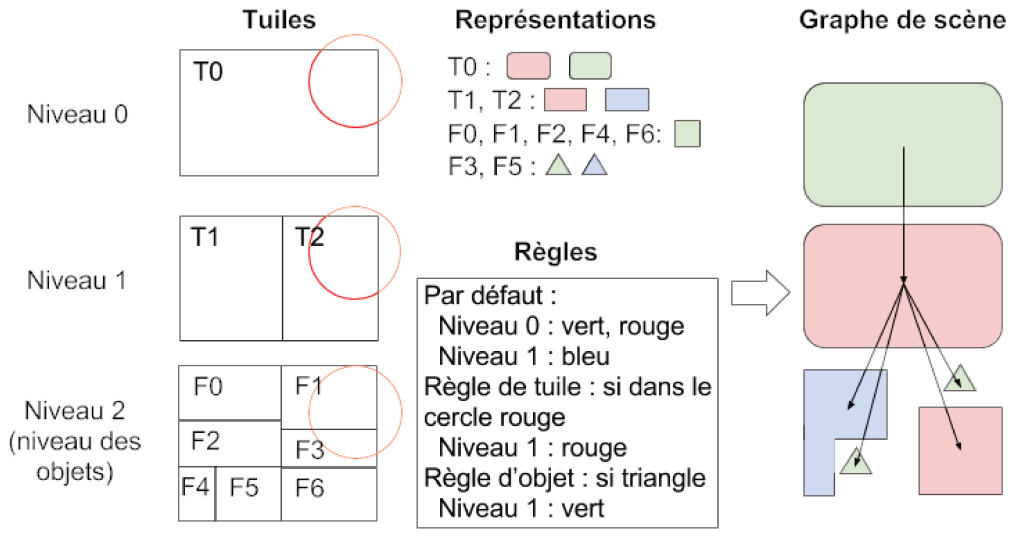

Figure 5.10 Exemple d'un ensemble de règles appliqué à un jeu de données. Les différents éléments de la structure hiérarchique (gauche) ont différentes représentations (haut). Les règles (bas) appliquées à ces éléments résultent en un arbre de scène (droite).

Moteur de règles Le moteur de règles est le processus qui permet de transformer une hiérarchie de tuiles en un graphe de scène par l'application d'un ensemble de règles. Chaque tuile de la hiérarchie est transformée en un ensemble de nœuds du graphe de scène, un nœud correspondant à une représentation d'un élément.

Le processus de résolution de règles permettant la création du graphe de scène est schématisé en fig. 5.11. Il y a deux phases dans ce processus : la détermination des représentations pour chaque élément et la construction du graphe de scène. Dans la suite de cette section, j'utilise les notations suivantes :

- T une tuile ;
- F un objet ;
- H la hiérarchie contenant toutes les tuiles et tous les objets ;

- $p(T)$ la profondeur (ou niveau d'échelle) d'une tuile dans la hiérarchie ;
- ER l'ensemble de règles ;
- RT une règle de tuile ;
- RF une règle d'objet ;
- $C(Rx, x)$ les conditions d'une règle appliquées à un élément x ;
- $A(Rx)$ l'action d'une règle ;
- $r(A, p)$ la liste de représentation d'une action pour le niveau d'échelle p ;
- $e(x)$ l'empreinte 2D d'un élément.

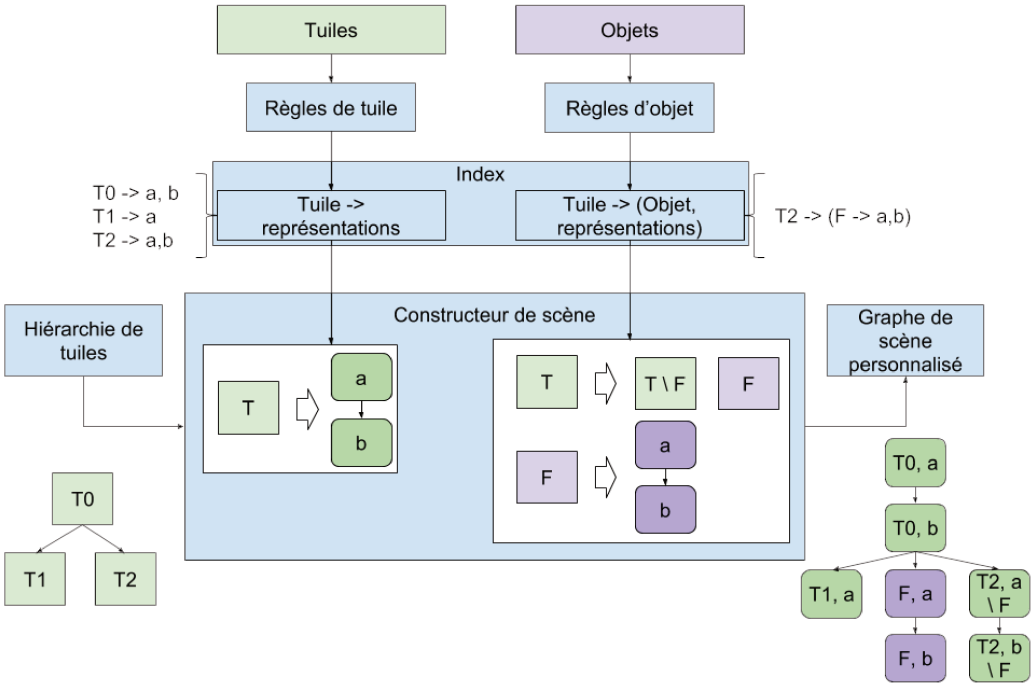

Figure 5.11 Le processus de création d'un graphe de scène personnalisé. Les règles sont appliquées aux tuiles et aux objets pour former les index. La hiérarchie de tuiles est parcourue et chaque tuile est transformée, en fonction des index, en une suite de nœuds, un par représentation. Une fois toutes les tuiles traitées, on obtient le graphe de scène personnalisé.

La première étape consiste à trouver quelle action appliquer à chaque tuile ou objet. Pour cela, il faut tester les conditions de chaque règle sur chacun de ces éléments. Cette étape est fortement accélérée par une base de données géospatiales qui permet de filtrer très rapidement ces éléments. Le résultat de ces tests est stocké dans deux index, un pour les règles de tuile IT, l'autre pour les règles d'objet IF.

IT est un index qui associe à chaque tuile une liste ordonnée des représentations qu'elle doit prendre. Pour ajouter une tuile à cet index, chaque règle de tuile est testée sur la tuile. Seule l'action de la première règle dont les conditions sont validées par la tuile est considérée. Si aucune règle ne correspond à la tuile, on utilise l'action par défaut. La liste de représentation de l'action choisie correspondant au niveau d'échelle de la tuile traitée est alors associée à la tuile dans l'index. La création de cet index est décrite plus formellement par l'algorithme 5.

Algorithme 5 Algorithme de création de l'index de tuiles.

```
for T in H do
    representations ← r(Defaut, p(T))
    for RT in ER do
        if C(RT, T) = True then
            representations ← r(A(RT), p(T))
            break
    IT(T) ← (T, representations)
```

IF est un index qui associe à chaque tuile un ensemble d'objets liés à une liste ordonnée de représentations. Cette fois-ci, ce sont les objets qui sont testées. Si un objet valide les conditions d'un règle, alors pour chacune des tuiles qui le contiennent (ces ancêtres dans la hiérarchie) l'action de la règle est appliquée. Si l'action définie une liste de représentations pour le niveau d'échelle de la tuile, l'index associe à la tuile un tuple constitué de l'objet et de cette liste de représentations. Une tuile peut être associée à plusieurs de ces tuples. La création de cet index est décrite plus formellement par l'algorithme 6.

Algorithme 6 Algorithme de création de l'index d'objets.

```
for F in H do
    for RF in ER do
        if C(RF, F) = True then
            for T in ancêtres(F) do
                if p(T) in A(RF) then
                    representations ← r(A(RF), p(T))
                    IF(T) ← (T, (F, representations))
                break
```

La seconde étape est la construction de la scène, qui consiste en un parcours de la hiérarchie et la génération de chacun des nœuds de l'arbre final à partir des index précédemment créés (algorithme 7, fig. 5.12). Pour chaque tuile visitée, on récupère la liste de représentations à partir de l'index de tuile IT. Si les représentations demandées existent pour la tuile, un nœud est créé pour chacune d'entre elles. Ensuite, on vérifie si des objets sont liés à la tuile dans l'index d'objets IF. Si c'est le cas, cela signifie qu'il faut soustraire ces objets à la tuile : on marque les représentations précédemment créées comme devant subir cette soustraction. Ensuite, un nœud est créé pour toutes les représentations de ces objets.

Lorsque les nœuds de chaque tuile ont été générés, ils sont reliés entre eux pour créer le graphe de scène final (fig. 5.13). Les liens sont créés de telles sortes que le dernier nœud d'une tuile soit le parent des premiers nœuds de chaque tuile fille.

Dans certains cas, seulement un sous-ensemble des enfants d'une tuile auront une représentation dans le graphe de scène personnalisé (par exemple, si les tuiles à une certaine profondeur sont chargées uniquement lorsqu'une règle est validée), comme

Algorithme 7 Algorithme de génération des nœuds du graphe de scène.

```
for T in H do
    noeuds(T) ← ∅
    objets_à_soustraire ← ∅
    for F, representations in IF(T) do
        objets_à_soustraire ← F
        liste_noeuds ← ∅
        for r in representations do
            noeud ← créer_noeud(F, r)
            insérer noeud en bout de liste_noeuds
        noeuds(T) ← liste_noeuds
    liste_noeuds ← ∅
    for representations in IT(T) do
        noeud ← créer_noeud(T, r)
        insérer noeud en bout de liste_noeuds
    noeuds(T) ← liste_noeuds
```

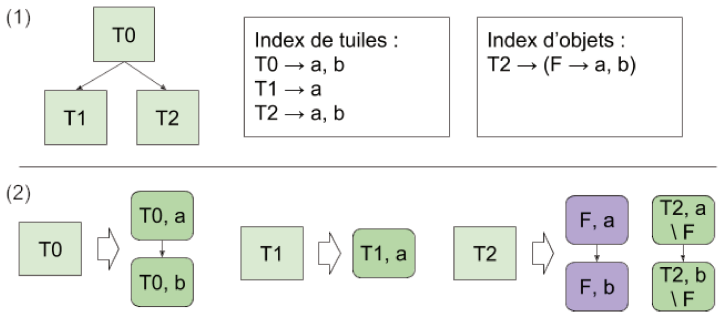

Figure 5.12 À partir d'une hiérarchie de tuiles et des index de tuiles et d'objets (1), chaque tuile est transformée en listes de nœuds (2). En vert, les tuiles et leurs nœuds, en violet, les objets et leurs nœuds. Les rectangles arrondis sont les nœuds générés.

illustré par la fig. 5.14. Cela signifie que lors du raffinement de la tuile, la tuile mère sera remplacée par une partie de ces enfants, et donc que l'autre partie disparaîtra. Pour compenser cela, il faut ajouter un nœud dont le contenu est le résultat de la soustraction des empreintes des tuiles filles manquantes à la représentation de la tuile mère.

5.2.3 Transfert et génération de tuiles

Le processus de transfert de données entre le client est le serveur est identique à celui de 3D Tiles ou i3s, permettant l'interopérabilité entre le serveur et tout client supportant ce genre de format. La différence par rapport aux méthodes de l'état de l'art est qu'au lieu de récupérer un tileset qui existe sous la forme d'un fichier, le client envoie une requête paramétrée contenant une description des règles de personnalisation. Le serveur génère alors le tileset comme indiqué précédemment.

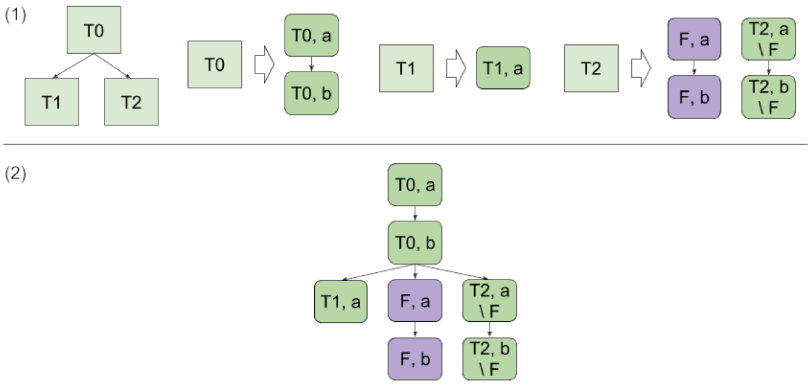

Figure 5.13 Les nœuds issus de chaque tuile de la hiérarchie (1) sont reliés entre eux pour former le graphe de scène (2).

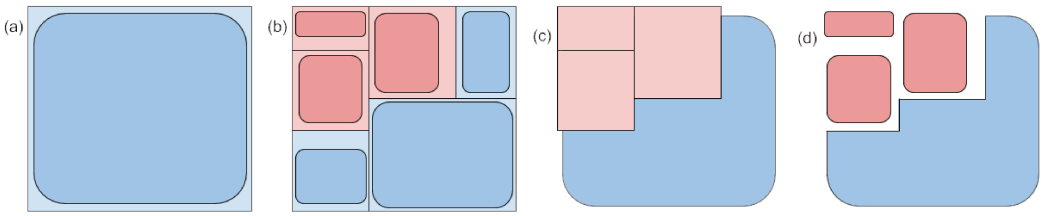

Figure 5.14 Illustration de la nécessité de créer un nœud compensatoire lorsque toutes les tuiles filles d'une tuile ne sont pas conservées. En couleurs foncés, les représentations des tuiles, en couleurs claires, les empreintes des tuiles. (a) La tuile mère. (b) Les tuiles filles, seules les tuiles rouges sont conservées dans le graphe de scène personnalisé. (c) Création du contenu du nœud compensatoire par la soustraction de l'empreint des tuiles rouges à la représentation de la tuile mère. (d) Tous les nœuds fils : les nœuds issus des tuiles rouges, et le nœud compensatoire.

Le lien vers le contenu de chaque tuile du tileset est une requête permettant de récupérer ou de générer à la volée les données attendues.

Trois types de requêtes différentes doivent pouvoir être réalisées par le serveur pour générer toutes les données décrites par le graphe de scènes :

1. récupérer une tuile avec une représentation particulière ;
2. récupérer un objet avec une représentation particulière ;

Algorithme 8 Algorithme d'ajout de nœud compensant l'absence de certaines tuiles filles.

$T_{manquantes} \leftarrow \emptyset$
for T_f **in** T_{filles} **do**
  if $noeuds(T_f) \leftarrow \emptyset$ **then**
    $T_{manquantes} \leftarrow T_f$
if $T_{manquantes} \neq T_{filles}$ **then**
  $noeud\_compensatoire \leftarrow créer\_noeud(T, r \setminus e(T_{manquantes}))$

3. découper la représentation d'une tuile pour en soustraire un ensemble d'objets et/ou de tuiles.

Les deux premières opérations sont triviales — il suffit d'aller chercher les bonnes données au bon endroit. La dernière est un peu plus complexe. Tout d'abord, ce ne sont pas des données qui peuvent être préparées à l'avance car trop de combinaisons de paramètres existent. Il faut donc faire des opérations spatiales, des soustractions, à la volée lors de la réception de ce type de requêtes. Des fonctions pour réaliser ce genre d'opérations sont fournies par les systèmes de gestion de base de données géospatiales, tels que PostGIS.

Soit $\{E_{soustrait}\}$ l'ensemble des éléments, tuiles ou objets, à soustraire d'une tuile. La requête (3) cherche à générer la géométrie résultante de cette opération $r(T \setminus \{E_{soustrait}\})$. Pour obtenir cette géométrie, on soustrait à la représentation $r(T)$ de la tuile T les empreintes des éléments $\{E_{soustrait}\}$.

$$r(T \setminus \{E_{soustrait}\}) = r(T) \setminus (\cup\{e(E_{soustrait})\})$$

5.2.4 Résultats

Un prototype de serveur a été implémentée en python, allié à PostGIS et le plug-in SFCGAL pour gérer le stockage des éléments et les opérations spatiales. La spécification 3D Tiles a été utilisée pour formater le graphe de scène et les géométries. Le résultat a été visualisé à l'aide du client web iTowns.

Mes tests ont été réalisés sur deux jeux de données :

- Lyon (14851 bâtiments, 107 km²) ;
- Helsinki (77231 bâtiments, 509 km²).

Le support pour trois différents types de données a été réalisé : les maillages, les maillages texturés et les données vectorielles 2.5D (des polygones extrudés).

Différents ensembles de règles ont été testés pour chaque ville. Pour Lyon :

- maillages et maillages texturés pour certains bâtiments spécifiques (fig. 5.15 haut) ;
- maillages de bâtiment dans le cercle rouge, bâtiment 2.5D généralisés dans le cercle bleu, et tuiles généralisés 2.5D à l'extérieur des cercles (fig. 5.15 bas).

Pour Helsinki :

- généralisation 2.5D de bâtiments (fig. 5.16 haut) ;
- généralisation 2.5D de quartiers et maillages pour les bâtiments dans la zone le long de la côte délimitée par les pointillés rouges (fig. 5.16 centre) ;
- généralisation 2.5D de quartiers et maillages pour les bâtiments plus haut que 60 mètres (fig. 5.16 bas).

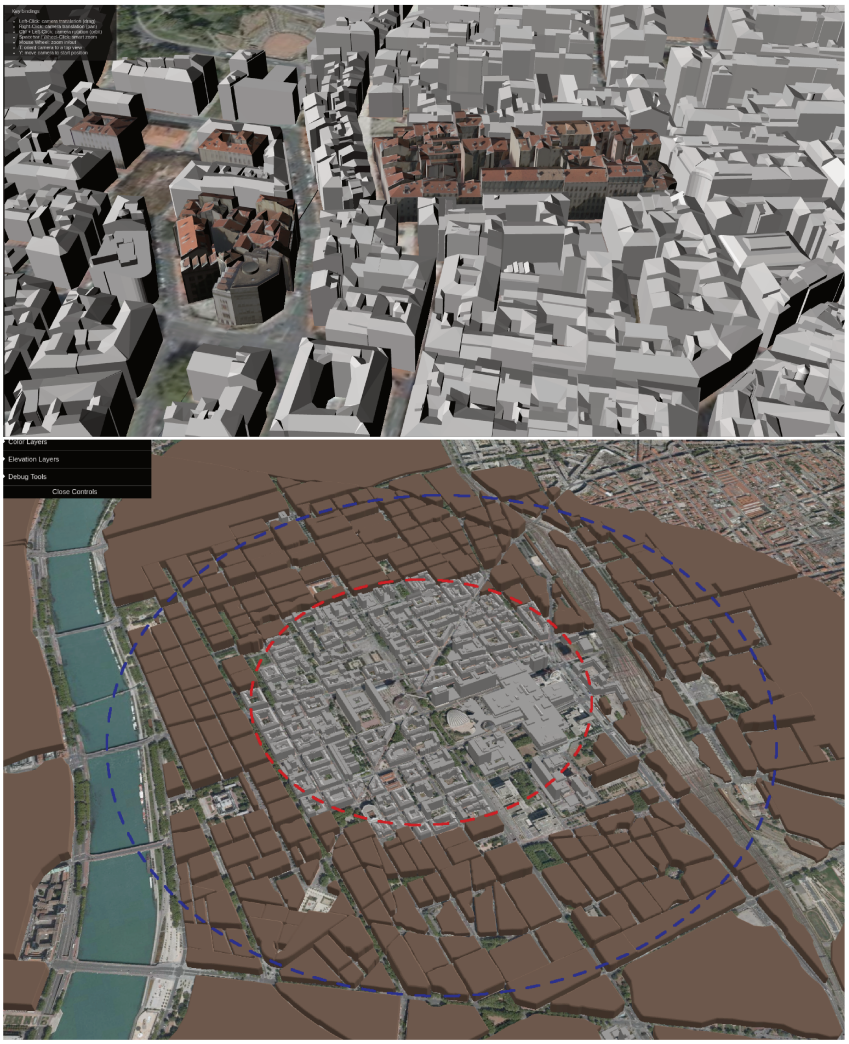

Figure 5.15 Texturation de certains bâtiments spécifiques de Lyon (haut) et augmentation progressive des détails des représentations (bas).

Les performances de cette implémentation ont été mesurées sur un ordinateur disposant d'un processeur Intel© Core™ i5-4590 CPU @ 3.3GHz x 4 et une carte graphique Nvidia GeForce GTX 970. Les trois phases telles que présentées dans le chapitre 3 ont été mesurées :

- la phase de pré-traitement a pris 1389 secondes (1332 secondes pour la segmentation des tuiles, 57 secondes pour l'import de la hiérarchie et l'indexation) pour Lyon et 1402 secondes (961 secondes et 441 seconds) pour Helsinki ;

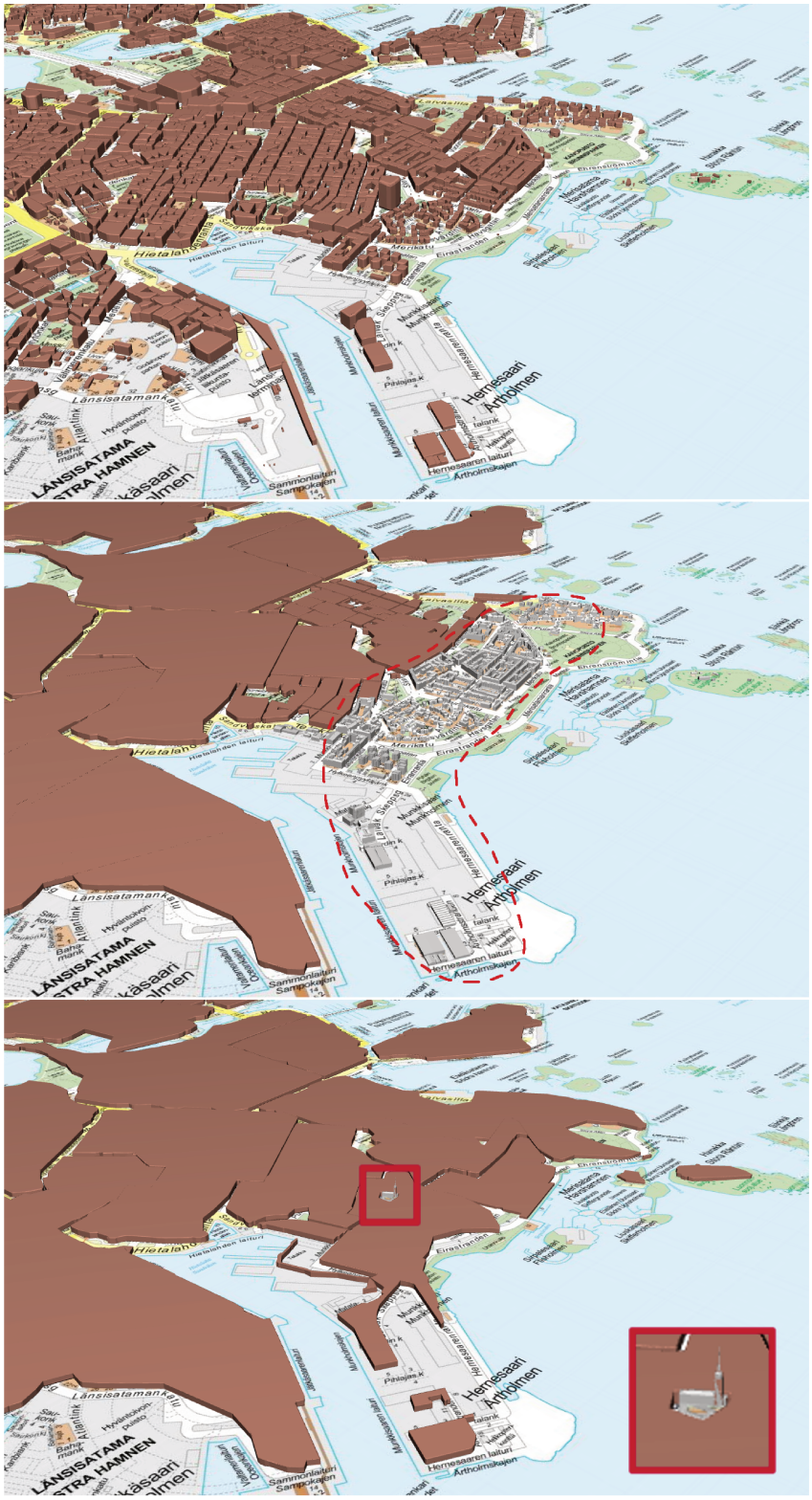

Figure 5.16 Trois représentations personnalisées de Helsinki à partir du même point de vue.

- la phase d'initialisation a pris 0.35 seconde en moyenne pour Lyon et 0.7 seconde en moyenne pour Helsinki ;

- lors de la phase de transfert, une tuile prenait en moyenne 25 millisecondes à générer pour Lyon et 10 millisecondes pour Helsinki.

On constate que ma méthode remplit ses deux objectifs initiaux : le choix des représentations des éléments de la scène et la mise en valeur d'objets individuels. L'utilisateur a à sa disposition des fonctions complexes (opérations spatiales, conditions sur les attributs sémantiques) pour définir les représentations. Les objets individuels important sont bien mis en avant, grâce au fait qu'ils soient plus haut dans la hiérarchie et grâce à la zone découpée dans la tuile généralisée.

Limites Une des limites de ces travaux est que le contrôle du raffinement des tuiles (à partir de quelle distance une tuile est raffinée, dans le cas 3D Tiles, c'est la valeur de l'erreur géométrique) n'a pas été suffisamment étudié. Pour le moment, le seuil de raffinement est fixe pour chaque niveau d'échelle. Ce n'est pas forcément ni le meilleur comportement, ni celui souhaité par l'utilisateur. Peut-être que permettre le contrôle de ce paramètre à travers les règles serait une bonne idée. La différence de traitement entre tuiles et objets gagnerait également peut-être à être effacée au profit d'un traitement plus générique des données.

Certaines améliorations pourrait être apportées à cette méthode. L'ajout de nouveaux comportements tout d'abord, comme le choix de la représentation basée sur leur existence ou non pour une tuile (si telle représentation n'est pas disponible, utiliser telle autre). Le fait que chaque représentation soit chargée avec un seul nœud est regrettable, certaines représentations volumineuse (comme un objet représenté par un nuage de points) gagnerait a être chargées en plusieurs étapes, donc à l'aide d'une suite de nœud en raffinement additif (fig. 5.17).

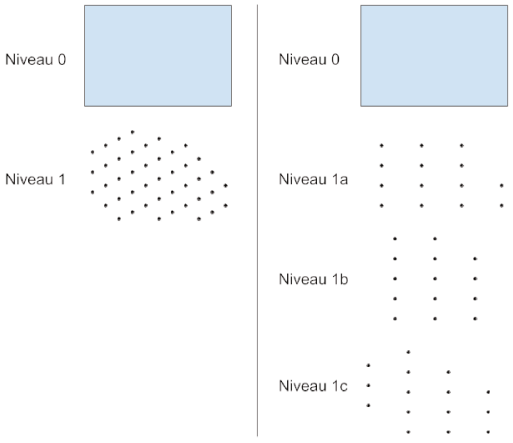

Figure 5.17 Tuile volumineuse découpée en trois tuiles distinctes. À gauche, le comportement actuel. La tuile de niveau 0 est remplacée par la tuile volumineuse de niveau 1. À droite, une extension de ma méthode. La tuile de niveau 0 est remplacée par la tuile de niveau 1a, auquel est ajoutée la tuile de niveau 1b, puis celle de niveau 1c.

5.3 Synthèse

J'ai présenté dans ce chapitre des méthodes pour améliorer l'expérience utilisateur en adaptant le chargement des données au cas d'utilisation, à travers la priorisation du chargement ou la personnalisation du graphe de scène transmis. Les résultats obtenus démontrent que les données d'intérêt pour l'utilisateur lui parviennent plus rapidement par l'application de ces méthodes.

Les méthodes présentées dans cette partie permettent de gagner en efficacité et aide à améliorer la clarté lors de la visualisation de jeux de données volumineux. Les solutions proposées sont également suffisamment paramétrables pour s'adapter aux nombreux cas d'utilisation du domaine.

Une piste d'amélioration intéressante à apporter est de rendre plus accessibles au grand publique ces approches. En effet, le but de la personnalisation reste d'améliorer l'expérience utilisateur, mais l'utilisateur n'a pas de moyen simple de définir ses besoins et doit, dans l'état actuel de mes recherches, définir des paramètres complexes. Alors qu'il est facile de choisir quelles couches de données afficher, il est bien plus compliqué de définir des règles de personnalisation ou des fonction de poids. Une perspective intéressante serait de créer une surcouche à mes méthodes, qui définirait les paramètres appropriés à partir d'une interface compréhensible par le plus grand nombre.

6 Conclusion

Le partage, la visualisation et l'édition de données géolocalisées 3D constituent des usages qui sont difficiles à mettre en œuvre, d'autant plus lorsque certains objectifs de ma thèses entrent en antagonisme. Une visualisation efficace de données 3D est rendue plus compliquée par le souhait de conserver la possibilité d'interagir avec ces données. Gérer des données complexes et volumineuses est d'autant plus compliqué quand on se place dans un contexte web.

Pour accomplir mes objectifs, il a fallu que je m'attaque à plusieurs problématiques :

- la prise en compte de l'aspect multi-échelle de la ville ;
- l'adaptation de la représentation de la ville à l'utilisateur ;
- la possibilité de modifier la ville et de changer sa représentation de manière interactive.

Pour résoudre ces problématiques, j'ai du surmonter plusieurs verrous technologiques et scientifiques :

- la grande volumétrie des données ;
- l'édition et l'interrogation des données géométriques et sémantiques ;
- les multiples représentations des objets géographiques ;
- l'hétérogénéité des données de la ville.

Un état de l'art riche existait au commencement de ma thèse, celle-ci étant de nature transverse et reposant sur plusieurs domaines (géomatique, 3D, Web notamment). J'ai dû utiliser au mieux ses méthodes pour parvenir à mes objectifs, et les enrichir lorsqu'elles ne suffisaient pas.

Les contributions de ma thèses s'articulent autour de trois axes, présentés en détails dans les chapitres précédents : l'architecture et le développement d'une application de géovisualisation, la conception de structures de données adaptées aux données géospatiales et la proposition d'un ensemble de méthodes de personnalisation et de priorisation afin d'apporter une meilleure visualisation des données.

Dans le cadre de ma thèse CIFRE, j'ai mis en place une architectures de géovisualisation basée sur des projets open source, dont certains auxquels j'ai contribué. Deux implémentations ont été testées côté serveur, la première offrant de très bonnes performances, mais avec des données statiques. La seconde, s'inscrivant plus dans la philosophie de ma thèse, repose sur une base de données géospatiale et troque une partie de sa vitesse d'exécution contre une forte interactivité avec les données stockées . Une bibliothèque a été développée pour convertir les données de la base de données en un format en cours de standardisation, 3D Tiles, ce qui garantit l'in-

teropérabilité du serveur. Côté client, iTowns permet l'utilisation d'un grand nombre de protocoles et de types de données différents, répondant à notre besoin de gérer l'hétérogénéité des données de la ville.

J'ai proposé plusieurs structures de données pour organiser les objets géographiques de la ville. Cette organisation des objets géographiques en tuiles permet de réduire la quantité de données à traiter simultanément par les applications et donc de rendre exploitables et visualisables de larges volumes de données. Ces structures ne se basent pas sur des présuppositions sur les types de données afin de gérer l'hétérogénéité des données. Ceci est particulièrement le cas pour la structure en grille, qui est capable de gérer des objets pouvant s'étendre sur de vastes surfaces. Les deux autres structures, grâce à leur caractère hiérarchique, sont adaptés pour visualiser la ville à tout niveau d'échelle. La structure de données basée sur un quadtree permet l'édition interactives des données qu'elle contient. Enfin, la structure basée sur un découpage sémantique de l'espace est capable de gérer les multiples représentations des objets géographiques et peut servir de base pour créer des généralisations plus fines.

J'ai proposé des méthodes de personnalisations et de priorisations. Les méthodes de priorisations réduisent le temps qu'il faut pour que l'utilisateur reçoive les données pertinentes du jeu de données, potentiellement très volumineux, qu'il est en train de manipuler. La méthode de personnalisation permet à un utilisateur de définir précisément la représentation des données qu'il souhaite recueillir à l'aide d'un langage de règles.

À travers mes recherches, j'ai pu apporter des réponses aux problématiques de ma thèse. La plupart de mes travaux ont été reversés à la communauté à travers ma participation à un ensemble de projets open source : iTowns, py3dtiles et building-server (le serveur implémentant mes différentes méthodes). Les solutions que j'ai proposées sont abouties : Oslandia a pu exploiter certains de mes travaux à travers des projets clients ou des démonstrations. Par exemple, l'application présentée fig. 6.1 a utilisé mon serveur de données, mon exportateur 3D Tiles et iTowns pour servir des donnés cartographiques à l'échelle de la Bretagne. Outre l'aspect industriel, deux articles ont été présentées dans des conférences internationales [Gai+15] [Gai+16] et un troisième est en préparation.

Il reste encore de nombreux défis dans le domaine de l'information géographique 3D sur le Web. Je présente ci-après certaines pistes qui me paraissent intéressantes à explorer dans le futur.

Tout d'abord, l'aspect multi-échelle pourrait être développé davantage. Dans les méthodes que j'ai développées, on peut se déplacer de manière fluide du niveau

Figure 6.1 Une application développée par Oslandia permettant de naviguer sur une carte comprenant réseaux de transport, rivières et bâtiments de Bretagne. Ici, le centre-ville de Rennes.

de la ville au niveau de la rue. Il est possible d'étendre l'échelle dans les deux sens. La transition entre le modèle de la ville et le modèle du bâtiment (*BIM, Building Information Modeling*) permettrait d'avoir sur la même maquette les informations urbaines et les informations architecturales. L'aspect 3D est bien plus présent au niveau du bâtiment, ce qui demande de nouvelles manières d'organiser les données. L'échelle peut être étendue dans l'autre sens, c'est-à-dire de passer de l'échelle de la ville à celle du pays ou du monde. Ici, la situation est inverse : l'aspect 3D est négligeable à cette échelle. Il faut donc réfléchir à des manières convenablement de représenter la donnée à cette échelle, soit en reprenant les codes de la cartographie (taches urbaines, etc.), soit en utilisant des représentations abstraites pour mettre en valeur les lieux où des données se situent. Ma structure basée sur un découpage sémantique paraît adaptée pour ces deux échelles, grâce à la possibilité de contrôler finement la représentation à chaque niveau d'échelle, bien que quelques ajustements soient nécessaires pour prendre en compte les contraintes exposées précédemment.

Pour améliorer les méthodes de découpe de l'espace, il serait intéressant de concevoir des indicateurs statistiques sur les géométries et leur répartition dans l'espace. Le calcul de ces indicateurs pourrait servir de base pour organiser au mieux les données, voire à les organiser en fonction de l'usage qui est prévu.

La manière dont les niveaux de détails sont gérés peut faire l'objet de nouvelles recherches. Pour le moment, il n'y a pas de transition entre les différents niveaux de détails dans mes travaux : le niveau précédent disparaît, le niveau suivant apparaît, ce qui cause des « sauts » lors du chargement. Pour éviter ce phénomène, on pourrait trouver une fonction d'interpolation entre deux niveaux de détails, ou passer par des niveaux de détails continus, mais cela complique considérablement la création de ces

niveaux de détails. [OM14] propose une piste intéressante, en considérant l'échelle comme une dimension à part entière et pourrait servir de base à ces recherches.

Les données géographiques n'ont pas que des dimensions spatiales. La dimension temporelle peut également être définie : les objets géographiques peuvent avoir une date de construction, de destruction ou avoir subi des modifications à certaines dates précises. Cette nouvelle dimension nécessite d'étendre les structures de données et d'enrichir les standards pour la prendre en compte. Certains travaux se sont déjà attelés au sujet et ont ajouté l'aspect temporel à CityGML [Cha+17]. Les dimensions thématiques des objets sont également intéressantes à investiguer. Les informations thématiques peuvent être représentées sous la forme de dimensions supplémentaires pour les objets géographiques, de telle sorte que l'on se retrouve à gérer un espace n-dimensionnel. La navigation entre ces couches serait une autre manière de gérer la personnalisation d'une scène 3D.

Finalement, la gestion de l'aspect multimodèle des données pourrait être améliorée. Pour le moment, chaque modèle est considéré indépendamment. Faire interagir les différents modèles d'un même objet pourrait enrichir la visualisation, à la manière de [DB16] qui combine nuages de points, maillages et textures pour améliorer une application de *street view*.

Contributions

Contributions académiques

Jérémy GAILLARD, Alexandre VIENNE, Rémi BAUME, Frédéric PEDRINIS, Adrien PEYTAVIE et Gilles GESQUIÈRE. « Urban Data Visualisation in a Web Browser ». In : *Proceedings of the 20th International Conference on 3D Web Technology*. Web3D '15. Heraklion, Crete, Greece : ACM, 2015, p. 81–88. https://hal.archives-ouvertes.fr/hal-01196834

Jérémy GAILLARD, Adrien PEYTAVIE et Gilles GESQUIÈRE. « A Data Structure for Progressive Visualisation and Edition of Vectorial Geospatial Data ». In : *3D GeoInfo*. T. 2. Athènes, Greece, oct. 2016, p. 201 –209. https://hal.archives-ouvertes.fr/hal-01420117

Jérémy GAILLARD, Adrien PEYTAVIE et Gilles GESQUIÈRE. « Visualisation and Personalisation of Multi-representations City Modles ». Soumission en cours.

Contributions industrielles

iTowns https://github.com/iTowns/itowns : support 3D Tiles, corrections de bugs, amélioration de l'architecture, participation à la vie du projet à travers des *code reviews*.

py3dtiles https://github.com/Oslandia/py3dtiles/ : refonte et ajout du support des maillages.

building-server https://github.com/Oslandia/building-server : totalité du projet. Implémentations des méthodes d'organisation des données et de personnalisation présentées dans ce manuscrit.

Crédits

Les données OpenStreetMap utilisées pour illustrer ce manuscrit sont disponibles sous la licence ODbL et la carte sous licence CC-BY-SA https://www.openstreetmap.org/copyright © les contributeurs d'OpenStreetMap.

Les données de Lyon sont sous licence ouverte et mises à disposition par la Métropole de Lyon (https://data.grandlyon.com). Leur dernière mise à jour date du 27 janvier 2015.

Les données de Montréal sont sous licence Creative Commons CC-BY 4.0 et ont été mises à disposition par la Ville de Montréal (http://donnees.ville.montreal.qc.ca/).

Les données d'Helsinki sont sous licence Creative Commons CC-BY 4.0 et ont été mises à disposition par Helsinki Region Infoshare (https://hri.fi/).

Licence Creative Commons CC-BY 4.0 https://creativecommons.org/licenses/by/4.0/.

Licence Creative Commons CC-BY-SA https://creativecommons.org/licenses/by-sa/2.0/.

Bibliographie

[AM+08] Tomas AKENINE-MOLLER, Eric HAINES et Naty HOFFMAN. *Real-Time Rendering*. 3rd. Natick, MA, USA : A. K. Peters, Ltd., 2008 (Cité page 20).

[And05] Karl-Heinrich ANDERS. « Level of detail generation of 3D building groups by aggregation and typification ». In : *International Cartographic Conference*. T. 2. 2005 (Cité pages 35, 36).

[BC07] Dirk BURGHARDT et Alessandro CECCONI. « Mesh simplification for building typification ». In : *International Journal of Geographical Information Science* 21.3 (2007), p. 283–298 (Cité page 36).

[Bea06] Jeff de la BEAUJARDIERE. *Web Map Service*. Standard 06-042. Open Geospatial Consortium, 2006 (Cité page 33).

[Bec+11] Thomas BECKER, Claus. NAGEL et Thomas H. KOLBE. « Integrated 3D Modeling of Multi-utility Networks and Their Interdependencies for Critical Infrastructure Analysis ». In : *Advances in 3D Geo-Information Sciences*. Sous la dir. de Thomas H. KOLBE, Gerhard KÖNIG et Claus NAGEL. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011, p. 1–20 (Cité page 33).

[Beh+09] Johannes BEHR, Peter ESCHLER, Yvonne JUNG et Michael ZÖLLNER. « X3DOM : A DOM-based HTML5/X3D Integration Model ». In : *Proceedings of the 14th International Conference on 3D Web Technology*. Web3D '09. Darmstadt, Germany : ACM, 2009, p. 127–135 (Cité page 31).

[Ben75] Jon Louis BENTLEY. « Multidimensional Binary Search Trees Used for Associative Searching ». In : *Commun. ACM* 18.9 (sept. 1975), p. 509–517 (Cité page 25).

[Bil+15] Filip BILJECKI, Jantien STOTER, Hugo LEDOUX, Sisi ZLATANOVA et Arzu ÇÖLTEKIN. « Applications of 3D City Models : State of the Art Review ». In : *ISPRS International Journal of Geo-Information* 4.4 (2015), p. 2842 (Cité pages 2, 32).

[Bil17] Filip BILJECKI. « Level of detail in 3D city models ». Thèse de doct. Delft, the Netherlands : Delft University of Technology, mai 2017 (Cité page 34).

[Bra+15] Mickaël BRASEBIN, Elodie BUARD, Sidonie CHRISTOPHE et Florian PELLOIE. « A knowledge base to classify and mix 3D rendering styles ». In : *27th International Cartographic Conference (ICC'15), Rio de Janeiro, Brazil*. Août 2015 (Cité page 39).

[Bra+16] Mickaël Brasebin, Sidonie Christophe, Florence Jacquinod, Anouk Vinesse et Hortense Mahon. « 3D Geovisualization & stylization to manage comprehensive and participative Local Urban Plans ». In : *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-2/W1 (2016), p. 83–91 (Cité page 39).

[Bur08] Earl F. Burkholder. *The 3-D Global Spatial Data Model : Foundation of the Spatial Data Infrastructure*. CRC Press, 2008 (Cité page 6).

[Cel14] Fabien Cellier. « Modélisation et calcul parallèle pour le Web SIG 3D ». working paper or preprint. Thèse de doct. Jan. 2014 (Cité page 38).

[Cha+08] Remco Chang, Thomas Butkiewicz, Caroline Ziemkiewicz, Zachary Wartell, Nancy Pollard et William Ribarsky. « Legible Simplification of Textured Urban Models ». In : *IEEE Computer Graphics and Applications* 28.3 (2008), p. 27–36 (Cité page 36).

[Cha+15] Kanishk Chaturvedi, Zhihang Yao et Thomas H. Kolbe. « Web-based Exploration of and Interaction with Large and Deeply Structured Semantic 3D City Models using HTML5 and WebGL ». In : *Bridging Scales - Skalenübergreifende Nah- und Fernerkundungsmethoden, 35. Wissenschaftlich-Technische Jahrestagung der DGPF*. 2015 (Cité page 39).

[Cha+17] Kanishk Chaturvedi, Carl Stephen Smyth, Gilles Gesquière, Tatjana Kutzner et Thomas H. Kolbe. « Managing Versions and History Within Semantic 3D City Models for the Next Generation of CityGML ». In : *Lecture Notes in Geoinformation and Cartography*. Sous la dir. d'Alias Abdul-Rahman. Advances in 3D Geoinformation. Springer, 2017, p. 191 –206 (Cité page 117).

[Cha14] Kanishk Chaturvedi. « Web based 3D analysis and visualization using HTML5 and WebGL ». Mém.de mast. ITC, IIRS, 2014 (Cité pages 33, 40).

[Cho57] Noam Chomsky. *Syntactic Structures*. Mouton, 1957 (Cité page 101).

[Chu12] Won Chun. « WebGL Models : End-to-End ». In : *OpenGL Insights*. Sous la dir. de Patrick Cozzi et Christophe Riccio. http://www.openglinsights.com/. CRC Press, juil. 2012, p. 431–453 (Cité pages 29, 30).

[Cla76] James H. Clark. « Hierarchical Geometric Models for Visible Surface Algorithms ». In : *Commun. ACM* 19.10 (oct. 1976), p. 547–554 (Cité pages 20, 23).

[CR11] Patrick Cozzi et Kevin Ring. *3D Engine Design for Virtual Globes*. 1st. http://www.virtualglobebook.com. CRC Press, juin 2011 (Cité page 37).

[Cur16] Rémi Cura. « Inverse procedural Street Modelling : from interactive to automatic reconstruction ». Thèse de doct. 2016 (Cité page 19).

[DB16] Alexandre Devaux et Mathieu Brédif. « Realtime Projective Multi-texturing of Pointclouds and Meshes for a Realistic Street-view Web Navigation ». In : *Proceedings of the 21st International Conference on Web3D Technology*. Web3D '16. Anaheim, California : ACM, 2016, p. 105–108 (Cité page 117).

[DC98] Allan Doyle et Adrian Cuthbert. *Essential Model of Interactive Portrayal*. Rapp. tech. 98-061. OpenGIS, nov. 1998 (Cité page 44).

[Dei+17] David Deibe, Margarita Amor, Ramón Doallo, David Miranda et Miguel Cordero. « GVLiDAR : an interactive web-based visualization framework to support geospatial measures on lidar data ». In : 38 (fév. 2017), p. 827–849 (Cité page 40).

[EA14] Claire Ellul et Julia Altenbuchner. « Investigating approaches to improving rendering performance of 3D city models on mobile devices ». In : *Geo-spatial Information Science* 17.2 (2014), p. 73–84 (Cité page 38).

[Eri+01] Carl Erikson, Dinesh Manocha et William V. Baxter III. « HLODs for Faster Display of Large Static and Dynamic Environments ». In : *Proceedings of the 2001 Symposium on Interactive 3D Graphics*. I3D '01. ACM, 2001, p. 111–120 (Cité page 26).

[Eva+14a] Alun Evans, Marco Romeo, Arash Bahrehmand, Javi Agenjo et Josep Blat. « 3D graphics on the web : A survey ». In : *Computers & Graphics* 41.0 (2014), p. 43 –61 (Cité pages 29, 32).

[Eva+14b] Alun Evans, Javi Agenjo et Josep Blat. « Web-based Visualisation of On-set Point Cloud Data ». In : *Proceedings of the 11th European Conference on Visual Media Production*. CVMP '14. London, United Kingdom : ACM, 2014, 10 :1–10 :8 (Cité page 30).

[Eva+15] Alun Evans, Javi Agenjo et Josep Blat. « Hybrid Visualisation of Digital Production Big Data ». In : *Proceedings of the 20th International Conference on 3D Web Technology*. Web3D '15. Heraklion, Crete, Greece : ACM, 2015, p. 69–72 (Cité page 38).

[Fan+09] Hongchao Fan, Liqiu Meng et Mathias Jahnke. « Generalization of 3D Buildings Modelled by CityGML ». In : *Advances in GIScience : Proceedings of the 12th AGILE Conference*. Sous la dir. de Monika Sester, Lars Bernard et Volker Paelke. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009, p. 387–405 (Cité page 35).

[Fil+16] Yevgeniya Filippovska, Andreas Wichmann et Martin Kada. « Space Partitioning for Privacy Enables 3D City Models ». In : *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2/W2 (2016), p. 17–22 (Cité pages 39, 98).

[FO98] Petr Felkel et Stepan Obdrzalek. « Straight skeleton implementation ». In : *Proceedings of spring conference on computer graphics*. Citeseer. 1998 (Cité page 98).

[For07] Andrea Forberg. « Generalization of 3D building data based on a scale-space approach ». In : *ISPRS Journal of Photogrammetry and Remote Sensing* 62.2 (2007). Including Special Section : p. 104 –111 (Cité page 35).

[FS93] Thomas A. Funkhouser et Carlo H. Séquin. « Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments ». In : *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. Anaheim, CA : ACM, 1993, p. 247–254 (Cité page 26).

[Gai+] Jérémy Gaillard, Adrien Peytavie et Gilles Gesquière. « Visualisation and Personalisation of Multi-representations City Modles ». Soumission en cours (Cité page 118).

[Gai+15] Jérémy GAILLARD, Alexandre VIENNE, Rémi BAUME, Frédéric PEDRINIS, Adrien PEYTAVIE et Gilles GESQUIÈRE. « Urban Data Visualisation in a Web Browser ». In : *Proceedings of the 20th International Conference on 3D Web Technology*. Web3D '15. Heraklion, Crete, Greece : ACM, 2015, p. 81–88 (Cité pages 115, 118).

[Gai+16] Jérémy GAILLARD, Adrien PEYTAVIE et Gilles GESQUIÈRE. « A Data Structure for Progressive Visualisation and Edition of Vectorial Geospatial Data ». In : *3D GeoInfo*. T. 2. Athènes, Greece, oct. 2016, p. 201 –209 (Cité pages 115, 118).

[GD09] Tassilo GLANDER et Jürgen DÖLLNER. « Abstract representations for interactive visualization of virtual 3D city models ». In : *Computers, Environment and Urban Systems* 33.5 (2009). Geo-information Generalisation and Multiple Representation, p. 375 –387 (Cité pages 36, 37).

[Gio+14] Luca GIOVANNINI, Stefano PEZZI, Umberto DI STASO, Federico PRANDI et Raffaele de AMICIS. « Large-Scale Assessment and Visualization of the Energy Performance of Buildings with Ecomaps-Project SUNSHINE : Smart Urban Services for Higher Energy Efficiency. » In : *DATA*. 2014, p. 170–177 (Cité page 39).

[Gla+10] Tassilo GLANDER, Matthias TRAPP et Jürgen DÖLLNER. « 3D Isocontours – Real-time Generation and Visualization of 3D Stepped Terrain Models ». In : *Eurographics 2010 Shortpaper*. Sous la dir. de Stefan SEIPEL et Hendrik LENSCH. Norrköping, Sweden : The Eurographics Association, mai 2010, p. 17–20 (Cité page 39).

[Gla13] Tassilo GLANDER. « Multi-scale representations of virtual 3D city models ». doctoralthesis. Universität Potsdam, 2013 (Cité page 34).

[GM12] Gilles GESQUIERE et Alexis MANIN. « 3D Visualization of Urban Data Based on CityGML with WebGL ». en. In : *International Journal of 3-D Information Modeling* 3.1 (juil. 2012), p. 1–15 (Cité pages 38, 55).

[Gob+12] Enrico GOBBETTI, Fabio MARTON, Marcos Balsa RODRIGUEZ, Fabio GANOVELLI et Marco DI BENEDETTO. « Adaptive Quad Patches : An Adaptive Regular Structure for Web Distribution and Adaptive Rendering of 3D Models ». In : *Proceedings of the 17th International Conference on 3D Web Technology*. Web3D '12. Los Angeles, California : ACM, 2012, p. 9–16 (Cité page 30).

[Gue+11] Richard GUERCKE, Timo GÖTZELMANN, Claus BRENNER et Monika SESTER. « Aggregation of LoD 1 building models as an optimization problem ». In : *{ISPRS} Journal of Photogrammetry and Remote Sensing* 66.2 (2011). Quality, Scale and Analysis Aspects of Urban City Models, p. 209 –222 (Cité page 35).

[Gut+16] Ralf GUTBELL, Lars PANDIKOW, Volker COORS et Yasmina KAMMEYER. « A Framework for Server Side Rendering Using OGC's 3D Portrayal Service ». In : *Proceedings of the 21st International Conference on Web3D Technology*. Web3D '16. Anaheim, California : ACM, 2016, p. 137–146 (Cité page 34).

[Gut84] Antonin GUTTMAN. « R-trees : A Dynamic Index Structure for Spatial Searching ». In : *SIGMOD Rec.* 14.2 (juin 1984), p. 47–57 (Cité pages 24, 49).

[Haa+17] Andreas HAAS, Andreas ROSSBERG, Derek L. SCHUFF, Ben L. TITZER, Michael HOLMAN, Dan GOHMAN, Luke WAGNER, Alon ZAKAI et JF BASTIEN. « Bringing the Web Up to Speed with WebAssembly ». In : *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2017. Barcelona, Spain : ACM, 2017, p. 185–200 (Cité page 28).

[Hag+17] Benjamin HAGEDORN, Simon THUM, Thorsten REITZ, Volker COORS et Ralf GUTBELL. *3D Portrayal Service 1.0*. Standard 15-001r4. Open Geospatial Consortium, 2017 (Cité pages 34, 44).

[HD07] Benjamin HAGEDORN et Jürgen DÖLLNER. « High-level Web Service for 3D Building Information Visualization and Analysis ». In : *Proceedings of the 15th Annual ACM International Symposium on Advances in Geographic Information Systems*. GIS '07. Seattle, Washington : ACM, 2007, 8 :1–8 :8 (Cité page 39).

[He+12] Shuang HE, Guillaume MOREAU et J.Y. MARTIN. « Footprint-Based Generalization of 3D Building Groups at Medium Level of Detail for Multi-Scale Urban Visualization ». In : *International Journal on Advances in Software* 5.3&4 (déc. 2012), 378 to 388 (Cité page 36).

[He12] Shuang HE. « Production et visualisation de niveaux de détail pour les modèles 3D urbains ». Thèse de doctorat dirigée par Moreau, Guillaume Aménagement de l'espace, urbanisme Ecole centrale de Nantes 2012. Thèse de doct. École centrale de Nantes, 2012, 1 vol. (xi–148 p.) (Cité page 34).

[Her10] John R. HERRING. *OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 2 : SQL option*. Standard 06-104r4. Open Geospatial Consortium, 2010 (Cité page 49).

[Her11] John R. HERRING. *Simple Feature Access - Part 1 : Common Architecture*. Standard 06-103r4. Open Geospatial Consortium, 2011 (Cité pages 4, 49).

[Hop96] Hugues HOPPE. « Progressive Meshes ». In : *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. ACM, 1996, p. 99–108 (Cité page 26).

[Jan+13] Jacek JANKOWSKI, Sandy RESSLER, Kristian SONS, Yvonne JUNG, Johannes BEHR et Philipp SLUSALLEK. « Declarative Integration of Interactive 3D Graphics into the World-wide Web : Principles, Current Approaches, and Research Agenda ». In : *Proceedings of the 18th International Conference on 3D Web Technology*. Web3D '13. San Sebastian, Spain : ACM, 2013, p. 39–45 (Cité page 31).

[Kad02] Martin KADA. « Automatic generalisation of 3D building models ». In : *Proceedings of the Joint International Symposium on Geospatial Theory, Processing and Applications*. 2002 (Cité page 35).

[Kad07] Martin KADA. « Scale-dependent Simplification of 3D Building Models Based on Cell Decomposition and Primitive Instancing ». In : *Proceedings of the 8th International Conference on Spatial Information Theory*. COSIT'07. Melbourne, Australia : Springer-Verlag, 2007, p. 222–237 (Cité page 35).

[KG15] Michel KRÄMER et Ralf GUTBELL. « A Case Study on 3D Geospatial Applications in the Web Using State-of-the-art WebGL Frameworks ». In : *Proceedings of the 20th International Conference on 3D Web Technology*. Web3D '15. Heraklion, Crete, Greece : ACM, 2015, p. 189–197 (Cité pages 38, 40).

[Kol+05] Thomas H. KOLBE, Gerhard GRÖGER et Lutz PLÜMER. « CityGML : Interoperable Access to 3D City Models ». In : *Geo-information for Disaster Management*. Sous la dir. de Peter van OOSTEROM, Siyka ZLATANOVA et Elfriede M. FENDEL. Berlin, Heidelberg : Springer Berlin Heidelberg, 2005, p. 883–899 (Cité pages 32, 33).

[Kra+17] Julian KRATT, Ferdinand EISENKEIL, Marc SPICKER, Yunhai WANG, Daniel WEISKOPF et Oliver DEUSSEN. « Structure-aware Stylization of Mountainous Terrains ». In : *Vision, Modeling & Visualization*. Sous la dir. de Matthias HULLIN, Reinhard KLEIN, Thomas SCHULTZ et Angela YAO. The Eurographics Association, 2017 (Cité page 39).

[Lav+13] Guillaume LAVOUÉ, Laurent CHEVALIER et Florent DUPONT. « Streaming Compressed 3D Data on the Web using JavaScript and WebGL ». In : *International Conference on 3D Web Technology (Web3D)*. Sous la dir. d'ACM. San Sebastian, Spain, juin 2013, p. 19–27 (Cité page 30).

[Ler09] Raphaël LERBOUR. « Adaptive streaming and rendering of large terrains ». Theses. Université Rennes 1, déc. 2009 (Cité page 38).

[Lim+13] Max LIMPER, Stefan WAGNER, Christian STEIN, Yvonne JUNG et André STORK. « Fast Delivery of 3D Web Content : A Case Study ». In : *Proceedings of the 18th International Conference on 3D Web Technology*. Web3D '13. San Sebastian, Spain : ACM, 2013, p. 11–17 (Cité page 29).

[Lue03] David P. LUEBKE. *Level of detail for 3D graphics*. Morgan Kaufmann, 2003 (Cité page 25).

[Lyn60] Kevin LYNCH. *The image of the city*. T. 11. MIT press, 1960 (Cité page 36).

[Lév+02] Bruno LÉVY, Sylvain PETITJEAN, Nicolas RAY et Jérome MAILLOT. « Least Squares Conformal Maps for Automatic Texture Atlas Generation ». In : *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '02. San Antonio, Texas : ACM, 2002, p. 362–371 (Cité page 22).

[Mag+15] Adrien MAGLO, Guillaume LAVOUÉ, Florent DUPONT et Céline HUDELOT. « 3D Mesh Compression : Survey, Comparisons, and Emerging Trends ». In : *ACM Comput. Surv.* 47.3 (fév. 2015), 44 :1–44 :41 (Cité page 30).

[Mao+11] Bo MAO, Yifang BAN et Lars HARRIE. « A multiple representation data structure for dynamic visualisation of generalised 3D city models ». In : *ISPRS Journal of Photogrammetry and Remote Sensing* 66.2 (2011). Quality, Scale and Analysis Aspects of Urban City Models, p. 198 –208 (Cité pages 36, 38).

[Mao11] Bo MAO. « Visualisation and Generalisation of 3D City Models ». QC 20111116. Thèse de doct. KTH, Geoinformatik och Geodesi, 2011, p. xii, 104 (Cité page 34).

[Mas+10] Joan MASÓ, Keith POMAKIS et Núria JULIÀ. *Web Map Tile Service*. Standard 07-057r7. Open Geospatial Consortium, 2010 (Cité page 33).

[MB10] Bo MAO et Yifang BAN. « A Dynamic Typification Method of 3D City Models using Minimum Spanning Tree ». In : *6th international conference on Geographic Information Science (GIScience2010), Zurich, Switzerland*. 2010 (Cité page 36).

[MB11] Bo MAO et Yifang BAN. « Online Visualisation of a 3D City Model Using CityGML and X3DOM ». In : *Cartographica* 46.2 (2011), p. 109–114 (Cité page 38).

[Mea80] Donald MEAGHER. *Octree Encoding : a New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer*. Rapp. tech. Rensselaer Polytechnic Institute. Image Processing Laboratory, 1980 (Cité page 23).

[Nou+15] Romain NOUVEL, Robert KADEN, Jean-Marie BAHU, Jerome KAEMPF, Piergiorgio CIPRIANO, Moritz LAUSTER, Joachim BENNER, Esteban MUNOZ, Olivier TOURNAIRE et Egbert CASPER. « Genesis of the citygml energy ADE ». In : *Proceedings of International Conference CISBAT 2015 Future Buildings and Districts Sustainability from Nano to Urban Scale*. EPFL-CONF-213436. LESO-PB, EPFL. 2015, p. 931–936 (Cité page 33).

[OM14] Peter van OOSTEROM et Martijn MEIJERS. « Vario-scale data structures supporting smooth zoom and progressive transfer of 2D and 3D data ». In : *International Journal of Geographical Information Science* 28.3 (2014), p. 455–478 (Cité page 117).

[Pap+14] Nicolas PAPARODITIS, Jean-Pierre PAPELARD, Bertrand CANNELLE, Alexandre DEVAUX, Bahman SOHEILIAN, Nicolas DAVID et Erwan HOUZAY. « Stereopolis II : A multi-purpose and multi-sensor 3D mobile mapping system for street visualisation and 3D metrology ». In : *Revue Française de Photogrammétrie et de Télédétection* 200 (2014), p. 69–79 (Cité pages 8, 9).

[PD15] Federico PONCHIO et Matteo DELLEPIANE. « Fast Decompression for Web-based View-dependent 3D Rendering ». In : *Proceedings of the 20th International Conference on 3D Web Technology*. Web3D '15. Heraklion, Crete, Greece : ACM, 2015, p. 199–207 (Cité page 30).

[Pra+12] Federico PRANDI, Raffaele DE AMICIS, Giuseppe CONTI et Alberto DEBIASI. « Use of OGC Web Standard for a Spatio-temporal Enabled SDI for Civil Protection ». In : *Proceedings of the 17th International Conference on 3D Web Technology*. Web3D '12. Los Angeles, California : ACM, 2012, p. 105–111 (Cité page 39).

[Pra+14] Federico PRANDI, Marco SOAVE, Federico DEVIGILI, Michele ANDREOLLI et Raffaele DE AMICIS. « Services Oriented Smart City Platform Based On 3d City Model Visualization ». In : *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* II-4 (2014), p. 59–64 (Cité page 39).

[Pra+15] Federico PRANDI, Giulio PANIZZONI, Daniele MAGLIOCCHETTI, Federico DEVIGILI et Raffaele DE AMICIS. « WebGL Virtual Globe for Efficient Forest Production Planning in Mountainous Area ». In : *Proceedings of the 20th International Conference on 3D Web Technology*. Web3D '15. Heraklion, Crete, Greece : ACM, 2015, p. 143–151 (Cité page 40).

[Pri+12] Iñaki PRIETO, Jose Luís IZKARA et Francisco Javier Delgado del HOYO. « Efficient Visualization of the Geometric Information of CityGML : Application for the Documentation of Built Heritage ». In : *Proceedings of the 12th International Conference on Computational Science and Its Applications - Volume Part I*. ICCSA'12. Salvador de Bahia, Brazil : Springer-Verlag, 2012, p. 529–544 (Cité page 38).

[Red+17] Paula REDWEIK, Paula TEVES-COSTA, Inês VILAS-BOAS et Teresa SANTOS. « 3D City Models as a Visual Support Tool for the Analysis of Buildings Seismic Vulnerability : The Case of Lisbon ». In : *International Journal of Disaster Risk Science* 8.3 (sept. 2017), p. 308–325 (Cité page 39).

[Rua04] Anne RUAS. « Le changement de niveau de détail dans la représentation de l'information géographique ». HDR. University of Marne la Vallée, 2004 (Cité page 34).

[Rua99] Anne RUAS. « Modèle de généralisation de données géographiques à base de contraintes et d'autonomie ». Thèse de doctorat dirigée par Désarménien, Jacques Sciences de l'information géographique Université de Marne-la-Vallée 1999. Thèse de doct. 1999, 1 vol. (323 p.) (Cité page 4).

[Sch+16] Arne SCHILLING, Jannes BOLLING et Claus NAGEL. « Using glTF for Streaming CityGML 3D City Models ». In : *Proceedings of the 21st International Conference on Web3D Technology*. Web3D '16. Anaheim, California : ACM, 2016, p. 109–116 (Cité page 39).

[Sch16] Markus SCHÜTZ. « Potree : Rendering Large Point Clouds in Web Browsers ». Mém.de mast. Favoritenstrasse 9-11/186, A-1040 Vienna, Austria : Institute of Computer Graphics et Algorithms, Vienna University of Technology, sept. 2016 (Cité page 30).

[Son+10] Kristian SONS, Felix KLEIN, Dmitri RUBINSTEIN, Sergiy BYELOZYOROV et Philipp SLUSALLEK. « XML3D : interactive 3D graphics for the web ». In : *Web3D '10 : Proceedings of the 15th International Conference on Web 3D Technology*. Los Angeles, California : ACM, 2010, p. 175–184 (Cité page 31).

[Sut+14] Jan SUTTER, Kristian SONS et Philipp SLUSALLEK. « Blast : A Binary Large Structured Transmission Format for the Web ». In : *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies*. Web3D '14. Vancouver, British Columbia, Canada : ACM, 2014, p. 45–52 (Cité page 31).

[Thi02] Frank THIEMANN. « Generalization of 3D Building Data ». In : *Part 4, &quot ;GeoSpatial Theory, Processing and Applications*. 2002, p. 286–290 (Cité page 35).

[Tra+08] Matthias TRAPP, Tassilo GLANDER, Henrik BUCHHOLZ et Jürgen DÖLLNER. « 3D Generalization Lenses for Interactive Focus + Context Visualization of Virtual City Models ». In : *2008 12th International Conference Information Visualisation*. Juil. 2008, p. 356–361 (Cité page 39).

[TS06] Frank THIEMANN et Monika SESTER. « 3D-symbolization using adaptive templates ». In : *Proceedings of the GICON* (2006) (Cité page 36).

[Ver+15] Yannick VERDIE, Florent LAFARGE et Pierre ALLIEZ. « LOD Generation for Urban Scenes ». In : *ACM Transactions on Graphics* 34.3 (2015), p. 15 (Cité page 35).

[Vre14] Panagiotis (Peter) A. VRETANOS. *Web Feature Service*. Standard 09-025. Open Geospatial Consortium, 2014 (Cité pages 33, 64).

[Wal+09] Georg WALENCIAK, Beate STOLLBERG, Steffen NEUBAUER et Alexander ZIPF. « Extending Spatial Data Infrastructures 3D by Geoprocessing Functionality - 3D Simulations in Disaster Management and environmental Research ». In : *Advanced Geographic Information Systems Web Services, 2009. GEOWS '09. International Conference on*. Fév. 2009, p. 40–44 (Cité page 40).

[Wal07] Ingo WALD. « On Fast Construction of SAH-based Bounding Volume Hierarchies ». In : *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*. RT '07. IEEE Computer Society, 2007, p. 33–40 (Cité page 25).

[Chr16] Martin CHRISTEN. « Openwebglobe 2 : Visualization of Complex 3D-GEODATA in the (mobile) Webbrowser ». In : *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* (juin 2016), p. 401–406 (Cité page 38).

[Cur+15] Rémi CURA, Julien PERRET et Nicolas PAPARODITIS. « Point Cloud Server (pcs) : Point Clouds In-Base Management and Processing ». In : *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* (août 2015), p. 531–539 (Cité pages 38, 40).