Traçage d'une navigation aérienne pour un moteur de rendu cartographique

Auteur : Deswysen, Benjamin
Promoteur(s) : Donnay, Jean-Paul
Faculté : Faculté des Sciences
Diplôme : Master en sciences géographiques, orientation géomatique et géométrologie, à finalité spécialisée
Année académique : 2016-2017
URI/URL : http://hdl.handle.net/2268.2/2497

Faculté des Sciences

Traçage d'une navigation aérienne pour un moteur de rendu cartographique

Mémoire présenté par
Benjamin Deswysen

pour l'obtention du titre de
Master en sciences géographiques
orientation géomatique et géométrologie

Sous la direction du
Pr. *Jean-Paul* Donnay

Lecteurs : Hallot Pierre, Schmitz Serge

Année académique 2016-2017

Je souhaiterais tout d'abord remercier mon promoteur, le professeur Jean-Paul DONNAY*, pour sa patience mais également son encadrement lors de la réalisation de ce mémoire.*

Je remercie également d'avance mes lecteurs, Pierre HALLOT *et Serge* SCHMITZ*.*

Ce travail n'aurait jamais pu voir le jour sans l'initiative de l'équipe d'ESNAH. Merci pour votre collaboration, votre sympathie naturelle et cette opportunité.

Merci à ma famille et mes proches pour leur appui. Plus particulièrement Stéphanie, ma soeur, pour ses relectures et encouragements durant la dernière ligne droite.

Enfin, merci à toi, Natacha, pour ta patience et ton soutien à toute épreuve.

Table des matières

Table des figures

Liste des tableaux

1 Introduction

1.1 Domaine de recherche

Cinq morts. C'est le bilan du dernier accident d'avion mortel en date en Europe. Ce crash d'un avion de tourisme a eu lieu le 17 avril 2017 dans la région de Lisbonne, au Portugal (Le Monde, 2017). Durant l'année 2016, ce n'est pas moins de cent six incidents graves qui ont eu lieu dans le domaine de l'aviation, au sein de l'Union européenne [1] (Ky, 2017). En Europe, comme dans le reste du monde, la sécurité aérienne est un enjeu capital : des dizaines de milliers de vols sont réalisés chaque jour à l'échelle du globe et leur nombre ne cesse de croître.

Heureusement, la navigation à vue, en vigueur aux prémisses de l'aviation et toujours largement utilisée par l'aviation légère, n'est désormais plus le seul mode de navigation des pilotes. Au fil du temps, des instruments ont été élaborés, afin d'aider les pilotes lors de leur navigation et de maîtriser leurs déplacements. Ils ont progressivement pu déterminer leur position par le biais de la radionavigation, puis, plus récemment, grâce au positionnement par satellites. Actuellement, certains de ces assistants sont même embarqués sous forme d'applications mobiles, depuis que les tablettes et smartphones bénéficient de récepteurs GNSS [2].

Parallèlement à ces avancées, les prévisions météorologiques deviennent de plus en plus fiables également. Un réseau de stations a été mis en place par l'*organisation de l'aviation civile internationale* (ICAO), afin d'aider les pilotes à s'adapter en fonction de la météo. Ainsi, ces stations ont commencé à diffuser des rapports d'observation (METAR) et des prévisions (TAF) se rapportant à la météo. Toutefois, ces données sont véhiculées au format texte selon une codification qui leur est propre. Pour en bénéficier, les pilotes doivent les interpréter et s'aider d'une carte lorsqu'ils veulent les situer dans l'espace.

La centralisation des données spécifiques à la navigation, d'une part, et les données météorologiques, d'autre part, est désormais possible et ce, également pour les intervenants de l'aviation légère. En effet, suite à leur finalité distincte de celle de l'aviation commerciale (plutôt orientée vers le transport aérien), les acteurs de l'aviation légère disposent souvent de moins d'appareils embarqués. Néanmoins, les applications d'aide à la navigation se multiplient depuis l'apparition des tablettes électroniques et smartphones. Ces appareils mobiles constituent des assistants de vol intéressants : ils disposent souvent d'un récepteur GNSS, d'un dispositif d'affichage interactif, de la possibilité de se connecter à internet, etc. Ceci fait d'eux des prétendants sérieux au poste de compléments des appareils déjà embarqués.

Ce domaine de recherche, l'élaboration de systèmes de navigation aéronautiques, se situe à l'intersection de plusieurs domaines tels que le développement d'applications mobiles, les prévisions météorologiques, la préparation de dossiers de vol ou encore la géomatique.

Ce mémoire résulte d'une collaboration entre l'unité de Géomatique de l'université de Liège, par le biais du professeur Jean-Paul Donnay, et la société ESNAH [3], société belge travaillant sur des systèmes d'aide à la navigation dans le domaine de l'aviation légère. La volonté d'élaborer une solution permettant le traitement et la visualisation de données géographiques multi-sources autour d'une route de navigation aérienne a progressivement vu le jour. Cette solution, mêlant outils géomatiques et informatiques, s'est concrétisée sous la forme d'un prototype élaboré dans un contexte de recherche-développement. Ce mémoire s'est donc réalisé selon une question de recherche visant à proposer une solution exploitable, par la suite, au sein des outils développés par la société. Toutefois, le contexte de développement que connait la startup ESNAH nécessite des solutions rapidement déployées. Dès lors, la réalisation de ce mémoire s'est achevée de manière indépendante, en parallèle à d'autres développements internes à la société issus d'une réflexion commune. Néanmoins, ce prototype constitue un exemple d'implémentation fonctionnelle et originale susceptible d'apporter certains enseignements.

1. Statistiques d'incidents au sein de pays de l'Union européenne réalisées par l'agence européenne de la sécurité aérienne. Ces incidents concernent des vols commerciaux, effectués par des avions dont le poids est supérieur 5700 kg.
2. Global Navigation Stallite System.
3. Etude pour la Sécurité de la Navigation Aérienne Hanse.

1.2 Plan du mémoire

Ce travail est organisé en 7 sections. Néanmoins, celles-ci peuvent être regroupées selon 5 thématiques principales. La première d'entre-elles est consacrée à la **démarche** entreprise dans le cadre de cette collaboration avec la société ESNAH (Section 2). Elle décrit l'environnement de travail en détaillant les spécialités d'ESNAH ainsi que leurs attentes à l'issue de cette collaboration. Parmi ces attentes ont été sélectionnées les fonctionnalités remplies par ce prototype : la création de fonds de cartes spécifiques au contexte aéronautique, le traitement de données météorologiques typiques de ce contexte et la création d'une interface permettant à l'utilisateur d'y préparer son vol sur base des deux éléments précédents. Le contexte associé à l'aéronautique et celui des instruments d'aide à la navigation sont également quelque peu développés afin d'assurer une bonne compréhension de la problématique. Enfin, les pistes reléguées au rang de perspectives et d'améliorations potentielles sont également évoquées.

La seconde thématique, plus technique, concerne l'**architecture informatique** déployée pour desservir le prototype (Section 3). Celui-ci, fonctionnant selon le paradigme client-serveur, fournit des services aux pilotes, afin qu'ils puissent préparer leurs vols depuis une interface client. Ces services sont fournis par le serveur, sous forme de services web, et leur déploiement a requis trois opérations :

1. Obtention des données.
2. Traitement des données.
3. Création de services web originaux.

Ainsi, ces opérations ont été appliquées aux données météorologiques mais également à celles destinées aux fonds de cartes. Toutefois, les traitements spécifiques des données géographiques bénéficient chacun de leur propre section au sein de ce travail (voir ci-après). Seuls les mécanismes nécessaires à ces opérations figure dans cette partie.

La troisième thématique reprend les **traitements spécifiques réalisés sur les données géographiques**. Ces traitements concernent les données météorologiques d'une part (Section 4), et les données planimétriques d'autre part (Section 5). Le premier type de données, réalisé à partir de données METAR, est traité de manière à être consultable par l'utilisateur au sein d'une application de cartographie. Il est également exploité afin d'indiquer à l'utilisateur les différentes règles de vol qui sont en vigueur sur le territoire. Le second type, quant à lui, a pour vocation de générer les fonds de cartes originaux servis dans l'interface accessible à l'utilisateur. Ces fonds de cartes sont réalisés à partir de données OpenStreetMap et SRTM [4].

La quatrième thématique exploitée est celle d'une **interface client** (Section 6) **proposée à un utilisateur pilote qui souhaite préparer son vol**. Des outils sont mis à sa disposition, afin qu'il puisse définir un couloir de vol en conséquence des données fournies par la troisième thématique (fonds de cartes et données météorologiques de type METAR). Une sélection des données METAR est ensuite possible sur base de ce couloir. Ainsi, le pilote pourra récupérer les données météorologiques en conséquence du vol qu'il planifie.

Enfin, la cinquième thématique regroupe les **travaux complémentaires** envisagés dans la continuité de ce travail (Section 7) et les **conclusions** tirées à l'issue de ce mémoire (Section 8).

4. Shuttle Radar Topography Mission.

2 Démarche

Suite au développement de son application "SkyLiberty" d'aide à la navigation, l'équipe d'ESNAH sprl a voulu y implémenter des solutions géomatiques, afin de réaliser *in fine* une fonctionnalité de type "moving map". Dès lors, c'est dans un premier temps vers le professeur Donnay de l'unité de géomatique à l'université de Liège, que Nicolas Hanse, CEO de la société, s'est tourné.

Cette problématique a donné naissance a un projet de mémoire que j'ai accepté d'amorcer. Ce projet, consistant à la modélisation d'une solution spécifique à l'environnement de travail d'ESNAH, a pour objectif d'aboutir à un prototype. Celui-ci étant réalisé à partir d'outils géomatiques "open source" (si possible) et se concrétisant jusqu'au niveau physique, afin de pouvoir par la suite, être combiné à un sac de vol électronique (EFB) sur appareil mobile (Section 2.3) ou être intégré à celui-ci.

2.1 Environnement et contexte de travail

2.1.1 La société ESNAH sprl

La société ESNAH est une compagnie belge travaillant sur des systèmes d'aide à la navigation pour les étudiants pilotes, les pilotes privés et commerciaux, mais également pour les propriétaires d'avions et les opérateurs (ESNAH sprl, 2017). Bien qu'une première version de son application de sécurité aérienne "SkyLiberty" soit disponible depuis 2015, celle-ci reste en développement et propose constamment de nouvelles fonctionnalités, sous forme de modules complémentaires. Jusqu'à présent, l'application propose essentiellement des outils de préparation "administrative" de vol. La prochaine étape de son évolution est donc tournée vers des outils de cartographie.

Les souhaits de la société en la matière sont ambitieux : application de type "moving map" offrant un rendu 3 dimensions, intégration de données multi-sources dont certaines doivent être stockées sur l'appareil afin de ne pas dépendre d'une connexion lors de la navigation, chargement en temps réel des données météorologiques, etc. Ces ambitions, d'intégrer des outils géomatiques au sein de thématiques innovantes, ont dès lors suscité une profonde réflexion lors de la planification de ce mémoire.

2.1.2 Thématique initiale et développement d'un prototype

Développement d'un prototype

Le prototype qui a été réalisé se place donc dans un contexte de **recherche développement** d'une solution géomatique et informatique selon une thématique d'avionique, qui se définit comme l' "Application des techniques de l'électronique au domaine de l'aviation" (Larousse, s.d.).

Ce mémoire n'étant pas réalisé selon une approche hypothético-déductive mais bien selon un cas de recherche-développement aboutissant à la réalisation d'un prototype, la structure de ce document sera quelque peu différente des écrits traditionnels de recherche. En effet, la conception du modèle de prototype a été réalisée en fonction du contexte (aéronautique) de la problématique et sa concrétisation en se basant sur les solutions informatiques "open source" disponibles. Par conséquent, une section de documentation abordant le contexte de l'affichage de données de vol est réalisée dans un premier temps et des compléments de documentation seront ajoutés au gré des besoins.

Thématique initiale

La requête initiale attendait une *optimisation du traçage de navigation aérienne pour un moteur de rendu cartographique* sur un appareil mobile (de type tablette ou smartphone). De cette thématique ont découlé trois problématiques principales :

1. La **préparation de vols** par l'élaboration de routes de navigations aériennes ;
2. L'**optimisation du chargement de données** météorologiques selon une sélection de données utiles à la navigation **en fonction de la trajectoire** prévue ;

3. L'**aide à la navigation sur un support mobile** offrant un affichage multi-sources dans une plate-forme de "mapping".

Les ambitions en rapport avec le sujet étant diverses, il a fallut organiser les réponses aux attentes et les pistes envisagées (Section 2.5), parfois en écartant certains développements (Section 2.6).

Dans le cadre de ce mémoire, l'intérêt est porté sur le rôle d'**assistance à la préparation de vol** d'un "sac de vol électronique" (Section 2.3). Celui-ci devrait permettre à l'utilisateur de planifier son vol, en optant pour une trajectoire qui est fonction des données de terrain mais également météorologiques. Il n'est pas impensable d'incorporer ces éléments dans une application de type "Moving Map" comme envisagé initialement par la société ESNAH, permettant également au pilote de bénéficier d'une assistance de navigation lors du vol.

2.2 Etat de l'art sur les instruments d'aide à la navigation

Étant donné les contingences d'un vol auxquelles un pilote est confronté, il va sans dire que toute assistance peut être profitable. En effet, malgré une préparation minutieuse du vol, celui-ci peut être perturbé par certains aléas. A titre d'exemple, on peut mentionner un problème de santé d'un occupant de l'avion, la rencontre d'une zone de turbulence ou encore un dysfonctionnement des appareils de navigation. En somme, en plus de préparer rigoureusement son vol, un pilote doit se préparer au plus complexe des scénarios, afin de pouvoir réagir le moment venu. Des compléments technologiques sont dès lors utilisés. Ils auront pour objectif d'assister le pilote (qui se base sur ce qu'il voit dans son champ de vision) dans sa tâche, afin de le rendre moins dépendant des conditions de vol.

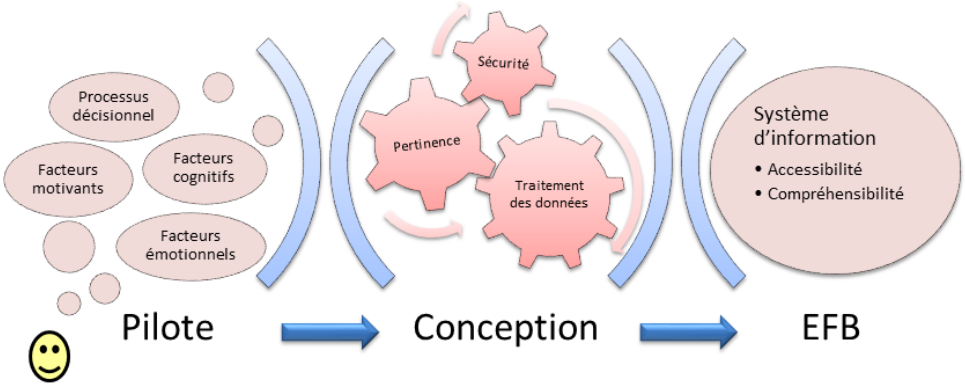

FIGURE 1 – Conception d'instruments d'aide à la navigation en fonction des besoins des pilotes (Stein & Sandl, 2012)

L'utilisateur est dès lors inondé de toutes sortes de données. Ce qui, paradoxalement, est contraire à son information(Stein & Sandl, 2012). Le défi est donc d'identifier, puis traiter les données pertinentes à présenter à l'utilisateur durant le vol sans accaparer sa concentration. En effet, pour une question évidente de sécurité, les données à afficher durant le vol doivent être facilement accessibles et compréhensibles en un simple coup d'œil : l'utilisateur a peu de temps pour la recherche d'informations et est susceptible d'être soumis à un stress lors de celle-ci. Par conséquent, il est vital de concevoir des systèmes d'information (SI) synoptiques, permettant de voir un ensemble de renseignements, qui prennent en compte :

- le processus décisionnel,
- les facteurs cognitifs,
- les facteurs motivants,
- les facteurs émotionnels.

C'est dans cette optique que sont développés les équipements d'avions tels que les sacs de vol électroniques, en anglais "electronic flight bags" (EFB). Ceux-ci ont pour but d'assister le pilote sans toutefois nuire à la navigation (voir section 2.3).

Afin de comprendre le contexte dans lequel s'inscrit ce type d'équipement, penchons-nous sur un bref historique. C'est en 1991, lorsque la société FedEx a mis en place un système d'ordinateurs embarqués, afin d'effectuer des calculs de décollage de dernière minute, qu'est né le principe d'EFB (Thomas, 2006; Turiak et al., 2014). Par la suite, est arrivé le concept de "Flight Management Desktop" (FMD) en 1996 : deux ordinateurs sont substitués à la documentation traditionnelle -au format papier- du poste de pilotage. Ceux-ci sont également capables de calculer des performances de l'appareil. La société JetBlue a même été jusqu'à convertir tous ses documents opérationnels en format électronique, avant de les distribuer via un réseau aux pilotes concernés. Ceux-ci peuvent dès lors les consulter sur leur ordinateur portable. Toutefois, c'est en 1999 que le premier vrai EFB est breveté par Angela Masson, en tant que "Sac à kit électronique" (EKB). Celui-ci se veut être un **substitut au sac complet d'un kit pilote** et ce dans "*un dispositif portable configuré pour afficher, organiser, gérer, manipuler et générer des données de vol* (...)" (Masson, 2016).

Depuis lors, nombre d'améliorations ont été apportées dans les cockpits pour s'ajouter aux instruments analogiques déjà présents : ordinateurs de bord avec écran tactile (Groen et al., 2008), couplage avec la technologie GNSS (Global Navigation Satellite System) dans des applications de type "Moving Map" (Federal Aviation Administration - U.S.Department of Transportation, Flight Standards Service, 2009) ou encore des applications pour appareils mobiles (Sanjay, s.d.; Avsoft, s.d.).

Ces solutions sont généralement fonction de l'avion : un avion de ligne qui a pour objectif (commercial) de transporter des dizaines de passagers sur de longues distances et à un niveau de vol élevé ne présentera pas le même équipement qu'un avion léger destiné au tourisme aérien (voir Figure 2 à gauche et à droite respectivement). Toutes ces améliorations n'ont pas vocation à ajouter des informations au pilote dont l'attention est déjà fortement sollicitée, mais bien de remplacer certains instruments, en incorporant leur(s) fonction(s) dans un support électronique, que ce soit directement dans le cockpit ou sur des appareils non inclus à l'avion. Ainsi, son champ de vision se voit épuré, libérant ainsi son attention.

FIGURE 2 – Avion de type lourd modèle Boeing 747-400 de la compagnie China Airlines pouvant accueillir 660 passagers (Bukowski, 2007) et avion de type léger modèle Diamond DA40 accueillant maximum 4 personnes, pilotes inclus (Simon, 2015).

2.3 Les sacs de vol électroniques (EFB)

Comme mentionné précédemment, les EFB sont de véritables outils multi-fonctions pour les cockpits modernes. A l'heure actuelle, il existe déjà beaucoup d'EFB et leur nombre continue de croitre, offrant toujours de nouvelles fonctionnalités aux utilisateurs (Thomas, 2006). Il convient dès lors d'en distinguer plusieurs types, soumis à différentes règlementations selon leur impact sur l'attention du pilote et la navigation (Turiak et al., 2014).

Aspect "hardware"

D'un point de vue purement matériel, les distinctions sont les suivantes :

- ◇ **Les EFB portables** constituent un équipement non certifié au sein de l'avion. Ceux-ci sont utilisables au sein du cockpit mais également en dehors. Ils peuvent s'avérer particulièrement utiles lors de la planification du vol ou lors du dépôt du plan de vol au format électronique.

- ◇ **Les EFB intégrés** sont un équipement certifié, selon un certificat de navigabilité et inclus à l'équipement de l'avion en étant intégré au cockpit.

Aspect "software"

Lorsque l'on considère un point de vue logiciel et non plus matériel :

- ◇ **EFB de type A**, sont des applications qui, lors d'une erreur ou d'un usage incorrect, n'auront pas d'influence sur la sécurité. Ils peuvent donc équiper les deux types de hardware et ne nécessitent pas de certification. Toutefois, l'appareil se doit d'être conforme avec des exigences spécifiques aux EFB, basées sur des analyses de comportement homme-machine : lisibilité de l'information selon les conditions d'éclairement, méthode adéquate d'avertissement de l'équipage, etc. Ce type d'EFB peut, par exemple, servir à l'affichage des certificats d'enregistrement ou encore en tant qu'application interactive de calcul du repos de l'équipage.

- ◇ **EFB de type B**, sont des applications qui peuvent causer un dysfonctionnement mineur lors d'une erreur ou d'un usage incorrect. Bien qu'ils ne remplacent aucun système exigé par les règlements de l'espace aérien et les règlements opérationnels, ils requièrent un examen opérationnel (Kalašová & Krchová, 2013). Ils peuvent correspondre aux deux types de hardware pré-cités. Leur fonction peut, par exemple, consister à afficher des manuels ou des cartes électroniques interactives (défilement, zoom, centrage, etc.), proposer une application de positionnement en temps réel, fournir une application de réception/traitement/distribution de données via internet (ou autre moyen de communication aéronautique) ou encore calculer des paramètres de vol tels que le calcul de la piste de décollage, le poids et l'équilibre de l'avion.

Législations quant à l'usage d'appareils multimédia à bord d'un avion

Le milieu de l'aviation étant fortement règlementé, il faut considérer ce nouveaux type d'appareils d'assistance au pilotage. En effet, si le pilote ne doit pas être tenté de suivre aveuglément son outil informatique, il doit pouvoir profiter rapidement de son assistance en cas de besoin, sans encombrement. Dès lors, des directives ont été établies concernant l'utilisation d'EFB en Europe dès 2004 (Turiak et al., 2014). Toutefois, ces directives remplissent plutôt une fonction informative que contraignante.

EFB sur appareils mobiles

Suite à leur popularité, les appareils mobiles de type tablettes sont de sérieux candidats au poste d'EFB (Turiak et al., 2014). Ceux-ci permettent d'effectuer certaines opérations en dehors du cockpit : lors de la préparation du vol ou encore après celui-ci, lors de la clôture du plan de vol par voie électronique. Ensuite, il peuvent assister le pilote lors de son vol afin de compléter les instruments déjà embarqués (voir Figure 3).

Dès lors, ce mémoire a pour objectif de réaliser une interface de préparation de vol qui pourra être incluse ou complémentaire à un EFB portable. Ce dernier fonctionnant sur un appareil mobile, tel qu'une tablette, et étant de type B. Certaines contraintes, liées à cet environnement, ont donc été prises en ligne de compte lors de la réalisation.

2.4 Contexte aéronautique général

Afin d'appréhender l'utilité du prototype réalisé lors de ce mémoire, il me faut contextualiser quelque peu le domaine aéronautique. Ainsi, quelques principes de bases sont résumés ci-après.

FIGURE 3 – EFB de type tablette utilisé durant un vol (Turiak et al., 2014)

2.4.1 Règles de vol

Les différents modes de pilotage dépendent des conditions météorologiques qui sont exprimées en fonction de la visibilité, de la distance par rapport aux nuages et du plafond (Ministère Francais Des Transports De L'Equipement Du Tourisme Et De La Mer, 2006b). Selon ces valeurs, les conditions permettent un vol à vue ou, a contrario, un vol aux instruments. Ces deux modes de vols sont associés à des règlementations distinctes.

- **Vol à vue (VFR)** : selon la section instruction de la division française de l'IVAO (International Virtual Aviation Organisation)(IVAO, 2014b) :

> *"Un pilote effectue un **vol selon les règles de vol à vue** ou VFR (Visual Flight Rules), lorsqu'il maintient son avion dans une configuration propre au vol (attitude, vitesse) sur sa trajectoire pour l'emmener vers sa destination en s'orientant par rapport à des repères extérieurs (reliefs, routes, villes...) et à l'aide d'une carte."*

  Ce mode de vol, datant des origines de l'aviation, permet une certaine liberté au pilote tout en ne nécessitant que peu d'instruments de bord. Toutefois, il n'est envisageable que lorsque les conditions météorologiques le permettent : on parle alors de "conditions météorologiques de vol VMC" (Visual Meteorological Conditions). Ces conditions doivent être égales ou supérieures aux minimums en vigueur pour permettre un vol VFR (voir Table A) (ICAO, 2005b). Notons que certaines classes d'espaces aériens sont interdites au mode VFR ou nécessitent un accord au préalable, imposant l'usage des instruments de bord.

- **Vol aux instruments (IFR)** : si l'on se réfère une nouvelle fois à la section instruction de la division française de l'IVAO (IVAO, 2014a) :

*"Un pilote effectue un **vol selon les règles de vol aux instruments** ou IFR (Instrument Flight Rules), lorsqu'il respecte un certain nombre de critères :*
- *L'ensemble des trajectoires IFR est basé sur des moyens radioélectriques et/ou de positionnement ;*
- *Il maintient une configuration propre aux vols IFR (sélection d'altitude, contraintes de vitesses ...) ;*
- *Il suit une trajectoire imposée par les organismes de circulation aérienne ;*
- *Il respecte la règlementation et les procédures IFR publiées."*

Ainsi, lorsqu'il n'est pas possible de maintenir les conditions VMC du vol VFR, il est demandé au pilote de poursuivre son vol à l'aide de ses instruments et du contrôle aérien, sans information visuelle extérieure (ICAO, 2005c). Dès lors, un avion certifié IFR comprend des moyens de radionavigation, un système inertiel et un récepteur GNSS. Ceci implique des instruments tels qu'un horizon artificiel, 2 VOR (système de positionnement radioélectrique), etc. Le GPS (ou tout autre récepteur GNSS), quant à lui, est devenu presque incontournable actuellement, car il permet au pilote de connaître sa position dans l'espace et de suivre des itinéraires précis. Suite aux contraintes supplémentaires qu'il implique, ce mode de vol est soumis à d'avantage de règles et est plus encadré. Malgré cela, un vol peut être prévu directement en IFR : on parlera alors de "conditions météorologiques de vol IMC" (Instrumental Meteorological Conditions)". Notons qu'un vol IFR peut s'effectuer en VMC si les conditions le permettent mais aura comme avantage de tout de même pouvoir être effectué malgré de mauvaises conditions météo.

2.4.2 METAR, TAF et SPECI

Les informations météorologiques propres à la navigation aérienne sont transmises sous différents formats.

◇ **Les METAR**(METeorological Aerodrome Report) sont des **rapports d'observations** météorologiques de surface destinés à l'aviation. Chaque état membre de l'ICAO [5] doit établir un réseau de stations météorologiques aéronautiques au sein des aérodromes de son territoire (IVAO, 2005). Ces stations devront périodiquement émettre leurs observations, en langage codé, au sein d'un message. Celui-ci devra ensuite être déchiffré par les pilotes (US Departement of Transportation Federal Aviation Administration, 1999). Les conditions météorologiques qui y sont décrites sont considérées comme valides dans un rayon de 3 km autour du site d'observation. Cependant, lors de ce travail, cette zone a été étendue afin qu'**à chaque point de la carte soient associées les données METAR de la station la plus proche** (Section 4). Ce principe est celui des polygones de Voronoï et il permet, dans le cas présent, de couvrir l'ensemble du territoire avec des données météo.

Un message METAR contient typiquement (Nav Canada, 2009) :
- code de modification,
- identificateur METAR,
- identificateur de la station (code ICAO),
- date de l'observation,
- information sur le vent,
- visibilité horizontale dominante,
- temps présent,
- groupes de nuages,
- température / point de rosée,
- calage altimétrique,
- remarques.

Voici un exemple de METAR (fictif) pour l'aéroport de Liège dont l'interprétation figure dans la Table

5. "International Civil Aviation Organization" ou, en français, "Organisation de l'aviation civile internationale" (OACI)

1 :

```
METAR EBLG 151430Z 18010KT 5500 –SHRA 17/12 Q1008 NOSIG=
```

Code	Interprétation
METAR	Type de message : METAR
EBLG	Code ICAO de l'aérodrome (EBLG = aéroport de Liège)
151430Z	Date de l'observation : 15 du mois courant à 14h30 UTC (Z pour "Zoulou" signifiant UTC)
18010KT	Vent du 180 ° de vitesse 10 noeuds (avec 1 noeud = 1.852 km/h)
5500	Visibilité horizontale de 5.5 km (9999 étant la valeur maximale)
-SHRA	Averse de pluie de faible intensité (SH= shower, RA= rain et -= de faible intensité)
17/12	Température +17 ° C et point de rosée à +12 ° C
Q1008	Pression de 1008 hPa rapportée au niveau de la mer
NOSIG	Pas d'évolution prévue pour les deux prochaines heures
=	Fin du message

TABLE 1 – Interprétation d'un METAR pour l'aéroport de Liège

◇ **Les TAF** (Types of Aeronautical Forecasts) sont des **prévisions** d'aérodromes qui contiennent les valeurs les plus probables des paramètres dans un rayon de 5 miles marins, ce qui équivaut à 9.26 km (ICAO, 2005a; Nav Canada, 2009). Son contenu, également sous forme de code, est similaire à celui d'un METAR et reprend les éléments suivants :
- identificateur du bulletin,
- code de modification/correction,
- identificateur de l'aérodrome (code ICAO),
- date de l'observation,
- période de couverture,
- information sur le vent,
- visibilité horizontale dominante,
- temps présent,
- groupes de nuages,
- groupes de changement temporaire ou permanent,
- remarques.

◇ **Les SPECI** (SPECIal observations) sont des **observations spéciales** qui ont pour but d'avertir et de *spécifier* tout changement survenu en matière de vent, visibilité, plage visuelle de piste, temps actuel, nuages et/ou température de l'air (Nav Canada, 2009). Par conséquent, ces correctifs supplantent les observations antérieures de type METAR.

2.4.3 Les NOTAM

Les NOTAM sont des **avis** véhiculés par télécommunication et contenant des informations à propos des aménagements, des services, des procédures aéronautiques ou des dangers pour la navigation aérienne (Nav Canada, 2009).

2.4.4 Plan de vol

Voici un résumé des étapes à suivre lors d'un vol en aviation civile (Ministère Francais Des Transports De L'Equipement Du Tourisme Et De La Mer, 2006c; ICAO, 2005d) :

1. **Réalisation** du plan de vol : Celui-ci doit contenir entre-autre les identifiants de l'aéronef, le nombre de personnes à bord, les règles et types de vol, l'équipement, les aérodromes de départ et de destination, un ou des aérodrome(s) de dégagement, la route à suivre, l'autonomie... ;
2. **Dépôt** du plan de vol projeté aux organismes des services de la circulation aérienne au moins une heure avant l'heure de départ ;

3. **Établissement** du plan de vol ;
4. **Modifications éventuelles** du plan de vol à signaler ;
5. **Clôture** du plan de vol en remettant un compte rendu d'arrivée par radiotéléphonie ou par liaison de données. Ce compte rendu comporte l'identification de l'aéronef, les aérodromes de départ et d'arrivée ainsi que l'heure d'arrivée. La non remise d'un tel document peut entraîner de fortes perturbations dans les services de la circulation aérienne, ainsi que des opérations de recherches superflues qu'il est préférable d'éviter.

Dès lors, on remarque qu'un assistant de préparation de vol, tel qu'un EFB, trouve sa place lors de la planification de celui-ci. En effet, si l'appareil permet de centraliser plusieurs données et/ou fonctions, le gain de temps peut être considérable.

2.5 Attendus couverts par ce mémoire

Après structuration des attentes auxquelles le prototype devait répondre, une réorganisation ainsi qu'une réévaluation des ambitions initiales se sont imposées. En effet, le domaine du développement pour appareils mobiles, ainsi que de certaines technologies qui lui sont associées (telles que le rendu 3D par le biais de plateformes spécifiques) se sont avérés être des obstacles de nature matérielle, mais dont l'intérêt scientifique dépassait le cadre et les exigences d'un mémoire.

Par conséquent, la solution proposée s'organise sous forme d'une **interface de préparation de vol** accessible depuis un navigateur et qui n'est donc pas directement intégrée au sein d'une application mobile. Cette interface fonctionne selon une architecture informatique de type client-serveur au sein de laquelle un pilote pourra planifier son vol (Figure 4). Dans cette interface sont présents :

- ◇ **Différents fonds de cartes** selon le type de vol (VFR/VFR de nuit/IFR) et basés sur les données OpenStreetMap auxquelles sera ajoutée l'orographie sur base de données SRTM. Une stratégie de tuilage sous-jacente est mise en place au sein d'une application "moving map", afin d'obtenir des *fonds de cartes en fonction de la position observée à l'écran et selon le niveau de zoom* (Section 5). Les tuiles accessibles proviennent de différentes sources dont une "locale" pour laquelle les tuiles sont générées sur une machine dédiée, endossant le rôle de serveur, et d'autres accessibles via une API (Application Programming Interface) de Mapbox.

- ◇ **Des données météorologiques** de type METAR *affichables à la demande*. Celles-ci sont disponibles de manière ponctuelle (pour chacune des stations) ou sous forme de polygones représentant la zone d'influence entourant chacune des stations sur base des polygones de Voronoï (voir Section 4 pour les détails des traitements des données météorologiques). Leur chargement s'effectue dynamiquement selon la fenêtre d'affichage.

- ◇ **Des outils interactifs** permettant la *définition de la trajectoire*. Ils permettent d'interroger les données météo selon la route planifiée en incorporant un "buffer" de sécurité autours de celle-ci. On obtient ainsi un "couloir de vol" qui permet de recouper les données METAR sur le trajet à partir d'une intersection avec les entités géographiques concernées. Ce recoupement, quant à lui, est exécuté côté client, à condition que les données météo aient été téléchargées depuis le serveur au préalable. Enfin, la possibilité d'importer et exporter des géométries a été implémentée, afin de pouvoir conserver les données récoltées lors des différentes sessions de l'utilisateur. Le développement de ces outils est repris dans la Section 6 qui leur est consacrée.

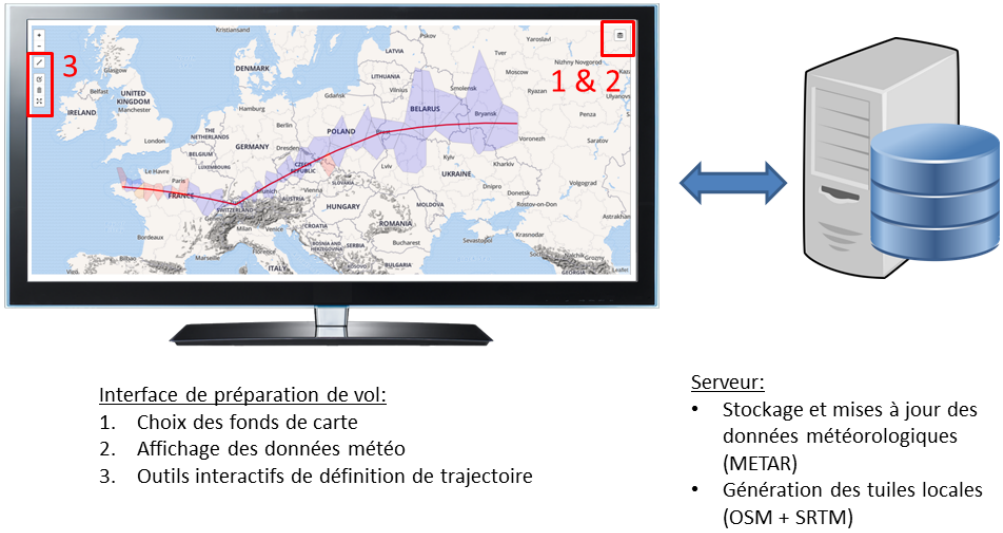

FIGURE 4 – Attendus couverts par le mémoire

2.6 Développements supplémentaires et pistes écartées

Adaptation à un environnement mobile

Bien que initialement destinée à un appareil mobile, l'interface accessible à l'utilisateur a été développée pour s'exécuter dans un navigateur web traditionnel, plaçant l'*adaptation à un environnement mobile* au rang de perspective. En effet, les solutions implémentées côté client sont transposables à un environnement mobile, mais nécessitent des adaptations dont la teneur ne fait pas réellement partie de la thématique de ce mémoire.

Rendu à 3 dimensions

Ensuite, il était initialement souhaité de fournir **un rendu à 3 dimensions** (3D) lors du développement sur l'appareil mobile. La piste du moteur de jeu vidéo "Unity", "game engine" multiplateformes fort répandu dans le monde du développement de jeux vidéos et permettant un développement d'applications sur tout support (Unity Technologies, s.d.), a alors été envisagée. En effet, celui-ci permet un affichage 3D. Il est possible d'y exploiter des données OSM ainsi que des modèles numériques de terrain (MNT) ou d'élévation (MNE) au format raster. Parmi ces dernières, figurent notamment les données SRTM [6] (Aitchison, 2013; Mapbox, 2017a). Toutefois, c'est une approche 2D traditionnelle avec des cartes ombrées sur base du relief qui a été conservée pour ce mémoire, bien que le traitement préalable des données SRTM y soit comparable (voir Section 5).

Notons également que des courbes de niveau traditionnelles auraient pu être déployées, mais leur interprétation nécessiterait trop d'attention de la part du pilote lors d'un vol. De plus, la seule information relative à l'altitude réellement requise par le pilote est la cote d'altitude minimale par carreau. Par conséquent, leur utilisation n'a pas été sollicitée et seule la perception du modelé du relief, fournie par l'estompage ("hillshading"), a été utilisée.

Données météorologiques GRIB

Les données météorologiques exploitées dans le cas présent sont de types METAR, mais d'autres données pourraient à terme être incorporée dans l'application. C'est le cas des *données météorologiques au format GRIB* qu'il est possible d'exploiter, afin d'obtenir des prévisions selon différents modèles météorologiques (pertinents selon l'échelle, l'étendue de la zone couverte, l'intervalle de temps évalué). En effet, le format

6. Shuttle Radar Topography Mission

GRIB (GRIdded Binary) contient des **données de prédictions issues de modèles météorologiques fonctionnant par simulation**. Les résultats sont présentés sur des grilles (grid), correspondant à un carroyage de points selon trois dimensions (World Meteorological Organization, s.d.). Ainsi, pour chaque variable météorologique, on obtient une superposition de grilles selon les différents niveaux d'altitude. Cette approche permet d'appréhender les différents phénomènes météorologiques selon 3 dimensions, en affichant les données météo propres à un palier d'altitude que l'on peut faire varier. Cela pourrait s'avérer particulièrement intéressant pour les données relatives à la vitesse du vent ou encore la couverture nuageuse dans notre cas.

Si l'on se base sur la description du logiciel libre Zygrib permettant d'appréhender facilement les données GRIB (Zaninetti, 2016), les variables météorologiques qui peuvent y figurer sont notamment :

- la pression atmosphérique au niveau de la mer,
- le vent à 10m du sol,
- les rafales de vent,
- la température à 2m du sol,
- la couverture nuageuse,
- le point de rosée à 2m du sol,
- ...

Il serait envisageable d'afficher certaines de ces couches de données (réparties selon un carroyage d'entités ponctuelles) en surcharge de la carte à la demande de l'utilisateur, de manière analogue à l'affichage des données METAR. Toutefois, leur affichage selon un support planimétrique tel qu'une carte nécessiterait que l'utilisateur précise l'altitude des données qu'il souhaite consulter.

Il serait donc possible d'intégrer ces données météorologiques au sein de l'interface, moyennant une précision sur l'altitude par l'utilisateur. Néanmoins, la prise en charge de ce type de données ne peut être faite à la légère et nécessiterait un traitement spécifique aux données météorologiques, ce qui laisse la porte ouverte à de nouveaux développements.

Données aéroportuaires AIXM

Enfin, l'*intégration des données aéroportuaires au format AIXM*[7], spécifique au contexte aéronautique, a été envisagée. Ce format de données aéronautiques vectorielles, basé sur le format "Geography Markup Language" (GML), a pour rôle de permettre la collecte, la vérification, la diffusion et la transformation des données aéronautiques officielles (Eurocontrol, 2017). Les données qu'il véhicule sont spécifiques aux aéroports : taxiways, terminaux, structure des espaces aériens, restrictions de vol, procédures, etc. Dès lors, on peut envisager un affichage à la demande de ces données en surchargeant les fonds de cartes ou encore en les y intégrant directement lors de la génération de tuiles par Mapnik. L'apport que procurerait ces données est considérable aux alentours des aéroports, leur concédant un rôle complémentaire aux données OSM.

En effet, leur contenu est dors et déjà disponible aux pilotes en sein de *cartes VAC*[8]. Celles-ci sont accessibles en ligne, sur le site de la vACC[9] pour les aéroports belges et luxembourgeois, mais sous forme de pdf pré-générés (Pol, s.d.). Un exemple d'une de ces cartes pour l'aéroport de Liège figure notamment dans l'Annexe I. On y observe entre autre :

- ◇ Les **différents espaces aériens règlementés** : les CTR[10], qui englobent les axes d'arrivées/départs et sont gérées par la tour de contrôle, ou les TMA[11], situées au-dessus des CTR et dont l'objectif est la protection des trajectoires de départ/approche ;

- ◇ Les **points d'entrée** associés aux différents espaces aériens contrôlés. Le passage par ces points est obligatoire pour entrer dans un espace aérien, sauf en cas de dérogation du contrôleur aérien ;

7. Aeronautical Information Exchange Model
8. Visual Approach Chart
9. Virtual Area Control Center
10. Control Traffic Region
11. Terminal manoeuvring area

- ◇ Les **balises de radionavigation** ainsi que leur fréquence ;
- ◇ Les **zones interdites** à la navigation ;
- ◇ Les **flight levels** correspondant aux altitudes de vol minimales au sein des zones ;
- ◇ Les **obstacles** notables à éviter : éoliennes, châteaux d'eau, usines, etc.

Toutefois, l'importation de ces données au sein d'une base de données spatiale nécessite une procédure de *parsing* spécifique qui a effectuée à l'aide de la plateforme FME (Safe Software, 2017) par l'équipe d'ESNAH parallèlement à la progression de ce mémoire. Dès lors, ces données ne seront pas traitées lors de ce travail mais leur prise en charge serait une amélioration certaine.

3 Architecture informatique

Ce mémoire ayant pour aboutissement l'élaboration d'un prototype en rapport avec le contexte aéronautique et fournissant une interface de visualisation de données de vol et météorologiques par-dessus des fonds de cartes sur mesure, il requiert un dispositif informatique circonstancié (Figure 5). Le fonctionnement de cette architecture, basée sur le principe client-serveur, est repris dans cette section mais les détails des opérations qui ont mené à son élaboration figurent dans l'Annexe D.

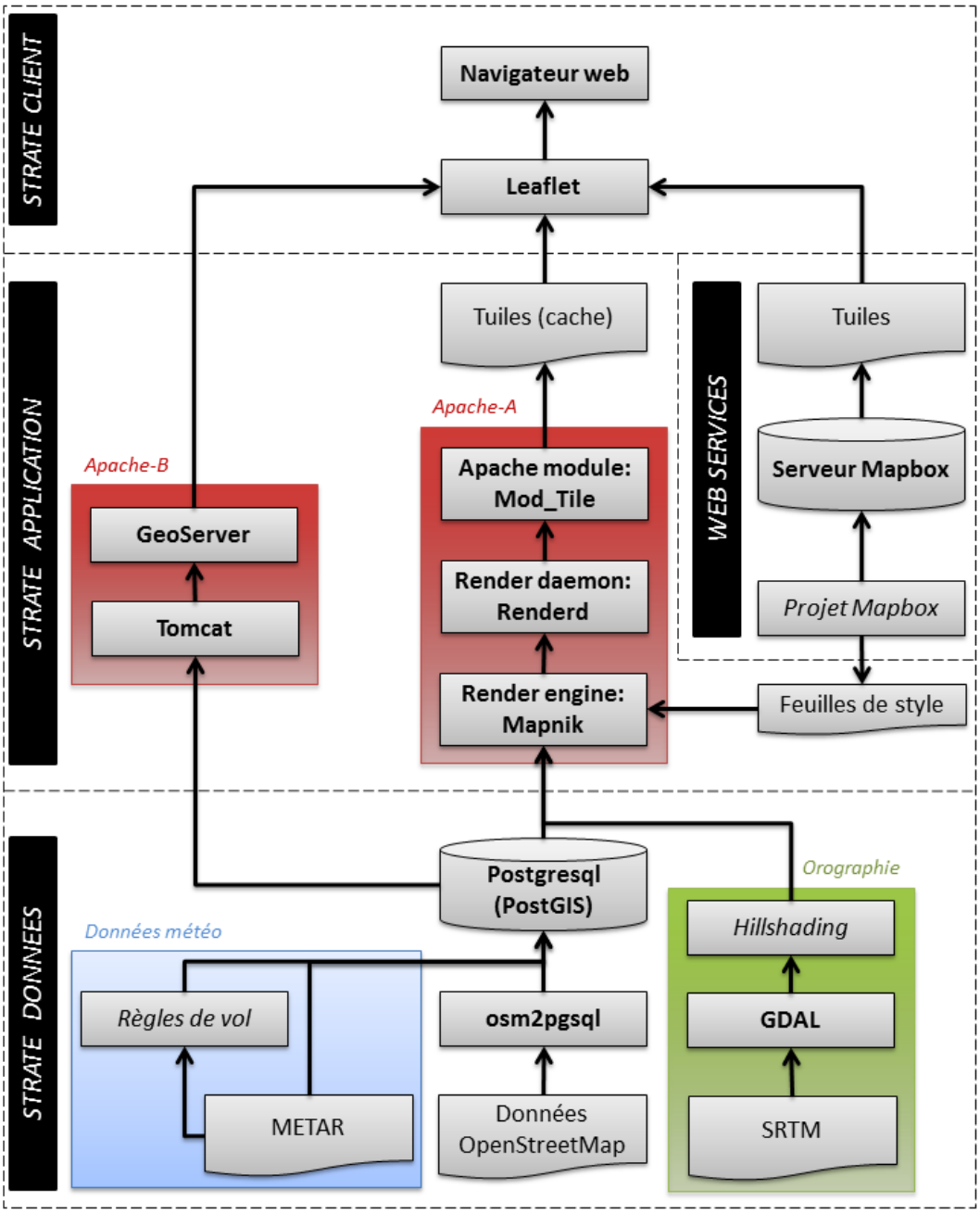

FIGURE 5 – Architecture informatique et fonctionnement du prototype

3.1 Contraintes matérielles et contextuelles

Selon les recommandations d'ESNAH et afin de pouvoir déployer les différentes composantes au sein de leur architecture serveur une fois abouties, ce prototype fonctionne sur base d'un **serveur GNU/Linux Ubuntu**. Cependant, le système d'exploitation utilisé en pratique s'avère être "Linux Mint", basé sur le noyau Ubuntu, avec un environnement de bureau Cinnamon. Il s'agit là d'un choix personnel, afin de bénéficier d'une interface plus conviviale et pratique d'utilisation, tout en étant basé sur les versions stables (LTS) d'Ubuntu (Communauté francophone d'utilisateurs d'Ubuntu, 2017b). Par défaut, n'importe quelle distribution Ubuntu peut servir de base à la construction d'un serveur. C'est d'ailleurs cette option qui a été préférée, afin de conserver une interface graphique. Toutefois, lorsqu'on cherche à bénéficier d'un noyau optimisé et à réduire les ressources consommées à cause d'une interface graphique trop gourmande, il est préférable de se tourner vers les versions "Ubuntu server" dédiées à cet effet (Communauté francophone d'utilisateurs d'Ubuntu, 2017c).

Par ailleurs, ce système d'exploitation présente, en plus de son caractère "open source", bon nombre de logiciels libres incontournables dans le domaine des SIG. De plus, son mode d'installation des logiciels facilite la compilation de programmes libres depuis leur code-source et permet une gestion des dépendances inter-logiciels. L'installation des logiciels et la configuration de ce serveur figurent en annexe (voir Annexe B).

Enfin, l'ensemble de logiciels sollicités relève des logiciels libres. Ceux-ci offrent une certaine transparence et la possibilité de coopérer par le biais de plateformes de développement, telles que github [12]. Ces plateformes permettant également d'héberger des projets et de les proposer en consultation et téléchargement à d'autres développeurs. Par ailleurs, cette orientation permet de s'émanciper des solutions propriétaires.

Notons également que ce prototype a été expérimenté sur un ordinateur portable personnel (modèle Acer 5810T) qui ne peut fournir les performances requises d'un "vrai" serveur. En effet, cet ordinateur offre de faibles performances CPU avec son processeur Intel Core 2 Solo SU3500 atteignant une fréquence de 1,4 Ghz et 4 Go DDR2-SDRAM. Il s'agit ici de tester la faisabilité du projet et non une réelle implémentation.

3.2 Stockage des données

3.2.1 SGBD Postgresql et PostGIS

Les données géographiques sont stockées sur un serveur incluant une base de données dédiée à cet effet. Cette dernière étant gérée par le SGBD PostgreSQL dont le support des données géographiques est assuré par l'extension PostGIS. Afin de pouvoir visualiser les données contenues dans la base, l'outil d'administration graphique (pour PostgreSQL) PgAdmin a également été exploité.

3.2.2 Données OpenStreetMap

Les données OpenStreetMap (OSM), destinées aux fonds de cartes tuilés, ont été téléchargées pour le territoire belge. En effet, de part les performances et l'espace de stockage requis côté serveur lors de la génération de tuiles (Section 3.3.1), le prototype ne fournira que des tuiles pour un échantillon : le territoire belge. La couverture d'un territoire plus vaste étant possible en téléchargeant un plus grand jeu de données mais non nécessaire dans le cas présent.

- ◇ **Téléchargement des données**
  Les données OSM sont disponibles sur divers serveurs. Le serveur utilisé lors de ce travail est celui de Geofabrik [13] : celui-ci permet de télécharger des extraits par pays ou régions.
  Les formats disponibles sont les suivants :
  - **Esri shapefile** (.shp) : format géographique vectoriel standard *de facto* ;
  - **Fichiel .osm.bz2** : fichier XML compressé selon l'algorithme de compression Bzip2 ;

12. https ://github.com
13. http ://download.geofabrik.de/

- **Fichiel .osm.pbf** : fichier XML compressé, cette fois selon la méthode Protocolbuffer Binary Format. Ce fichier est plus léger à télécharger et exploitable directement sans décompression par les logiciels supportant les données vectorielles OSM. C'est donc ce format qui a été privilégié.

A titre d'exemple, le jeu de données OSM mondial [14], au format XML non compressé, occupe 784.5 Go d'espace disque, alors qu'avec la compression bz2 et pbf, la taille est réduite à 56.3 Go et 35.2 Go respectivement (OpenStreetMap Wiki, 2017b).
Notons que les données OSM sont disponibles sous licence ODbL 1.0 [15] et libres d'accès (Section 5.1).

◇ **Import des données dans la base de données PostgreSQL**
L'insertion des données OSM dans une base de données s'effectue grâce à l'outil en ligne de commande **osm2pgsql** disponible directement sur la plateforme collaborative github (OpenStreetMap Foundation, 2017). Ce programme permet certaines fonctionnalités lors de l'import, telles que le changement du système de projection ou encore la configuration des noms de tables. Il a également pour avantage de formater les tables selon une configuration directement exploitable par Mapnik (3.3) et de pouvoir lire tous les formats de données OSM.

La commande ayant permis d'insérer les données au sein de la base ést du type :

```
osm2pgsql --slim -d gis ~/osm-BE/belgium-latest.osm.pbf
```

où -d précise le nom de la base de données, –slim active le mode "slim" où les données ne sont pas stockées dans la RAM lors du transfert mais dans des "tracking tables". Pour d'avantage de précisions sur l'utilisation d'osm2pgsql, les détails des opérations figurent dans l'Annexe B.

◇ **Organisation des tables OSM au sein de la base de données**
Grâce à l'outil d'import osm2pgsql, les données OSM sont insérées en respectant leur modèle conceptuel de données (voir Section 5.1). Il en résulte une série de tables affectées du préfixe planet_osm :
- planet_osm_line,
- planet_osm_point,
- planet_osm_polygon,
- planet_osm_roads.

Ces données sont directement exploitables en interrogeant la base de données via un client de type SIG logiciel. A titre d'exemple, la Figure 6 contient un aperçu du réseau hydrographique de premier ordre en Belgique, obtenu en interrogeant directement la base de données via QGIS.

14. Ensemble des données en date du 15/04/2017
15. Open Database License

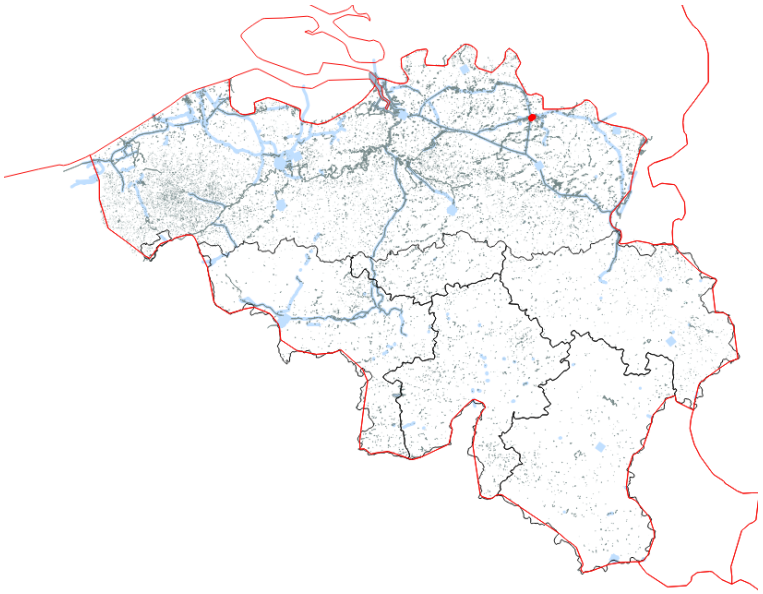

FIGURE 6 – Réseau hydrographique belge de premier ordre. Données : OpenStreetMap.

Ainsi, il était envisagé initialement d'**extraire les données reprises selon une classification** spécifique (décrite en détail dans la section 5.1) afin de ne **conserver que les données nécessaires à la génération des tuiles** au sein de notre contexte. Toutefois, cette approche a été écartée, car elle nécessiterait un tri des données en amont du tuilage. Ceci ralentirait le processus de mise à jour des données au sein du serveur tout en étant une source potentielle d'erreur en créant des incohérences entre données, après suppression de certaines d'entre-elles. En effet, les données OSM sont mises à jour dans leur entièreté chaque semaine et il serait hautement préférable d'éviter un tri hebdomadaire venant s'ajouter au processus de mise à jour. C'est l'usage d'un fichier de style, définissant les règles du rendu opéré par Mapnik (Section 3.3.2), qui permet d'éviter cette sélection au sein des données et de les garder intactes.

3.2.3 Données SRTM

Dans le cadre du rendu de cartes, un estompage (hillshading) des données SRTM est exploité afin d'offrir une perception du relief, en jouant sur les transparences des couches. Le fichier contenant ce hillshading est au format GeoTIFF, permettant d'ajouter des métadonnées géographiques à une image TIFF (Tag Image File Format), comme les fichiers originaux du SRTM (Section 5.3).

L'approche choisie pour l'inclusion du SRTM est de conserver celui-ci dans un fichier (GeoTIFF) et de pointer vers son répertoire. Toutefois, une autre solution aurait été d'insérer celui-ci dans la base de données PostGIS et de bénéficier du . Selon Kresse (2012), cela aurait eu pour conséquences :

- de centraliser la gestion des données,
- d'offrir un support pour plusieurs utilisateurs,
- de fournir une sécurité quant à l'accès des données,
- de proposer un traitement transparent des requêtes,
- de rendre les interrogations de la base possibles par le biais du langage SQL.

Cette seconde approche, renvoyée au rang de perspective, serait celle à favoriser dans le cas d'un déploiement du prototype dans un cadre professionnel, mais nécessite une gestion spécifique au sein de la base. En effet, la gestion de ces données raster par un SGBD serait **justifiée pour un territoire plus ample**. Celui-ci serait alors découpé en tuiles lors de l'import et l'accès à ces données (spatiales) serait plus rapide qu'au sein d'un seul fichier image. Néanmoins, les gains potentiels n'étaient pas justifiés pour un jeu de données de

l'étendue utilisée. Par ailleurs, cette approche trouve son sens lorsque les raster nécessitent des traitements "à la volée" directement au sein de la base de données or, dans le cas présent, ces fichiers ne sont utilisés qu'en consultation.

3.2.4 Données météorologiques

Les données METAR ainsi que les zones d'influence des stations (Section 4) sont exploitées dans le format SpatiaLite, l'extension spatiale du format SQLite. Ces données figurent donc dans un fichier qui, lors d'une implémentation réelle, sera mis à jour lorsque les nouvelles données sont disponibles (Section 3.4.3).

Ensuite, ces données sont importées dans la base de données PostgreSQL grâce à l'outil de conversion "ogr2ogr" intégré à la librairie GDAL (GDAL, 2017b). Ainsi, un import de ces données selon un système de coordonnées WGS84 (EPSG : 4326) dans la même base de données que les données OSM (gis) s'effectue via la commande suivante :

```
ogr2ogr -append -lco GEOMETRY\_NAME=geometry -lco SCHEMA=public
        -f "PostgreSQL" PG:"host=localhost port=5432 user=postgres dbname=gis"
        -a\_srs "EPSG:4326" metar.sqlite
```

L'opération étant exécutée depuis la machine hôte (host= localhost), en éditant les couches si elles existent (-append), en précisant le champ des géométries (-lco GEOMETRY_NAME=geometry) ainsi que le schéma où importer les données au sein de la base (-lco SCHEMA=public).

3.3 Rendu des données planimétriques

3.3.1 Stratégie de tuilage

Les premières applications de cartographie en ligne, bien que assez rudimentaires, datent de la fin des années 1990 (Sample, 2010). On peut mentionner "Yahoo!Maps", "MapQuest" ou encore "TerraServer". Ces applications offraient un rendu relativement lent, car elles ne généraient qu'une seule (grande) image selon les couches de données souhaitées et l'étendue géographique de la fenêtre du client, destinée à contenir la carte. Ces applications demandaient des plugins spécifiques à ajouter au navigateur ou des plateformes de développement telles que Java ou Flash.

Ce n'est qu'en 2005, avec l'apparition de "Google Maps", que la cartographie en ligne a évolué vers un concept que l'on connait toujours actuellement : le tuilage (Neogeo-online, 2012). D'aucuns disent que *"Google Maps a changé la façon dont nous interagissons avec les cartes"* (Peterson, 2012). Le tuilage, qui est devenu un standard *de facto*, puis un standard de l'OGC avec la norme WMTS (voir ci-après), permet la création d'interfaces de type "Slippy Map" (ou carte glissante en français) qui offrent à l'utilisateur la possibilité de se déplacer et de "zoomer" au sein de la carte. Notons que les niveaux de zoom disponibles correspondent, dans ce cas, à des échelles de valeur pré-définie.

- ◇ **Principe de base du tuilage**
  Les fonds de cartes sont divisés en plus petites images, les tuiles. Par comparaison à celles d'un toit, les tuiles sont organisées conjointement en mosaïque pour construire le résultat attendu. A chaque niveau de zoom correspond une mosaïque (et donc une série de tuiles) différente.

- ◇ **Dans la pratique**
  Selon le principe de base (et le plus primitif) du tuilage, les tuiles sont générées à l'avance et stockées sur le serveur, afin de réduire les traitements demandés. En effet, puisqu'il n'est plus nécessaire de générer ces tuiles sur demande et que celles-ci sont déjà en mémoire, le serveur est capable de les envoyer plus rapidement. De plus, lorsqu'on combine cette anticipation avec la possibilité de stocker certaines données en mémoire cache (du côté client et/ou du côté serveur), la navigation s'en trouve grandement fluidifiée par réduction des temps de chargement. Il convient toutefois de se doter de mécanismes de vérification de l'actualité des tuiles stockées en cache, par exemple, en leur attribuant une durée de vie limitée. Par ailleurs, lors des déplacements au sein de la carte, seules les tuiles manquantes de la

fenêtre d'affichage doivent encore être téléchargées. Notons également que les tuiles jouxtant la fenêtre d'affichage sont généralement pré-chargées par le client, afin d'anticiper les déplacements.
Cette stratégie est celle adoptée initialement par "Google Maps". Toutefois, des besoins plus exigeants (plus grand nombre de couches disponibles, réduction du coût du stockage, nécessité de mise à jour des données plus régulièrement, etc.) ont conduit, successivement, à l'élaboration de la spécification TMS, puis au standard WMTS de l'OGC (voir ci-après) pour des applications de tuilage plus élaborées.

◇ **Propriétés du tuilage**
Selon Sample (2010), *"les propriétés du tuilage se décomposent en deux catégories :*

1. ***Les propriétés indispensables :***
   - *Les tuiles sont générées sur base de niveaux de zoom pré-définis et correspondant à une échelle précise,*
   - *Une vue dans la fenêtre d'affichage utilise plusieurs tuiles,*
   - *Les tuiles sont accessibles via un schéma d'adressage direct,*
   - *Les tuiles stockées côté serveur sont envoyées au client avec un traitement minimal.*
2. ***Les propriétés (importantes mais) optionnelles :***
   - *Les tuiles ne sont générées que selon une projection globale,*
   - *Les tuiles sont distribuées grâce à une architecture client-serveur,*
   - *Les tuiles sont organisées en peu de couches (qui sont fixes)."*

◇ **Le schéma de tuilage**
La correspondance entre les coordonnées géographiques de l'étendue de la fenêtre d'affichage et les tuiles correspondantes s'effectue par le biais d'un schéma de tuilage. Selon Sample (2010) toujours, *"un schéma de tuilage est construit sur :*
- *La définition de l'adressage discret des tuiles,*
- *La définition des tuiles en fonction des différents niveaux de zoom,*
- *La définition de la méthode de traduction entre l'adresse de la tuile et les coordonnées géographiques."*

◇ **La carte glissante ou "slippy map"**
L'application de tuilage la plus simple est basée sur un usage de la technologie AJAX[16]. Celle-ci permet aux pages web d'exécuter des tâches de gestion de données et d'envoi de requêtes aux serveurs web à l'arrière plan, sans que l'utilisateur ne s'en rende compte (Nixon, 2015). Cet ensemble de méthodes JavaScript assure un transfert de données entre le client et le serveur sans imposer de recharger la page. Dès lors, le mode de fonctionnement de la carte glissante est proche de celui initialement proposé par "Google Maps", à l'exception des tuiles qui ne sont pas systématiquement pré-générées et directement disponibles en mémoire.

Ainsi, un scénario type de requête de tuiles est le suivant (Figure 7) :

1. Le client utilise une application de carte glissante par le biais de son navigateur ;
2. L'application cartographique identifie la série de tuiles dont elle a besoin pour réaliser une vue couvrant l'étendue géographique ;
3. Le navigateur envoie une requête contenant les identifiants (ligne, colonne) de chaque tuile et le niveau de zoom (selon plusieurs valeurs discrètes possibles) au serveur cartographique ;
4. Le serveur réceptionne la requête et, à partir des paramètres contenus dans l'URL, déduit les tuiles à renvoyer ;
5. Le schéma de tuilage permet au serveur d'identifier les fichiers de tuiles pré-générées à récupérer sur le disque dur (adressage) ;
6. Les tuiles sont envoyées au client (dans un format image) ;
7. L'application de carte glissante assemble et organise les tuiles dans la fenêtre d'affichage.

16. Asynchronous JAvascript and Xml

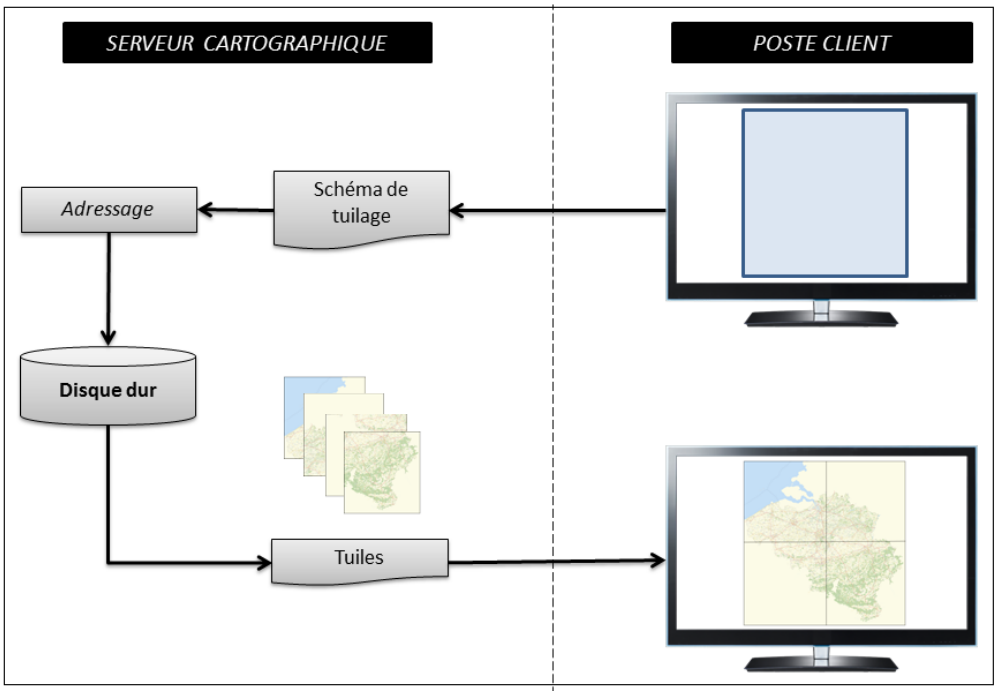

FIGURE 7 – Principe de base du tuilage

◇ **Le tuilage et les projections cartographiques**
Lorsque l'on cherche à représenter les données géographiques selon deux dimensions, il faut recourir à une projection cartographique. Bien que d'autres projections soient possibles, la plupart des applications de cartographie en ligne de type "mosaïque de tuiles" recourent à la projection "Pseudo-Mercator" (EPSG :3857), ce qui lui aura d'ailleurs valut le nom de projection "Web Mercator". Elle est construite sur une projection cylindrique de Mercator en aspect direct et en considérant des coordonnées ellipsoïdales (WGS84). Toutefois, ce sont les développements sphériques qui sont utilisés sur ces coordonnées géodésiques car ils sont plus simples à implémenter et permettent d'obtenir une approximation rapide de la projection réalisée à partir de l'ellipsoïde(International Association of Oil And Gas Producers, 2017). Il en résulte que les angles, initialement conservés au sein de la projection basée sur l'ellipsoïde, ne sont pas précisément conservés dans ce cas. Toutefois, en première approximation, les loxodromies y sont représentées selon des trajectoires quasi-rectilignes. Cet aspect a son importance dans le cas de la navigation aérienne où les trajectoires sont réalisées selon des caps.

Par ailleurs, cette projection n'est pas équivalente. Dès lors, les données tuilées présentent peu de fiabilité quant à la conservation des surfaces : les superficies sont grandement exagérées lorsqu'on approche des pôles.

Cette projection a été conservée pour le développement de ce prototype, afin de pouvoir faire cohabiter les tuiles générées localement et celles obtenues par des web services au sein d'une même interface. En effet, ces dernières sont disponibles selon la projection "Pseudo-Mercator" mais, lors du déploiement d'une plateforme réelle de tuilage destinée à l'aéronautique (par exemple inspirée de ce prototype), il est fortement recommandé de recourir uniquement à la projection "WGS 84/World Mercator" pour la génération des tuiles.

◇ **Nombre de tuiles et niveaux de zoom**
Il apparaît rapidement que le nombre de tuiles augmente avec le niveau de zoom. En effet, le principe du tuilage est de doubler le nombre de lignes et de colonnes couvrant une même étendue (en conservant la même résolution pour chacune des tuiles), lorsque le niveau de zoom est incrémenté (Figure 8).

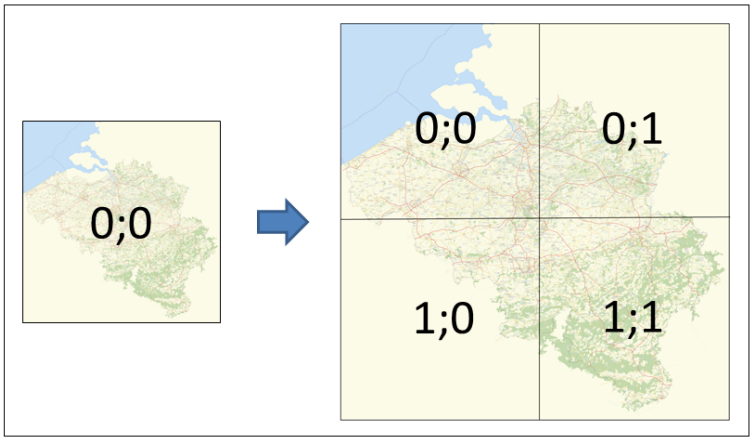

FIGURE 8 – Augmentation du niveau de zoom lors d'un tuilage sous forme matricielle

Ainsi, pour une Terre sphérique que l'on représente selon une projection "plate carrée" (projection cylindrique tangente à l'équateur), simple à implémenter, on obtient une représentation rectangulaire dont l'extension en x (correspondant à 360 °) vaut le double de l'extension en y (correspondant à 180 °).

Par conséquent, cette forme s'avère particulièrement intéressante dans le cadre du tuilage : le niveau 1 correspondant à une représentation planimétrique divisée en deux selon le méridien de Greenwich. Ensuite, les tuiles sont subdivisées selon la moitié de leur largeur et la moitié de leur hauteur. Ainsi, une même tuile donne naissance à 4 tuiles du niveau de zoom supérieur.

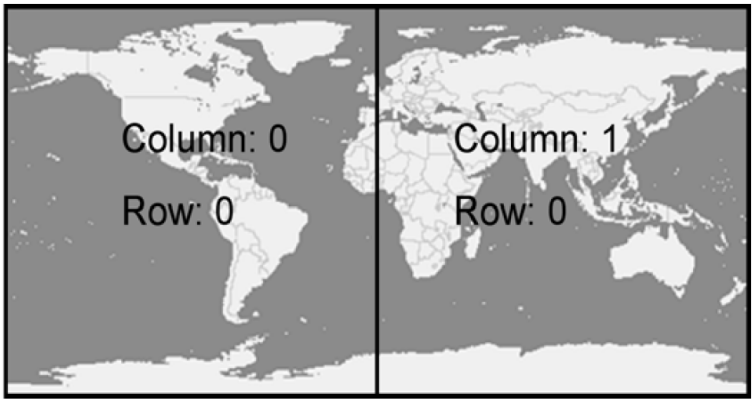

FIGURE 9 – Subdivision en tuile selon le niveau de zoom 1 (Sample, 2010).

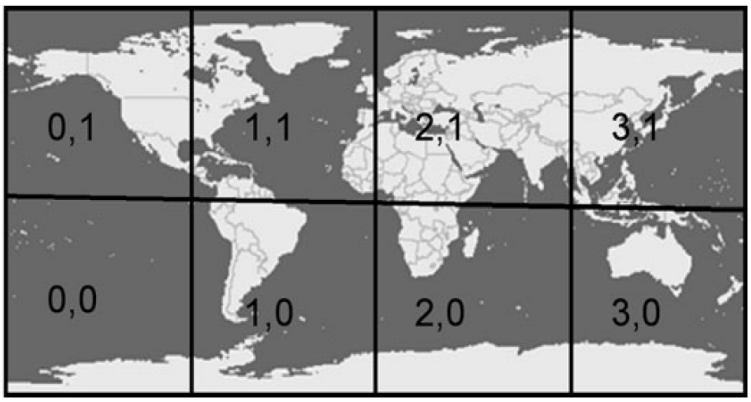

FIGURE 10 – Subdivision en tuile selon le niveau de zoom 2 (Sample, 2010).

Un tableau reprenant le nombre de tuiles (en ligne et en colonne) selon les niveaux de zoom ainsi que les degrés par pixel (pour une image de 512x512 pixel) figure en Annexe F. On remarque que le nombre de tuiles augmente très rapidement, ce qui se traduit également par un espace disque alloué aux tuiles augmentant avec les niveaux.

Ainsi, si l'on se base sur l'exemple d'OpenStreetMap avec un espace disque de 633 octets par tuile en moyenne[17] (OpenStreetMap Wiki, 2016c), on obtient les résultats suivants pour les niveaux de zoom les plus élevés, ainsi que pour le total des 20 niveaux :

	Nombre de tuiles	**Espace disque (o)**	**Espace disque (Go)**	**Espace disque (To)**
	1	633	6,33 10^{-13}	6,33 10^{-10}
Niveau 18	34359738368	21749714386944	21749,71	21,75
Niveau 19	137438953472	86998857547776	86998,86	87,00
Niveau 20	549755813888	347995430191104	347995,43	348,00
Total	733007751850	463993906921050	463993,91	463,99

TABLE 2 – Stockage de tuiles OSM sur disque.

On observe que le stockage du niveau de zoom le plus élevé, a lui seul, requiert 348 téraoctets. Le total quant à lui plafonne à 463,99 téraoctets. Considérant cela et sachant que ces données sont mises à jour de manière hebdomadaire, on comprend que l'ensemble des tuiles n'est pas pré-généré au-delà de certains niveaux de zoom.

◇ **Le standard Web Map Service (WMS)**
Ce standard est *"un service web standardisé produisant des cartes (en tant que fichier image numérique) de manière dynamique à partir d'informations géographiques"* (ISO, 2005). Une implémentation basique d'un servlet WMS doit répondre à 2 requêtes (Sample, 2010) :
- **GetCapabilities :** qui demande un document XML[18] contenant les métadonnées du service (structure de la matrice et description de la fonction GetMap) ;
- **GetMap :** qui récupère une carte selon une étendue géographique (la "bounding box" de la fenêtre d'affichage) et des paramètres spécifiés (dimensions de l'image).

Ce type de service web n'étant pas conçu pour le tuilage, cela a encouragé l'OGC[19] a créer le standard WMTS par la suite.

17. Moyenne mesurée en mars 2011
18. Extensible Markup Language ou language de balisage extensible en français
19. Open Geospatial Consortium

◇ **Le standard "Web Map Tile Service" (WMTS)**
Cet autre standard définit les capacités d'un serveur de tuiles, mais également la manière d'interagir avec elles(OGC, 2010). Son fonctionnement repose sur une gestion des tuiles de manière matricielle, sans toutefois imposer de schéma de tuilage, de projection ou de résolution d'image. Cela limite l'inter-opérabilité entre différents WMTS : il est nécessaire pour le client de connaître les contraintes du serveur (grille de tuilage, niveaux de zoom, projection et liste des couches disponibles). Néanmoins, cette norme offre une approche standardisée, afin de converger vers un seul type de client compatible avec tous les serveurs WMTS.

	WMS	**WMTS**
Étendue géog. de la fenêtre	Quelconque (↗ flexibilité, ↘ performances)	Fonction de niveaux de zoom prédéfinis (↗ performances, ↘ flexibilité)
Utilisation de la mémoire cache	Non*	Oui

TABLE 3 – Comparaison entre le WMS et le WMTS

◇ **Cohabitation WMS et WMTS**
Il est possible d'exploiter des tuiles WMTS, disponibles en mémoire cache, afin d'accélérer la réponse à une requête GetMap (Figure 11). C'est d'ailleurs ce qui est effectué par MapCache au sein de la plateforme de publication de services web MapServer, lors d'une requête GetMap (Open Source Geospatial Foundation , 2017c). En effet, bien que cette dernière attende une réponse au format WMS, les tuiles peuvent être fusionnées, ré-échantillonnées et découpées, afin de fournir une image correspondant à la requête. On évite donc de (re)générer une représentation à la demande, et le WMTS devient complémentaire au WMS. En revanche, si aucune tuile WMTS adéquate n'est présente en mémoire, le scénario classique de carte à la demande est exécuté : reprojection et ensuite symbolisation des données vectorielles.

Ainsi, la stratégie à adopter sur un serveur cartographique, quant à la gestion et la diffusion des services (WMS/WMTS) à prévoir est variable. En effet, celle-ci dépend :

- **Du Système d'Information** disponible. De lui dépendent les moyens informatiques à disposition pour le déploiement des services ;
- **De la fréquence des mises à jours**. Des données souvent mises à jour ne sont généralement pas pré-calculées, mais plutôt générées à la demande (WMS) ;
- **Des fonctions à la demande**. Si des options sont offertes à l'utilisateur, une gestion à la demande est préférable (WMS) ;
- **De la fréquence d'utilisation**. Si des données sont régulièrement utilisées, il peut être avantageux de les pré-générer et de les garder en mémoire cache (WMTS) ;
- **Du coût de la génération**. Si des données nécessitent beaucoup de ressources pour être générées, il est préférable d'éviter toute dépense inutile et opter pour une gestion à la demande (WMS).

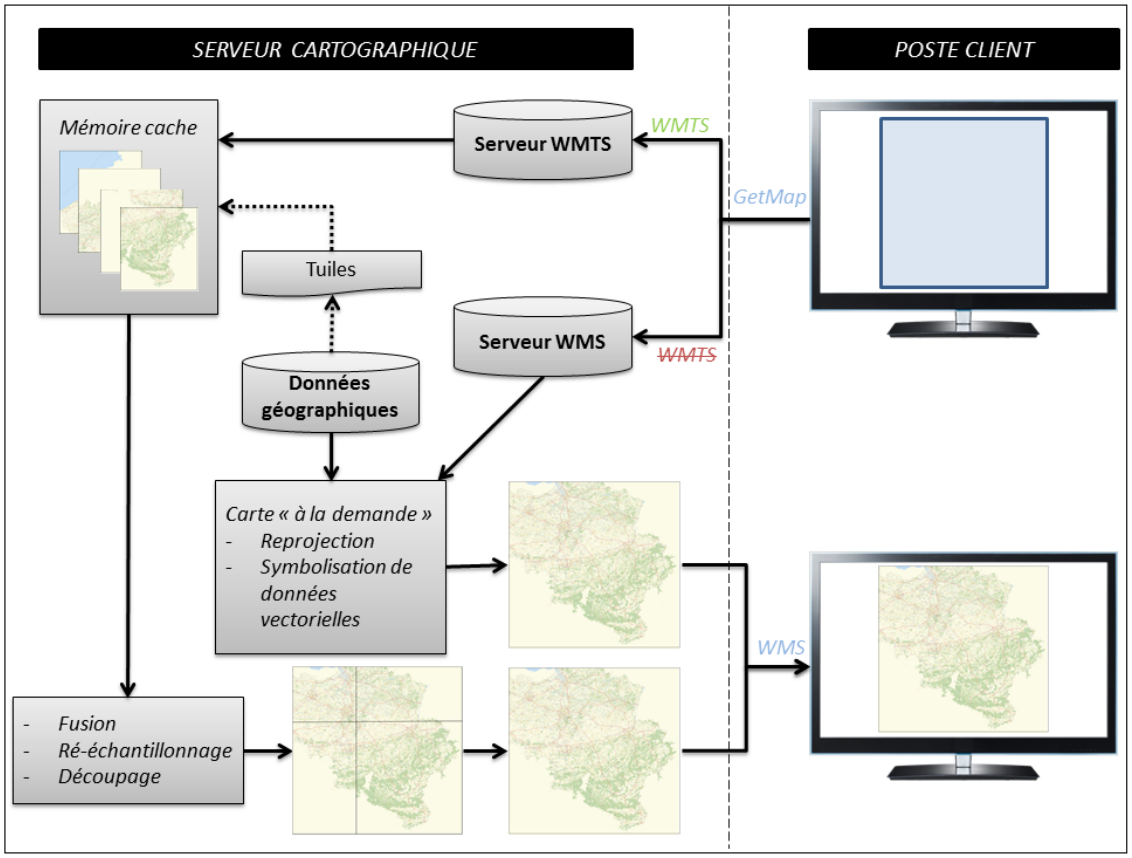

FIGURE 11 – Requête GetMap sur un serveur combinant WMS et WMTS

◇ **La spécification "Tile Map Service" (TMS)**
Cette spécification est une ligne de conduite pour l'élaboration de solutions de cartographie utilisant des pyramides d'images multi-résolution selon une architecture client-serveur (OSGeo, 2012). Elle contient les instructions pour les requêtes de tuiles effectuées par les clients, en utilisant des méthodes du principe REST au sein du protocole HTTP. Le style d'architecture REST, basé sur la définition du protocole HTTP, permet de référencer les ressources directement au sein d'une URL dans un environnement client-serveur (Fielding, 2000). Ces ressources sont, dans le cas présent, les paramètres ligne, colonne et niveau de zoom des tuiles.

La spécification TMS évoque également les procédés selon lesquels les serveurs de tuiles précisent leur fonctionnement. Cette spécification, plus élaborée que le modèle "tout public" de carte glissante, permet au serveur de supporter d'autres systèmes de coordonnées. Elle n'est en revanche pas aussi complexe que les standards WMS/WMTS, qui sont exclusivement destinés à des professionnels. Notons également que le système d'axes implémenté en TMS est différent du standard WMTS. En effet, alors que l'axe y du système WMTS pointe vers le bas, celui du TMS est orienté vers le haut. Les autres axes (x et z), quant à eux, restent similaires. Par conséquent, une transformation selon l'axe y est nécessaire si l'on veut passer d'un système à l'autre.

3.3.2 Mapnik

Mapnik est un logiciel libre multi-plateforme de **rendu cartographique** (OpenStreetMap Wiki, 2017a) : à partir de données géographiques, il génère une image selon les critères demandés. Ainsi, il permet de développer des applications cartographiques en ligne basées sur un principe de tuilage, telles que le portail OpenStreetMap, ou de simples cartes.

Mapnik et les données géographiques

Ce programme est principalement écrit en C++ mais comporte quelques scripts en python. Ceci s'avère particulièrement intéressant car il est, dès lors, aisé d'utiliser la bibliothèque GDAL depuis Mapnik pour le traitement de données géographiques aussi bien raster (via GDAL) que vectorielles (via la bibliothèque OGR) (GDAL, 2017a). Bien que Mapnik supporte une multitude de formats, il est initialement conçu pour le rendu de données OSM.

Rendu et définition des styles

Pour effectuer l'opération de rendu, Mapnik utilise la librairie AGG qui permet de réaliser un antialiasing[20] jusqu'au sous-pixel (AGG Project, 2006). Cette librairie contient un ensemble d'algorithmes (dont une implémentation de l'algorithme de Bresenham) qui permettent d'effectuer un rendu 2D automatique de haute qualité. L'apparence et l'habillage de la carte sont gérés dans un **fichier de style au format XML** qui est consulté par Mapnik lors du rendu. Au sein de ce fichier, figurent les différentes couches de données (OSM, PostGIS, GeoTIFF, shp, etc.) et le rendu, qui leur est affecté par l'intermédiaire de "styles" et de "règles" : la couleur, la transparence ou encore la typographie y sont spécifiées. Par ailleurs, ce fichier permet de définir des rendus différents dans une application de tuilage, selon l'échelle et le niveau de zoom.

Afin de comporter une syntaxe plus facilement interprétée par l'utilisateur, ce format de fichier a progressivement évolué vers ce qui est appelé le **"carto CSS"** de Mapbox (OpenStreetMap Wiki, 2016a), puis vers le **"Mapbox GL"**(MacWright, 2016) : la syntaxe XML y a laissé successivement la place à un langage comparable au CSS, puis à un langage basé sur le JSON (JavaScript Object Notation). Ce dernier permet, tout comme le XML, de structurer les informations selon des étiquettes, et en utilisant des notations dérivées du langage JavaScript.

La **création de styles** peut, dès lors, être réalisée au sein d'interfaces plus conviviales et par le biais d'un langage plus adapté à l'habillage. A cet effet, l'interface de "Mapbox Studio" met à disposition des outils de création/édition de styles dans le bus de réaliser des cartes en ligne selon un rendu personnalisée. Au sein de cette interface, les tuiles sont pré-générées dynamiquement, afin de permettre à l'utilisateur d'avoir directement un aperçu du style en édition. Par ailleurs, des styles pré-existants tels que "OSM Bright" sont disponibles, afin de servir de base à l'élaboration d'autres styles. Notons également que ces styles peuvent être visualisés au sein d'une page web par le biais l'API de cartographie en ligne Mapbox.js. C'est notamment ce qui a été implémenté au sein de ce prototype, afin de bénéficier de tuiles dont le rendu est réalisé rapidement. En effet, la génération locale de tuiles par Mapnik, selon la configuration déployée, ne permet pas un affichage "dynamique" suite au temps d'exécution élevé du processus de rendu. Toutefois, cette configuration devrait présenter de bien meilleurs résultats une fois déployée sur un ordinateur adéquat de type "serveur".

Ensuite, ces nouveaux types de styles sont exportables et doivent ensuite être **convertis en un fichier XML**, par le biais du programme "carto", pour obtenir un fichier de style exploitable par Mapnik. Ce programme, fonctionnant grâce à la plateforme serveur JavaScript orientée applications *Node.JS*, est intégré aux applications de CartoCSS telles que TileMill de Mapbox. Toutefois, cet outil est également accessible en ligne de commande, ou peut être intégré dans un navigateur web (Mapbox, 2017).

Dans le cas de rendu de données OSM, il est possible de se procurer librement des exemples de fichiers de style selon l'apparence qu'on souhaite. En effet, il existe plusieurs projets de cartographie en ligne exploitant les données OSM, selon différentes thématiques. C'est vers l'un d'eux que je me suis orienté dans le cadre de ce mémoire : le projet OSM Bright (Mapbox, 2016).

Mapnik au sein d'un serveur de tuiles

Lorsque l'on souhaite exécuter Mapnik dans un environnement client-serveur, afin de déservir des tuiles à partir d'une base de données géographiques, il convient de combiner son utilisation avec un serveur web, Apache dans le cas présent (Figure 5). A cet effet, deux programmes conçus pour fonctionner en combinaison avec Mapnik dans le contexte OpenStreetMap ont été utilisés : le module Apache "Mod_Tile", ainsi que

20. L'antialiasing (ou anticrénelage) est une opération permettant de corriger un effet visuel "en forme d'escalier" (le crénelage) lors du redimensionnement d'images.

le daemon[21] de rendu "Renderd" (OpenStreetMap Wiki, 2016b). Associés, ces programmes fournissent une application de tuilage de type *carte glissante* (ou *"slippy map"*).

3.3.3 Mod_Tile et Renderd

Renderd

Le daemon informatique Renderd (par convention, les programmes de type daemon se terminent par un "d") permet de réaliser un **rendu à la volée** selon les besoins (hiérarchisés dans un mécanisme de file d'attente), en sollicitant Mapnik. Ainsi, aucune tuile n'est pré-calculée de manière systématique, et seule une fraction est stockée sur le disque. Les autres étant réalisées au fur et à mesure sur demande de Mod_Tile.

Il est à noter que Renderd procède au rendu de chacune des tuiles (généralement 256x256 pixels) de manière individuelle, avant que celles-ci ne soient regroupées en "*metatiles*" de 8x8 tuiles, ce qui correspond à 2048x2048 pixels (OpenStreetMap Wiki, 2010; OpenStreetMap, 2017). Cette approche offre différents avantages :

— **Augmentation de l'efficacité de rendu**
En se basant sur l'hypothèse qu'un client ne sollicitera pas qu'une seule tuile au sein de sa carte, il semble plus adéquat de **générer plusieurs tuiles simultanément** par le biais d'une *metatile*. En effet, redémarrer le processus de rendu de Mapnik, qui doit interroger la base de données et ouvrir le fichier de style, pour chacune des tuiles, serait plus lent.
Par ailleurs, si toutes les tuiles d'une *metatile* ne sont pas nécessaires au client directement, elles peuvent toutefois l'être pour un autre par la suite. Pensons aux serveurs à forte affluence. Ces tuiles supplémentaires ne sont donc pas tout à fait "perdues".
De plus, si la génération de *metatiles* a été entraînée par l'obsolescence de tuiles après une mise à jour des données (voir le paragraphe consacré à Mod_Tile), on peut émettre l'hypothèse que ces tuiles, ayant déjà été requises, sont susceptibles d'intéresser de nouveaus clients dans le futur.

— **Simplification des emplacements d'étiquettes de texte**
Le rendu des éléments textuels (tels que les étiquettes associées à une route ou à un nom de lieux) est bien meilleur lorsque l'on envisage une **vue selon plusieurs tuiles conjointes**. On évite ainsi une surcharge, de même qu'une gestion complexe du placement des étiquettes pour le moteur de rendu, en diminuant les chevauchements et doublons éventuels.

— **Amélioration de l'efficacité de stockage et de transfert**
La gestion des *metatiles* nécessite de stocker 64 fois moins de fichiers. En effet, bien qu'une *metatile* soit décomposable en tuiles, son stockage est individuel.

Mod_Tile

Le module Apache Mod_Tile organise la **gestion de la mise en mémoire** cache des *metatiles* compilées par Renderd et ce, selon plusieurs styles si nécessaire. Par ailleurs, il gère le traitement des requêtes de clients de manière équitable, de même que la validité des tuiles après une mise à jour des données. En effet, lorsqu'on modifie les données d'origine, les tuiles qui en découlent doivent être régénérées puis ré-assemblées en *metatiles*. Dans ce but, Mod_Tile compare la date de la tuile en mémoire à la date de la dernière importation complète de données dans la base : si la date de la tuile est antérieure à la mise à jour, elle est déclarée obsolète (ou "dirty", qui signifie "sâle" en anglais, dans le jargon OSM). Ce qui implique que cette tuile, lors de sa prochaine sollicitation, devra être régénérée. Toutefois, celle-ci ne sera pas régénérée tant qu'elle n'a pas été requise. On évite ainsi d'occuper de l'espace mémoire inutilement. Par ailleurs, toutes les tuiles sont, logiquement, déclarées obsolètes après une mise à jour complète des données : il n'est pas nécessaire de les évaluer individuellement.

Notons toutefois que, si la mise à jour de données est partielle, seules les tuiles en rapport avec les données mises à jour sont déclarées obsolètes en modifiant artificiellement leur date, afin que celle-ci soit antérieure à

21. Un daemon informatique est un processus qui s'exécute en arrière-plan sans intervention de l'utilisateur et souvent lancé automatiquement au démarrage du système d'exploitation.

la dernière mise à jour complète des données.

3.4 Gestion des données météorologiques

Alors que les données météorologiques sont stockées au sein d'une base de données PostGIS une fois leurs traitements effectués, leur gestion s'effectue à l'aide de *Geoserver*. Celui-ci prend en charge les données géographiques sur un serveur web (Open Source Geospatial Foundation , 2017a). Il propose une interface homme-machine pour gérer des services web, tels que WMS, WMTS ou encore WFS. Dans le cas présent, Geoserver est implémenté en tant que *servlet*[22] de Tomcat sur base d'un serveur web Apache.

3.4.1 Tomcat et Apache

Tomcat est un servlet engine contenant également un serveur HTTP (The Apache Software Foundation, 2017). Il permet d'exécuter des servlets JAVA, via la machine virtuelle associée (JVM pour "Java Virtual Machine"). Il rend dès lors des services informatiques sur un réseau (Figure **??**) (Coward, 2001). Notons que, dans le cadre de ce mémoire, c'est l'environnement JAVA d'Oracle qui a été utilisé, car Tomcat est optimisé pour fonctionner selon ce JRE. Tomcat est souvent utilisé en association avec un serveur web plus généraliste tel qu'**Apache**, lui permettant de déléguer les tâches traditionnelles à celui-ci.

Notons qu'il est nécessaire d'autoriser certaines requêtes "cross plateform" par l'ajout d'un "CORS[23] filter" à la configuration initiale de Tomcat. En effet, le mécanisme de sécurité recommandé par le World Wide Web Consortium (2014) et associé à l'exécution du JavaScript restreint les requêtes possibles aux applications web : celles-ci ne peuvent obtenir des données extraites d'une autre origine. Dès lors, des spécifications d'authentification ont été élaborées en conséquence afin d'autoriser les requêtes nécessaires à l'échange des données météo.

3.4.2 Geoserver

Bien qu'une version autonome incluant son propre serveur web (Jetty) soit également disponible, Geoserver a été déployé en tant que **servlet de Tomcat**, car celui-ci propose une documentation plus fournie et son utilisation est plus répandue sous cette forme.

Format supportés en entrée

Pour le traitement des données géographiques, Geoserver utilise la bibliothèque Java "GeoTools" (Open Source Geospatial Foundation , 2017b). Celle-ci lui permet de supporter les formats spécifiques au contexte de la géographie tels que :

- PostGIS,
- ESRI Shapefile,
- SpatiaLite,
- Oracle Spatial,
- GeoTIFF,
- ...

Service "Web Feature Service" (WFS)

Les services disponibles en sortie de Geoserver sont multiples. C'est un service en particulier qui a été sollicité : le WFS. En effet, celui-ci permet de récupérer des données géographiques vectorielles (géométrie et attributs) par le biais d'une URL (Open Source Geospatial Foundation , 2017f). Ce protocole standardisé de l'OGC prend en charge la requête, ainsi que la récupération des données :

1. **La requête** s'effectue par le protocole HTTP en insérant les paramètres dans l'URL : il est possible de créer, mettre à jour ou effacer des objets, mais également de rechercher des objets géographiques à partir d'une requête spatiale.

22. Mélange des mots "server" et "applet", un servlet Java est un programme exécuté sur un serveur web. Son rôle sera d'exécuter des services (côté serveur) selon le protocole HTTP (ou autre) et les résultats seront renvoyés au client dans un langage de balisage (html, JSON, XML, etc.).

23. Cross-Origin Resource Sharing

Selon le standard de l'OGC (Open Geospatial Consortium , 2014) et en se basant sur le manuel de Geoserver (Open Source Geospatial Foundation , 2017e), *"un serveur WFS proposant un service WFS simple doit implémenter les opérations suivantes :*

- ***GetCapabilities :*** *pour récupérer les opérations et services disponibles sur le serveur ;*
- ***DescribeFeatureType :*** *demande des informations sur un type de fonctionnalité particulier avant de demander les données réelles (...) ;*
- ***ListStoredQueries :*** *renvoie une liste des requêtes stockées, actuellement conservées par le serveur WFS ;*
- ***DescribeStoredQueries :*** *renvoie des métadonnées détaillées sur chaque requête stockée maintenue par le serveur WFS. Une description d'une requête individuelle peut être demandée en fournissant l'ID de la requête spécifique.*
- ***GetFeature :*** *renvoie une sélection de fonctionnalités à partir de la source de données."*

Il doit également supporter au moins une méthode de requête parmis HTTP GET, HTTP POST et SOAP.

C'est principalement la fonctionnalité GetFeature qui a été utilisée pour la récupération de données météorologiques sur base d'une requête spatiale : les données récupérées figurent dans la "bounding box" de la fenêtre d'affichage du client. Dans ce cas, on peut parler d'un **service WFS en lecture seule**, car aucune édition des données n'est effectuée.

2. **La réponse** est récupérée au format texte (sauf pour le format shapefile qui est alors compressé) par le client, selon une syntaxe spécifiée lors de la requête (Open Source Geospatial Foundation , 2017d). Les formats disponibles sont :
   - **GML2 :** le "Geography Markup Language" dans sa version 2 est la valeur par défaut du WFS 1.0.0 ;
   - **GML3 :** valeur par défaut du 1.1.0 et 2.0.0 ;
   - **Shapefile :** archive ZIP contenant le shp ;
   - **JSON :** "JavaScript Object Notation" ou GeoJSON, si celui-ci contient des données géospatiales ;
   - **JSONP :** JSON avec Padding, une fonction de retour y est indiquée ;
   - **CSV :** "Comma-Separated Values".

   En conséquence de l'environnement du côté client, qui est essentiellement élaboré avec du JavaScript (Section 6), le format GeoJSON a été privilégié. En effet, la bibliothèque JavaScript Leaflet prend ce format en charge nativement.

Rôle de Geoserver

Au sein de l'architecture déployée dans ce cadre (Figure 5), le rôle de Geoserver est de **fournir**, par le biais du protocole WFS, **les données METAR** (ainsi que **les polygones associés** (Section 4)) à un client de cartographie construit sur des services web et accessible par un navigateur. Ces données, téléchargeables dynamiquement sans interrompre la navigation grâce au mécanisme AJAX, sont donc affichables dans une API de cartographie à la demande de l'utilisateur. Elles y sont superposées à des fonds de cartes (Section 5), ainsi qu'à des données de préparation de vol entrées par l'utilisateur (Section 6.2).

3.4.3 Mise à jour automatique des données METAR

Les données METAR évoluant toutes les 30 minutes (Section 2.4), il est nécessaire de mettre en place un système de mise à jour (si possible automatique) de celles-ci. Cette tâche peut être programmée à l'aide d'un "script shell" qui contient une ou plusieurs commandes qui sont exécutées de manière séquentielle dans l'interpréteur de commande. L'exécution de ce script peut être programmée de manière systématique en respectant un certain timing : une exécution par 30 minutes au minimum. A cet effet, le système d'exploitation Ubuntu fournit un gestionnaire des tâches planifiées : Cron (Communauté francophone d'utilisateurs d'Ubuntu, 2017a). Ce script contiendrait les instructions suivantes :

1. Téléchargement des données,
2. Transformation dans le format SpatiaLite,
3. Création des zones d'influence des stations météo (Section 4.3),
4. Import des METAR et des zones d'influences dans la base de données (Section 3.2.4).

Ce système de mise à jour directement dans la base de données permettrait à GeoServer de prendre en compte les nouvelles données dès leur insertion, étant donné que l'on conserve le même schéma.

Toutefois, dans le cas présent **ce mécanisme n'a pas été implémenté** car un seul fichier METAR (fourni par ESNAH à titre d'échantillon) a été exploité.

4 Traitement des données météorologiques

L'interface de préparation de vol, ayant pour objectif de fournir des données météorologiques au pilote, comporte un module de traitement des données METAR. Par ailleurs, une zone d'influence, regroupant tous les points les plus proches de chacune des stations, leur est affectée à chacune d'entre-elles sur base de polygones de Voronoï (Section 4.2).

4.1 Les données METAR

4.1.1 Données METAR et architectures orientées web

Dans le contexte de l'affichage de données géographiques en "temps réels" [24], les METAR vont souvent de paire avec les architectures orientées web. En effet, la diffusion de celles-ci (qui sont, rappelons-le, au format texte) est facilitée par les protocoles, tels que HTTP, en vigueur dans le monde du web.

Ainsi, les exploitations de ces données au sein de plateformes web à caractère géographique sont multiples. On peut citer la plateforme géospatiale multi-source GEMSS (Geospatial Emergency Management Support System) déployée afin de fournir un *système de surveillance pour les populations vulnérables quant aux dangers liés au climat* (Houghton et al., 2011). Au sein de cette application, diverses sources de données sont exploitées en temps réel. Les données METAR y jouent un rôle de grande importance, d'un point de vue météorologique.

Par ailleurs, les données METAR peuvent jouer un rôle clé lors de *simulations de feux de forêts* (Bogdos & Manolakos, 2013). En effet, celles-ci sont directement exploitées au sein du programme FLogA (Fire Logic Animation) afin de fournir la direction du vent et sa vitesse lors de scénarios d'incendies. Ce programme permet de simuler le comportement d'un feu, à partir de données géographiques réelles ou modifiées. Il joue ainsi des scénarios selon certaines modifications des paramètres définissant le contexte : type de forêt, direction et vitesse du vent, autre emplacement géographique selon les mêmes conditions météorologiques, etc.

Ensuite, les données METAR des stations peuvent être utilisées au sein d'une application de cartographie, afin de compléter *une étude sur un phénomène météorologique* tel qu'un cyclone (Conte et al., 2011). Dans ce cas, elles viennent renforcer les arguments permettant d'appréhender le phénomène, en s'ajoutant aux autres sources de données récoltées par la plateforme, et en proposant un aperçu des conditions météorologiques régnant sur le territoire couvert par les stations.

De plus, des opérations de "*data mining*" sont également réalisées, en vue d'intégrer de grands jeux de données météorologiques au sein d'une plateforme, afin de *prédire la formation de bancs de brouillard ou de couverture nuageuse* (Bartok et al., 2010). Ces prédictions permettent, dans un contexte similaire à celui-ci, de mieux appréhender les problèmes de gestion du traffic aérien mais leur implémentation nécessiterait une architecture serveur plus complexe. Celle-ci serait capable de prendre en charge la modélisation de données météo et non une simple visualisation.

Enfin, la centralisation automatique de différentes données météorologiques, dont les données METAR, existe déjà dans le cadre de simulations et d'analyses de systèmes biologiques (Yang et al., 2010). En effet, celles-ci sont récoltées, traitées et stockées au sein d'un *système intégré de gestion et d'information agricole*.

Finalement, le *téléchargement des données METAR au format texte* est déjà implémenté dans des applications embarquées sur des appareils mobiles (Turiak et al., 2014). Ainsi, ces applications permettent à l'utilisateur de se procurer les METAR des stations qui l'intéressent et ce, même durant le vol, s'il bénéficie d'un réseau 3G/4G.

En conclusion, les données METAR sont exploitées au sein de domaines variés mais la représentation de leur zones d'influence, sous forme de polygones de Voronoï, n'est pas exploitée au sein de la littérature.

24. Bien que, dans le cadre de ce travail, un seul fichier de METAR a été exploité, on peut se placer dans un contexte d'application en temps quasi-réel avec des METAR mis à jour toutes les 30 minutes.

4.1.2 Source de données

Les données METAR récoltées sont celles diffusées librement par la NOAA (National Oceanic and Atmospheric Administration). En effet, ces données du domaine publique américain sont sous le régime de la loi sur la liberté d'information (National Oceanic and Atmospheric Administration, 2017).

Bien qu'une interface interactive soit disponible pour la pré-visualisation de ces données (NOAA National Weather Service, 2017), c'est par le biais d'une requête de type GET sur les serveurs d'aviationweather.gov et selon le protocole HTTP que les données sont obtenues en format XML. Les différents paramètres sont introduits dans l'URL et séparés par le caractère "&". En guise d'exemple, une requête renvoyant les données METAR d'une seule station, celle de Liège (EBLG) :

Exemple de requête : metar de l'aéroport de Liège

```
http://aviationweather.gov/adds/dataserver_current/httpparam?
        datasource=metars&
        requestType=retrieve&
        format=xml&
        mostRecentForEachStation=true&
        hoursBeforeNow=48&
        stationString=EBLG
```

Détail des paramètres utilisés

- **"datasource=metars"** précisant le type de données choisi parmis les suivants : metars, tafs, aircraft reports et airsigmets ;
- **"requestType=retrieve"** spécifiant le type de requête ;
- **"format=xml"** pour le format récupéré qui peut être XML, CSV ou gzip ;
- **"mostRecentForEachStation=true"** afin de n'obtenir que les METAR les plus récents pour chacune des stations ;
- **"hoursBeforeNow=48"** afin de ne pas obtenir des données trop anciennes et donc obsolètes ;
- **"stationString=EBLG"** précisant le code ICAO de la station météorologique de Liège.

Résultat

Le texte, selon la syntaxe XML, récupéré après la requête est le suivant :

```
<response xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        version="1.2" xsi:noNamespaceSchemaLocation=
                "http://aviationweather.gov/adds/schema/metar1_2.xsd">
        <request_index>194838317</request_index>
        <data_source name="metars"/>
        <request type="retrieve"/>
        <errors/>
        <warnings/>
        <time_taken_ms>37</time_taken_ms>
        <data num_results="1">
                <METAR>
                        <raw_text>
                        EBLG 311350Z 30004KT 220V360 9999 FEW044 22/11 Q1022 NOSIG
                        </raw_text>
                        <station_id>EBLG</station_id>
                        <observation_time>2017-05-31T13:50:00Z</observation_time>
                        <latitude>50.63</latitude>
                        <longitude>5.45</longitude>
                        <temp_c>22.0</temp_c>
                        <dewpoint_c>11.0</dewpoint_c>
```

```
                        <wind_dir_degrees>300</wind_dir_degrees>
                        <wind_speed_kt>4</wind_speed_kt>
                        <visibility_statute_mi>6.21</visibility_statute_mi>
                        <altim_in_hg>30.177166</altim_in_hg>
                        <quality_control_flags>
                        <no_signal>TRUE</no_signal>
                        </quality_control_flags>
                        <sky_condition sky_cover="FEW" cloud_base_ft_agl="4400"/>
                        <flight_category>VFR</flight_category>
                        <metar_type>METAR</metar_type>
                        <elevation_m>178.0</elevation_m>
                </METAR>
        </data>
</response>
```

On y observe la version texte (raw text), telle que diffusée par les stations, mais également les différents champs entre balises : identifiant ICAO, date de l'observation, latitude et longitude, température (°C), etc. Parmi ceux-ci, figure notamment la catégorie de vol autorisée (VFR, MVFR, IFR, LIFR), qui sera exploitée par la suite. En effet, ce paramètre permet d'établir une classification au sein des zones d'influence des stations construites à partir des polygones de Voronoï (Section 4.2).

Parsing des données XML

Ensuite, les données fournies (jusqu'alors en XML) doivent être analysées : on parle de "**parsing**" du XML. En effet, ces données sont hiérarchisées selon un arbre par le biais des balises. Chaque noeud au sein de cet arbre apporte une information. La Figure 12 montre un aperçu de l'organisation en arbre (sur deux niveaux) du fichier XML de la requête précédente. Cette visualisation est effectuée à l'aide du logiciel "Oxygen XML Editor".

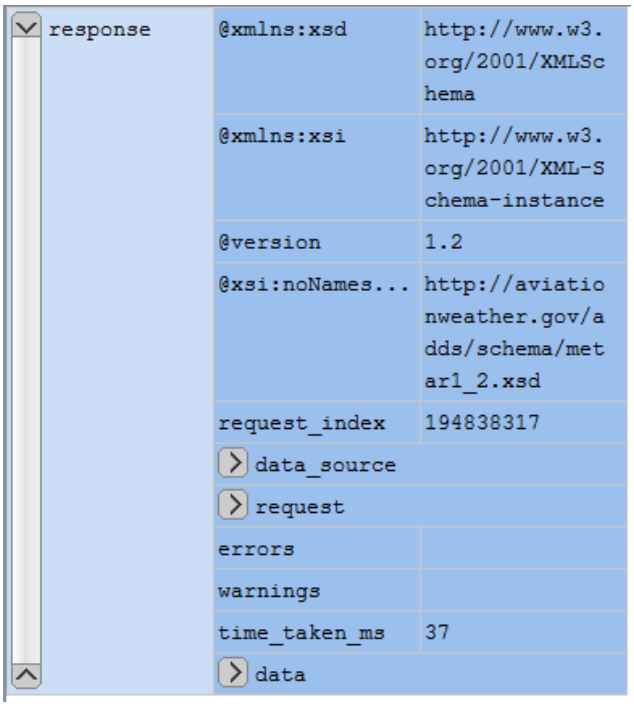

FIGURE 12 – Vue en arbre du fichier XML "response" obtenu par requête HTTP.

Dès lors, l'opération de parsing va permettre d'organiser les données en tuples au sein d'une base : les balises correspondant à l'attribut et sa valeur (pour le tuple évalué) étant comprise entre celles-ci. Ainsi, si l'on se réfère à l'exemple pré-cité, un tuple sera généré pour l'aéroport de Liège. Les valeurs attributaires qui lui seront affectées sont les suivantes :

Attribut	**Valeur**	**Unité**
Identifiant ICAO	EBLG	
Temps d'observation	2017-05-31T13 :50 :00Z	
Latitude	50.63	°
Longitude	5.45	°
Température	22.0	° C
Point de rosée	11.0	° C
Direction du vent	300	°
Vitesse du vent	4	Nœuds
Visibilité	6.21	Milles terrestres
Pression barométrique	30.177.166	Centièmes de pouce de mercure
Couverture nuageuse	Few	
Hauteur de la base des nuages	4400	Pieds
Catégorie de vol	VFR	
Altitude	178	m

TABLE 4 – Données contenue dans un METAR : exemple de l'aéroport de Liège

Cette opération de parsing est exécutée directement au sein des serveurs d'ESNAH, après la requête des données METAR. Son implémentation, en python, utilise l'API "ElementTree" (lxml) qui contient des méthodes de parsing. Son fonctionnement, en pseudo-code, est le suivant :

```
Import des librairies ogr , os, glob, sqlite3, re, math
Import de la classe etree (librairie lxml)

DEBUT
                // CREATION DU FICHIER SQLITE
        SI ("metar.sqlite" existe) alors
                Connexion a la base de donnees de "metar.sqlite"
                Suppression des donnees anterieures
        SINON
                Creation du fichier "metar.sqlite" et de sa base de donnees
                Connexion a la base de donnees
        FINSI

        Creation de la table "metar"
        Ajout de la geometrie a la table "metar"

                // PARSING DES DONNEES XML
        Creation d'un objet etree et affectation du contenu du fichier xml
                par parsing (methode parse)

                // IMPORT DES ATTRIBUTS DANS LE SPATIALITE
        POUR CHAQUE ID_ICAO au sein de l'etree
                POUR CHAQUE station meteo
                        Attribut_SQLite = Attribut_etree
                FINPOUR
        FINPOUR
FIN

Sauvegarde de "metar.sqlite"
```

Ainsi, les données METAR sont importées dans une base de donnée SQLite, elle-même au sein d'un seul fichier, bénéficiant de l'extension spatiale Spatialite (The Gaia-SINS federated project, s.d.). Le SQLite est une bibliothèque libre compacte, proposant un moteur de base de données relationnelle interrogeable en SQL, mais sans dépendre du schéma client-serveur : un seul fichier contient la base de données et le moteur SQL est directement intégré dans une application (SQLite, 2017). Cette particularité en fait un moteur de base de données multi-plateformes facilitant l'**échange de données** et propice aux développements sur appareils mobiles.

Notons que, dans le contexte de ce prototype qui n'implémente jusqu'à présent qu'un simple client web, il serait plus direct d'importer les données dans une base PostGIS sans passer par le format SQLite. En effet, une connexion est possible si l'on utilise l'API "psycopg" qui contient les drivers nécessaires à une connexion Python/PostgreSQL. Toutefois, le format SQLite a été privilégié au sein d'ESNAH afin d'obtenir une base de données exploitable côté client au sein d'une application sur appareil mobile et ce, même hors connexion. Les données METAR, ainsi que les polygones de Voronoï associés (Section 4.2), seraient donc téléchargeables en SQLite et **directement exploités au sein de l'application**, limitant les échanges client-serveur durant la navigation. Cette approche a donc été conservée, afin d'anticiper les besoins d'une implémentation de ce prototype sur appareil mobile ou en complément de ce dernier.

Par conséquent, cela implique une opération supplémentaire dans le cadre de ce prototype : l'import du fichier SQLite au sein de la base de données PostGIS. Ce transfert a été réalisé à l'aide de la fonction "ogr2ogr" de GDAL, comme détaillé dans la Section 3.2.4, et peut également être automatisé du côté serveur par le biais d'un gestionnaire des tâches planifiées tel que Cron.

Requête pour tous les aéroports du réseau

Concrètement, la requête récupérant les METAR pour l'ensemble du réseau ICAO doit être ré-exécutée toutes les 30 minutes et est la suivante :

```
http://aviationweather.gov/adds/dataserver_current/httpparam?
        datasource=metars&
        requestType=retrieve&
        format=xml&
        mostRecentForEachStation=true&
        hoursBeforeNow=48
```

4.2 Polygones de Voronoï

Les diagrammes de Voronoï permettent de diviser géométriquement l'espace 2D en zones d'influence autours d'entités ponctuelles (Boots, 1986). Ces diagrammes sont un partitionnement du territoire réalisé en attribuant une zone à chaque point d'un échantillon. Chaque zone, centrée sur une de ces entités ponctuelles, rassemble les points de l'espace les plus proches du centre.

Ces polygones peuvent porter des appellations différentes selon le contexte dans lequel ils sont exploités : diagrammes de Voronoï, tessellation de Thiessen ou encore polygones de Dirichlet. Toutefois, le même principe est appliqué et ils représentent l'**ensemble des points les plus proche du centre** de leur polygone que d'aucun autre centre.

En mode vectoriel (comme dans le cas présent), leur définition s'effectue géométriquement en construisant un réseau de triangles dont les entités ponctuelles sont les sommets. Ce réseau constitue alors une **triangulation de Delaunay** (Figure 13 (gauche)). Ensuite, on génère les médiatrices des côtés des triangles du réseau. Ces médiatrices définissent alors les côtés des **polygones de Voronoï**. Ceux-ci constituent le graphe dual à la triangulation de Delaunay (Figure 13 (droite)).

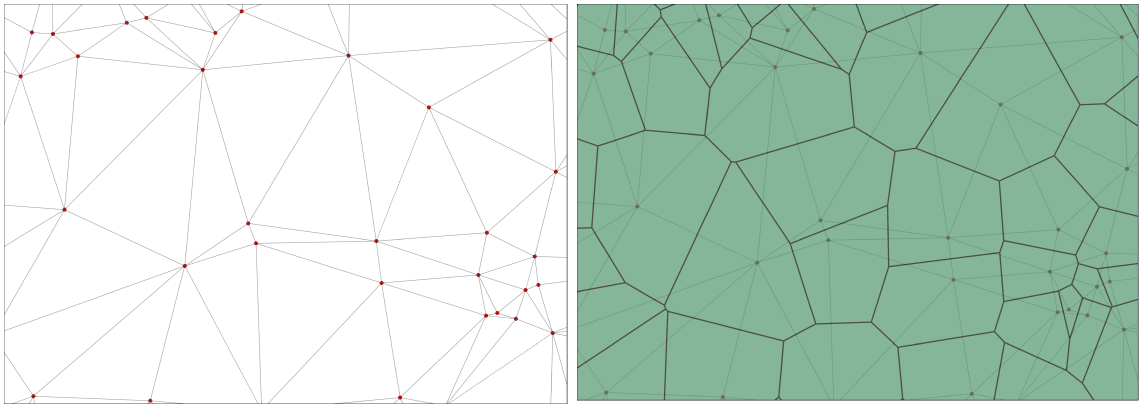

FIGURE 13 – Réseau de triangles de Delaunay (gauche) et réseau de polygones de Voronoï avec tringles de Delaunay en gris clair (droite).

Dans notre contexte, ces centres correspondent à des stations météorologiques émettrices. Les diagrammes de Voronoï en sont les zones d'influence. Lors de l'interprétation de ces diagrammes, on émet l'hypothèse que cet ensemble de points au sein de la zone, est soumis à des conditions météorologiques relativement similaires à celles décrites par la station émettrice, celle-ci étant la plus proche pour tous les points au sein de son polygone. Notons que cette hypothèse va à l'encontre de l'applicabilité initiale des données METAR, fixée à 3km (Section 2.4.2).

Concrètement, ce sont les règles de vol en vigueur pour chacune des données METAR qui sont exploitées afin de réaliser une classification, et proposer au pilote un aperçu des règles de vols en vigueur (Figure 14).

Dès lors, le pilote est à même de réaliser sa trajectoire selon le type de plan de vol qu'il envisage (VFR ou IFR) lors de sa préparation. Par exemple, s'il projette un vol VFR, il lui faudra éviter les zones ou la règle de vol est IFR (ou LIFR). Un autre exemple serait celui d'un pilote planifiant un vol IFR, mais souhaitant toutefois en effectuer une partie en conditions de vol VMC car celles-ci sont plus clémentes.

Les conditions de vols autorisées sont fonctions des conditions météorologiques, et peuvent prendre les valeurs suivantes au sein d'un METAR :

- **VFR :** vol à vue,
- **MVFR :**[25] vol à vue selon les conditions minimales avec conditons moins bonnes que VFR (plafond de 1000 à 3000 pieds et / ou 3 à 5 milles de visibilité qui correspondent respectivement à 5,56 km et 9,26 km),
- **IFR :** vol aux instruments,
- **LIFR :**[26] vol aux instruments avec conditions moins bonnes qu'IFR.

Par la suite, ce sont les polygones sur la trajectoire planifiée qui seront extraits (Section 6.2.4) et affichés.

On peut envisager d'intégrer les données TAF selon la même approche au sein de la plateforme. En effet, celles-ci ont une structure similaire aux METAR et permettraient au pilote de planifier un vol selon des conditions futures de vol basées sur des prévisions.

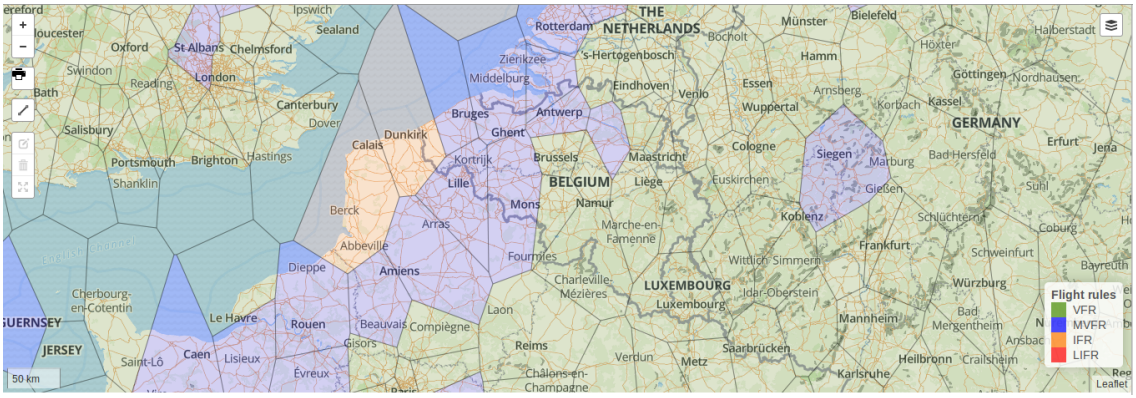

FIGURE 14 – Affichage des polygones de Voronoï associés aux données METAR.

4.3 Traitements

Bien que la création de polygones de Voronoï soit possible au sein d'une interface d'un SIG-logiciel proposant des fonctions d'analyse spatiale, il est nécessaire d'automatiser ces opérations au sein d'un script. Le langage python a été choisi pour remplir ce rôle car il permet un accès à différents outils d'analyse spatiale, tels que GDAL/OGR, et de connexions aux différents SGBD, dont PostgreSQL. Le script réalisé serait, par la suite, exécuté de manière systématique du côté serveur, afin de traiter les données METAR lors de chaque mise à jour (Section 3.4.3). Les opérations d'extraction de données à partir d'un fichier SpatiaLite/SQLite de données METAR, dans le but de générer les géométries des polygones de Voronoï sont les suivantes :

1. **Accès au fichier SQLite** : connexion à la base de données ;
2. **Chargement** dynamique **de Spatialite**, en tant que module d'extension du SQLite. On peut ainsi bénéficier des fonctions spatiales implémentées dans SpatiaLite (The Gaia-SINS federated project, 2017). Ce chargement s'effectue en exécutant la requête SQL suivante :

```
SELECT load_extension('mod_spatialite')
```

25. Marginal Visual Flight Rules ou Minimal Visual Flight Rules
26. Low Instrument Flight Rules

3. **Création** d'un objet "**curseur**". Il s'agit d'une structure de contrôle qui permet de parcourir les tuples de la base de données ;
4. **Création** de la **table destinée aux polygones** de Voronoï ;
5. **Création des polygones de Voronoï** à partir des emplacements des stations METAR, en recourant à la fonction "ST_VoronojDiagram" de SpatiaLite ;
6. **Récupérations des variables exploitées** au sein de chacune des stations : identifiant ICAO, règle de vol, latitude et longitude ;
7. **Création et remplissage des tables** en conséquence des variables récupérées.

L'implémentation en python de ces opérations figure dans l'Annexe G.

Notons que, à nouveau, un fichier SQLite est généré puis, pour la réalisation de ce prototype, importé dans une base de données PostGIS. Or, il est plus direct d'insérer directement ces données dans la base depuis le script python, comme mentionné précédemment (Section 4.1.2).

5 Planimétrie et orographie

5.1 Données OpenStreetMap

5.1.1 Source des données OSM : la géographie participative

Les données de la plateforme de géographie participative (VGI[27]) OpenStreetMap sont obtenues à partir des contributions de volontaires à travers le monde (OSM Foundation, 2017; OpenStreetMap, 2017). Ces données sont encodées sur base de "tags" qui permettent de les **classifier** selon un **processus d'indexation personnelle**, parfois appelé "folksonomie". Ce système est donc basé sur une classification collaborative spontanée mais décentralisée.

Ainsi, ce qui suscite l'intérêt de ces données présente également des inconvénients : l'indexation est effectuée par des non spécialistes. Ce qui compromet la mise en commun, le partage et la réutilisation de ces données. Par conséquent, des mécanismes de contrôle de la qualité doivent être élaborés (Comber et al., 2013). De plus, les tags attribués aux données dépendent de l'interprétation du volontaire lors du lever. Cela peut créer certaines ambiguïtés. Par conséquent, certaines améliorations du modèle de données OSM sont envisagées à l'avenir, afin de palier aux manquements du principe "clé-valeur" des tags, sur base d'ontologie, tout en conservant l'aspect sémantique des données OSM (Hombiat et al., 2015).

Dans la pratique, les volontaires attribuent des tags aux données levées en se basant sur un wiki, représentant un **guide de bonnes pratiques** (OpenStreetMap Wiki, 2017c). Celui-ci contient bon nombre d'exemples, afin d'aider l'utilisateur à choisir les tags de manière appropriée et sans ambiguïté.

Malgré ce petit inconvénient, les données OSM permettent une **cartographie du monde entier** qui est à la fois *libre d'utilisation* et librement *modifiable*. C'est pour cette raison que ces données ont été privilégiées lors de la réalisation de ce mémoire.

5.1.2 L'utilisation des données OSM et la license ODbL

La licence ODbL1.0 permet aux utilisateurs d'accéder aux données OSM gratuitement et selon l'usage qu'ils souhaitent. En effet, ces données ne comportent ni droits d'auteurs, ni licence. Il convient toutefois d'indiquer l'origine des données (par exemple dans le cartouche d'une carte) et de rendre publique toute amélioration et/ou modification de celles-ci (OpenStreetMap Foundation, 2012).

5.1.3 Modèle conceptuel des données OSM

Les données OSM sont structurées selon un modèle conceptuel de données du monde physique. Celui-ci s'avère plus complexe qu'un simple fichier contenant des géométries et, éventuellement, leurs attributs. En effet, la structure des données OSM offre un aspect sémantique par le biais d'une catégorisation par étiquetage ("tagging"), liant les différentes données entre-elles.

Dans un premier temps, elles sont stockées selon trois types primitifs d'objets :
— **Noeuds** ("nodes") symbolisant des éléments ponctuels de l'espace ;
— **Chemins** ("ways") symbolisant des éléments linéaires, ainsi que des délimitations d'éléments polygonaux ;
— **Relations** ("relations") définissant comment les éléments sont liés entre-eux.

Ensuite, ces primitives de données servent à représenter les entités sémantiques suivantes. Parmi celles-ci, les trois primitives graphiques vectorielles permettant de modéliser la géométrie des entités géographiques :
— **Points** (primitive vectorielle),
— **Lignes** (primitive vectorielle),
— **Polygones** (primitive vectorielle),
— **Relations**.

27. en anglais "Volunteered Geographic Information"

C'est pourquoi, lors de l'ouverture d'un fichier de données OSM dans un logiciel SIG tel que QGIS, on obtient les couches suivantes :

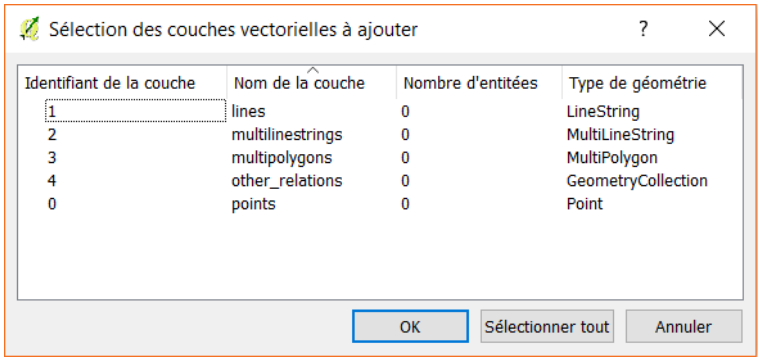

FIGURE 15 – Types de couches vectorielles OSM

A chaque objet, peuvent être affectées des données attributaires sous forme de "tags", afin de le compléter. Ces tags, définis lors du lever par les volontaires de la communauté OpenStreetMap, sont organisés selon un principe "clé-valeur" et doivent respecter certaines conventions de bon usage. Des exemples figurent notamment dans la Table 5.

Clé	**Valeur**	**Description**
Highway	Residential	*Voie résidentielle*
Maxspeed	50	*Vitesse maximale : 50 km/h*
Amenity	University	*Université*
Aeroway	aerodrome	*Aéroport*
Aeroway	terminal	*Bâtiment d'aéroport destiné à l'accueil des passagers*

TABLE 5 – Principe clé-valeur des tags des données OpenStreetMap

5.1.4 Classification des données OSM

Ce principe de clé-valeur présente un grand intérêt pour effectuer une **classification des données OSM** dans un but précis. Ainsi, une classification spécifique a été élaborée avec ESNAH selon les critères de pertinence d'un point de vue aéronautique (voir Annexe E). Ainsi, il est possible d'épurer les fonds de cartes, afin de ne pas encombrer l'affichage destiné au pilote avec des données superflues. Dans cette classification, figurent :

— Des **repères visuels** tels que des églises, chemins de fer, prison, stade, rivières... ;
— Des **sources de danger** potentiel, comme des éoliennes, ou encore des pilonnes électriques (à éviter en cas d'atterrissage d'urgence ou lors de vol VFR à basse altitude).

Notons que certains éléments peuvent, selon le contexte, figurer dans les deux catégories.

L'opération de sélection des données basées sur cette classification à partir d'étiquettes (tags) est détaillée dans la Section 5.2.

5.1.5 Etat de l'art de l'utilisation de données OSM pour des applications de navigation

A l'heure actuelle, les applications de navigation exploitant les données OpenStreetMap sont nombreuses et selon des perspectives diverses. En effet, depuis l'arrêt de la disponibilité sélective des données de positionnement par GPS et du succès des appareils mobiles, l'**aide à la navigation par le biais d'applications**

rencontre un certain succès. Ces applications mobiles de cartographie multi-usages intègrent généralement des opérations de routage GPS, de guidage visuel et/ou vocal (OsmAnd, 2017; Navmii, 2017). Elles ciblent généralement les usagers de la route mais proposent également leurs services aux piétons. Certaines d'entre-elles n'hésitent pas à se **spécialiser pour des types d'utilisateurs** tels que les cyclistes (Bikecitizens, 2017). Plusieurs de ces applications offrant même des interfaces web pour préparer la navigation et intégrer des données définies par l'utilisateur au sein de l'application par la suite (OsmAnd, 2017; Outdooractive GmbH, 2017).

Par ailleurs, les données OSM sont également présentes dans le contexte de la recherche. En effet, bien que leurs perspectives ne sont pas limitées à la navigation (Neis & Zielstra, 2014) , elles interviennent de manière récurrente au sein d'applications de navigation associées à des domaines multiples et, parmi eux, on peut citer la navigation pédestre (Tomàs Castellà, 2012), la navigation à l'intérieur de bâtiments (Czogalla, 2015; Goetz, 2012), la navigation autonome de robots (Fleischmann et al., 2016) ou encore la navigation d'urgence en cas de feu de forêt (Wang et al., 2014). Toutefois, la piste d'une application de service web de préparation de vol n'a pas été explorée au sein de la littérature scientifique.

5.2 Sélection des données OpenStreetMap selon une classification

Comme mentionné précédemment (Section 5.1.3 et Section 5.1.4), les données OSM sont organisées selon des étiquettes (tags). C'est sur base de celles-ci qu'est défini le style associé au rendu effectué par Mapnik. Ces informations stylistiques sont regroupées au sein d'un fichier XML, le "Map Definition File", qui comporte notamment les éléments suivants (Eastcott, s. d.) :

- **Map** qui est l'élément racine définissant le rendu associé à l'élaboration de la carte. Ainsi, l'objet "Map" est un contenant de tous les autres éléments et ceux-ci héritent également de ses propriétés ;
- **Datasource Element** qui contient la définition d'une source de données géographiques à exploiter. Ces données peuvent être de type ESRI Shapefile, PostGIS, raster supporté par GDAL, OSM (.osm.pbf), etc. Ces sources de données sont généralement définies directement dans un élément "layer". Dans notre cas, les données OSM sont récupérées sur base d'une requête vers la base de données PostGIS. Cette requête est incluse à l'objet "datasource", en tant que paramètre ;
- **Filesource Element** qui précise un chemin d'accès à un (ensemble de) fichier(s). Il associe, par ailleurs, un nom à ces fichiers. Ce nom permet de référencer les éléments selon une même appellation ;
- **FontSet Element** qui permet d'ajouter une liste de polices non incluses dans l'application. Ainsi, en cas de manquement lors de l'implémentation de polices au sein du rendu, c'est cet élément qui est consulté ;
- **Layer Element** qui précise le rendu associé à une source (Datasource) selon un certain rendu (Style). Cet élément effectue la jonction entre ces deux autres éléments. Lors de l'exécution, les éléments "layer element" sont consultés dans l'ordre dans lequel ils figurent au sein du fichier XML ;
- **Rule Element** qui sert à définir d'autres éléments intervenant selon des "règles" spécifiques. Il est possible de définir des filtres réalisant la sélection des données, la symbolisation de certaines types (ligne, points, étiquettes, polygones, raster, etc.) ou encore de règlementer les différentes échelles lors du rendu. En effet, les données ne seront pas représentées de la même manière selon l'échelle à laquelle on travaille ;
- **Style Element** qui définit le style selon lequel une source de données (Datasource) est rendue. Il est généralement composé de "règles" (Rule). Son utilisation est sollicitée depuis un élément "Layer".

La réalisation d'un style spécifique peut dès lors s'effectuer à partir d'un fichier de style pré-existant en ajustant les couches (layers), en éditant/supprimant certaines règles (rules) sur base des tags, etc.

C'est selon cette approche, soit la modification d'un fichier pré-existant, que le fichier de style de ce prototype a été réalisé. La réalisation d'un fichier de style totalement dédié au contexte de la navigation aérienne reste à faire, mais sort évidemment des limites de ce mémoire (plusieurs milliers de lignes de code à étudier dans le détail par des professionnels du domaine d'application). À titre d'exemple, les questions que doivent se poser les rédacteurs d'un tel travail sont du type :

"A partir de quelle échelle faut-il afficher les noms de ville ? Et ceux des rues ?",
"Quelle occupation du sol doit prévaloir sur une autre en cas de conflit ?",
"Comment obtenir tous les éléments d'une certaine thématique sachant que les volontaires ayant procédé au lever les ont potentiellement encodées différemment ?", etc.

5.3 Données SRTM et ASTER

5.3.1 Documentation

La mission SRTM, mise en œuvre par la NASA [28] et la NGA [29] en 2000, a permis de réaliser différentes versions de modèles numériques d'élévation (MNE) couvrant près de 80 % des terres émergées du globe selon une précision verticale de 16m (Farr et al., 2007; Jet Propulsion Laboratory (NASA), 2017). Cette mission offre une couverture des régions comprises entre 60 ° N et 56 ° S à des résolutions d' 1 sec d'arc, de 3 sec d'arc ou de 30 sec d'arc, selon les disponibilités et les besoins.

La Figure 16 montre le nombre de passage effectués par la navette de la mission SRTM (selon la bande C) au-dessus des zones du globe. On remarque les limites précédemment évoquées (60 ° N et 56 ° S), mais également la redondance de passage à certains endroits émergés (les zones vertes), ainsi que les passages multiples en zones côtières afin de contrer l'effet des vagues (voir Section 5.3.2).

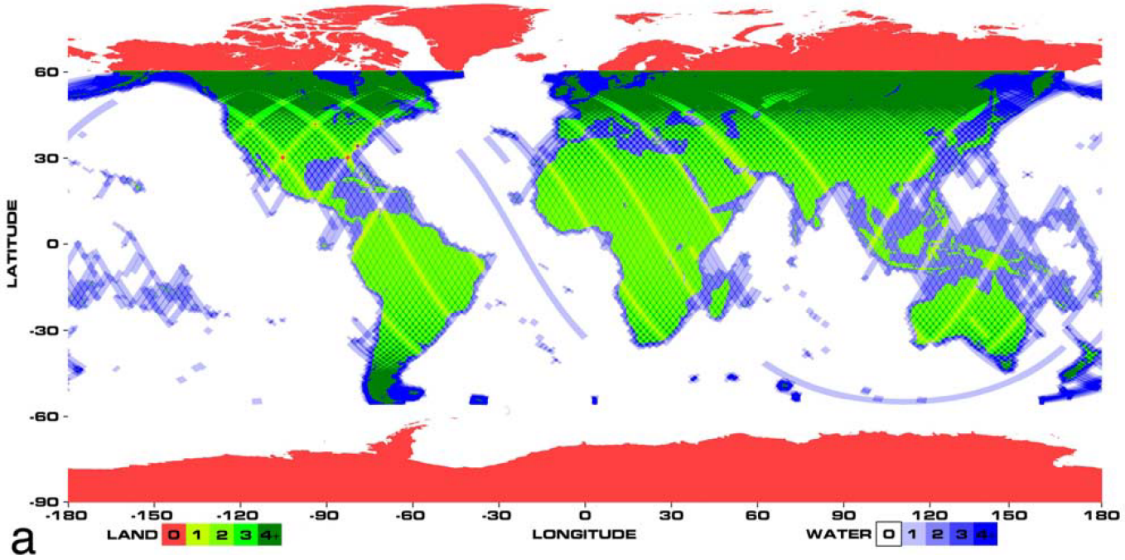

FIGURE 16 – Carte de couverture finale du SRTM (bande C) (Farr et al., 2007)

D'autres données de MNE global sont également disponibles librement : **les données ASTER** [30] produites par la NASA et le ministère japonais de l'économie, du commerce et de l'industrie. La résolution

28. National Aeronautics and Space Administration
29. National Geospatial-Intelligence Agency
30. Advanced Spaceborne Thermal Emission and Reflection Radiometer

proposée est similaire au SRTM1, c'est-à-dire 1 seconde d'arc. Leur précision verticale annoncée est de 20m (selon un intervalle de confiance à 95%) et de 30m horizontalement (selon un intervalle de confiance à 95%). Ensuite, la couverture de ces données va de 83 ° S à 83 ° N, englobant 99% de la masse terrestre (NASA Jet Propulsion Laboratory, s.d.).

5.3.2 Inconvénients de la méthode d'interférométrie radar

Suite au faible recouvrement de certains emplacements du globes (le satellite n'a pu effectuer qu'une seule mesure pour certains endroits en conséquence de son orbite), des artefacts sont présents dans le MNE résultant du SRTM sous différentes formes (Farr et al., 2007) :

— **Des zones de rivage mal interprétées** le long des côtes à cause des vagues perturbant la définition des lignes de rivage ;

— **Des zones de vides ("trous")** masquées au signal émis depuis la navette par le relief avoisinant.

Plusieurs passages selon des angles différents ont été nécessaires pour palier à ces manques. Toutefois, certaines zones restent affectées et nécessiteront des corrections.

Voici un exemple de données SRTM manquantes à proximité de la ville de Prague, d'après le travail de Bartoň (2010) :

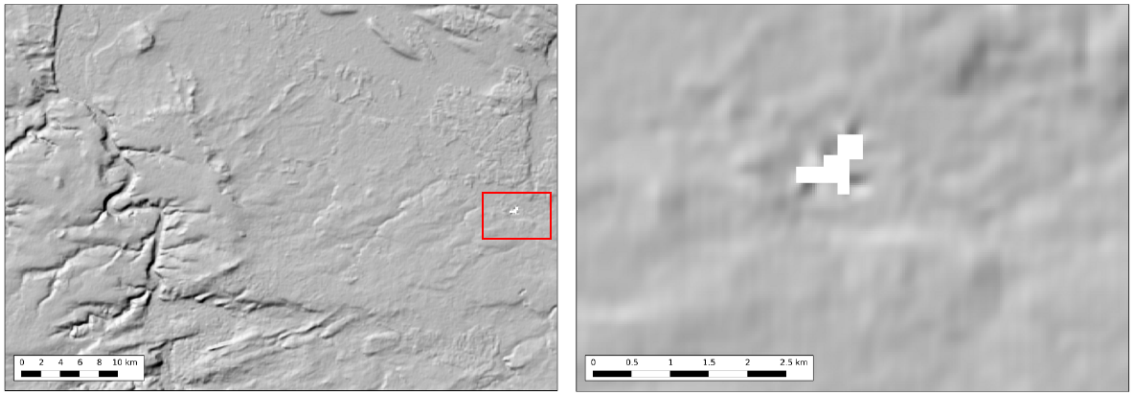

FIGURE 17 – Données manquantes du SRTM3 v1 à proximité de Pragues (Bartoň, 2010).

5.3.3 Disponibilité des MNE

Les données SRTM existent à différentes résolutions. Celles-ci sont disponibles sur les serveurs de l'US Geological Survey [31]. Les données ASTER, quant à elles, sont disponibles selon une seule résolution depuis les serveurs publics de la NASA [32].

Données	Résolution	Distance à l'équateur	Disponibilité (tout public)	Versions
SRTM1	1 sec d'arc	31 m	2015	v1, v2.1 (USA)
SRTM3	3 sec d'arc	93 m	2003	v1, v2.1
SRTM30	30 sec d'arc	926 m	2003	v1, v2.1
ASTER	1 sec d'arc	31 m	2009, 2011	v1, v2

TABLE 6 – Données raster SRTM et ASTER

Par ailleurs, les données SRTM de l'USGS sont exploitables sans restriction, à condition d'en citer la source (USGS, 2015), de même que les données ASTER (USGS, s.d.).

31. `https://dds.cr.usgs.gov/srtm/`
32. `https://reverb.echo.nasa.gov/reverb/`

5.3.4 Évolutions et versions

Les versions successives de **données SRTM** correspondent à l'évolution des corrections progressives de celles-ci. Les différentes versions officielles (de l'USGS) sont les suivantes :

- **Version 1 :** équivalent au MNE original non édité, tel qu'obtenu directement après la mission STS-99. Contient beaucoup d'artefacts dans les zones à faible rétro-diffusion, comme les masses d'eau ainsi que des zones manquantes (Figure 17) ;
- **Version 2 :** résultat d'une importante correction par la NGA. Les vides et autres aberrations de mesures ont été corrigés, de même que les côtes et les masses d'eau, à l'aide d'un masque de littoral. Il subsiste toutefois certaines zones manquantes ;
- **Version 2.1 :** recalcul du SRTM3 et correction de la version 2 par la NGA ;
- **Version 3 (ou SRTM plus) :** complétion des vides à partir du MNE des données ASTER GDEM2 de la NASA et du MNE GMTED2010 compilé par l'US Geological Survey. Cette version n'est pas accessible librement.

L'estompage de l'orographie réalisé lors de ce prototype exploite une version 4 distincte des versions originales de l'USGS. Il s'agit d'une version retravaillée de la version 2.1 (ou de la version 3 pour les USA) et qui est rendue disponible sur les dépôts du CGIAR CSI (CGIAR CSI, s.d.).

Les **données ASTER**, quant à elles, bénéficient de deux versions :

- **Version 1 :** correspondant aux données brutes telles que générées à l'aide d'images stéréoscopiques collectées par l'instrument ASTER à bord du satellite Terra. Toutefois, la version 1 de ce MNE correspond plutôt à une version expérimentale et ne présente pas encore de post-traitements en vue d'en corriger les manquements ;
- **Version 2 :** comprend plus de paires d'images stéréoscopiques, ce qui implique une meilleure couverture et une réduction des artefacts. Par ailleurs, l'algorithme de génération du MNE à partir des images stéréoscopiques a été amélioré pour cette version. Néanmoins, *"les données contiennent toujours des anomalies et artefacts qui entraveront l'efficacité de l'utilisation dans certaines applications"* (NASA Jet Propulsion Laboratory, s.d.).

5.3.5 Comparaison entre les données SRTM et les données ASTER

Selon Doumit (2013-2014) qui a procédé à une comparaison entre les données SRTM et les données ASTER v1 : *"le MNE global ASTER contient des anomalies et des artefacts qui réduiront sa convivialité pour certaines applications, car elles peuvent introduire de grandes erreurs d'élévation sur les échelles locales"*. De plus, selon lui, les données ASTER v1 ne constituent pas une réelle amélioration de la résolution quant au SRTM3 v2 (également disponible globalement), malgré un taux d'échantillonnage plus élevé : 31m pour les données ASTER et 93m pour les données SRTM3.

La Figure 18 illustre des artefacts, apparents sous forme d' "éclaboussures" Bartoň (2010), rencontrés lors de l'utilisation de données ASTER v1 à proximité de Prague.

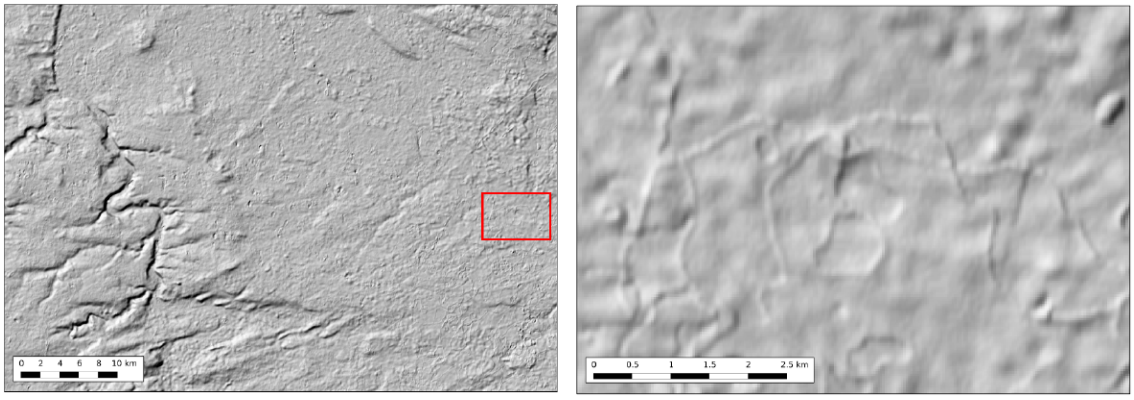

FIGURE 18 – Artefacts des données ASTER v1 à proximité de Prague (Bartoň, 2010).

L'évaluation des données ASTER v2 montre que la précision verticale selon ce MNE est encore inférieure à celle des MNE SRTM par endroit (Suwandana et al., 2012). Par conséquent, en l'absence de post-traitement plus avancé, les données ASTER v2, bien que prometteuses et complémentaires aux données SRTM, ne sont pas utilisées dans le cadre de ce prototype.

5.3.6 Données utilisées dans le cadre de ce travail

Lors de la réalisation de ce travail, c'est la **version 4 du SRTM3** (obtenue à partir de la version officielle 2.1 de l'USGS) qui a été implémentée, car celle-ci est la **version corrigée la plus précise** disponible pour le territoire belge. En effet, les données du SRTM1 ainsi que les données ASTER, bien que plus précises, ne comportent pas les post-traitements requis et présentent donc des artefacts qu'il faut corriger selon un travail d'interpolation. A défaut de nouvelle version corrigée et accessible librement, les données du SRTM1 ainsi que les données ASTER sont donc à éviter au sein d'une implémentation en vraie grandeur de ce prototype.

5.4 Estompage du SRTM

La prise en considération de l'orographie en arrière plan des tuiles OSM est réalisée en recourant à un estompage (hillshading) des données SRTM, en combinaison avec des paramètres de transparence des autres couches. Ainsi, il est nécessaire successivement d'obtenir les données SRTM, de réaliser le hillshading avec GDAL, puis de découper ce hillshading selon les frontières du territoire qui nous intéresse, à savoir la Belgique. Les frontières belges ont été téléchargées au format ESRI shapefile sur le site de l'atlas de Belgique [33].

5.4.1 Obtention des images SRTM concernées

Le territoire belge est situé à cheval sur 4 images SRTM (de 5 ° de côté) différentes :

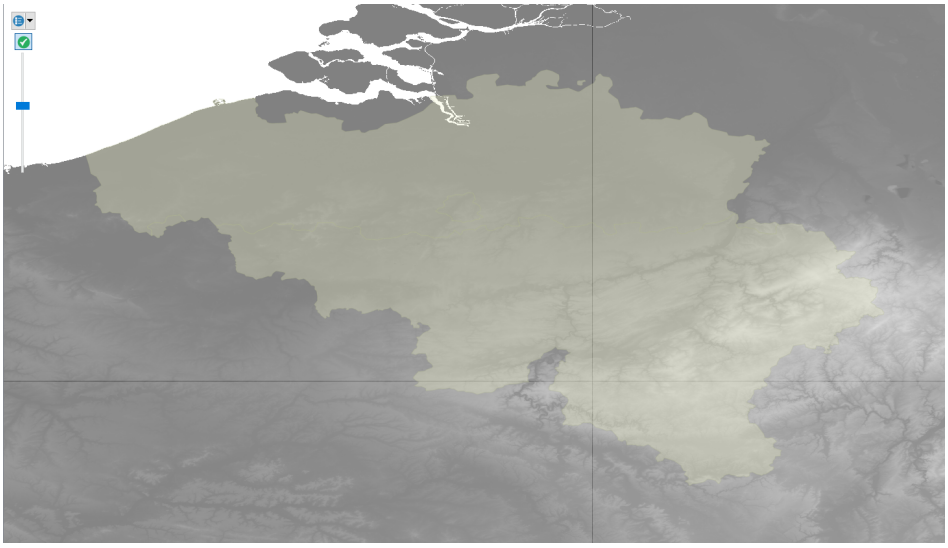

FIGURE 19 – Différentes images du SRTM couvrant le territoire belge

33. http://www.atlas-belgique.be/cms2/index.php

Fichier	φ_{min}	φ_{max}	λ_{min}	λ_{max}	Stockage	Dimensions
srtm_37_03	45 °	50 °	0 °	5 °	Entiers stockés sur 16 bits	6000*6000 pixels
srtm_38_03	45 °	50 °	5 °	10 °	Entiers stockés sur 16 bits	6000*6000 pixels
srtm_37_02	50 °	55 °	0 °	5 °	Entiers stockés sur 16 bits	6000*6000 pixels
srtm_38_02	50 °	55 °	5 °	10 °	Entiers stockés sur 16 bits	6000*6000 pixels

TABLE 7 – Propriétés des fichiers SRTM v4 requis pour le territoire belge

5.4.2 Realisation du hillshading avec GDAL

L'estompage de l'orographie combiné à des données OSM est une opération récurrente au sein des services de cartographie en ligne utilisant Mapnik (Section 5.5.1). Dès lors, des lignes de conduites sont présentes sur l' OpenStreetMap Wiki (2015) et l'opération de "hillshading" au sein de ce mémoire a été réalisée sur base de ces recommandations.

Fusion des images

Afin de ne travailler qu'avec un seul fichier GeoTIFF, la jonction des fichiers images compris dans une bounding box (entre 49.5 ° et 51.5 ° en latitude et 2.5 ° et 6.5 ° en longitude) est réalisée avec GDAL en ligne de commande, grâce à l'outil "gdal_merge.py".

```
gdal_merge.py -v -o srtm_belgium_merged -ul_lr
        2.5 51.5 6.5 49.5 srtm_belgium_merged.tif
```

Le paramètre -v permet d'afficher les détails des opérations dans la console au fur et à mesure. Le paramètre -o permet de définir le fichier en sortie et -ul_lr son étendue.

Vérification du SRS lié à l'image

Il peut arriver que le système de coordonnées ne soit pas spécifié pour le fichier. Dès lors, il faut spécifier que l'on travaille latitude/longitude WGS84 en utilisant gdal_translate :

```
gdal_translate -of GTiff -co "TILED=YES" -a_srs "+proj=latlong"
        srtm_belgium_merged.tif srtm_belgium_translate.tif
```

Le paramètre -of permet de définir le format destinataire (GeoTIFF), -co est une option sur le format en sortie (TILED précisant que ce fichier image sera exploité au sein d'une mosaïque de tuilage), -a_srs définit le système de coordonnées de la couche.

Reprojection en Mercator sphérique

La fonction gdalwarp est exploitée, afin de reprojeter le fichier raster selon le système de coordonnées "Web-Mercator", (GDAL, s.d.). Celle-ci permet de reprojeter une image en spécifiant, notamment, la taille des pixels dans le système de destination et en précisant l'ordre de l'interpolation réalisée.

```
gdalwarp -of GTiff -co "TILED=YES" -srcnodata 32767 -t_srs EPSG:3857 -order 3
        -tr 30 30 -multi srtm_belgium_translate.tif srtm_belgium_warped.tif
```

Les paramètres utilisés sont :

- **of GTiff** pour préciser le format de destination ;
- **co "TILED=YES"** pour passer l'option de tuile au résultat, celle-ci est recommandée pour les fichiers volumineux ;
- **srcnodata 32767** pour spécifier les valeurs de masquage. Les valeurs masquées ne sont pas utilisées lors de l'interpolation ;

- **t_srs EPSG :3857** pour spécifier le référentiel cible (web mercator) ;
- **order 3** ordre de l'interpolation ;
- **tr 30 30** résolution cible en x puis en y, selon le système de coordonnées de destination. En mètres pour le Pseudo-Mercator ;
- **multi** pour utiliser un warping "multithreading" qui réalisera plusieurs opérations simultanément.

Il est également possible d'augmenter la résolution finale afin d'éviter l'aliasing lors de la superposition avec les données OSM. Il s'agit donc de lisser le résultat sur base d'une interpolation, toujours par le biais de gdalwarp.

```
-tr 15 15 -wt Float32 -ot Float32 -wo SAMPLE_STEPS=100
```

- **tr 15 15** précise la résolution que l'on souhaite en mètre par pixel ;
- **wt Float32** définit le format de stockage des pixels de "travail" (utilisés dans la mémoire tampon) en "float 32". En utilisant ce type, on permet à gdal de travailler avec des décimales, sans devoir arrondir à un entier près, lors des traitements en conséquence du format initial (entiers stockées sur 16 bits) ;
- **ot Float32** définit le format en sortie en "float 32". A nouveau, le type "float 32" permet l'usage de décimales ;
- **wo** précise une option de la fonction warp. "SAMPLE_STEPS=100" permet de densifier la grille d'échantillonnage qui comporte, par défaut, 21 "étapes".

Hillshading

L'opération de hillshading fournit un ombrage du terrain, ce qui est particulièrement utile pour **visualiser les variations du relief**. Il s'agit d'une variation de la tonalité de la lumière (du clair au foncé) pour délimiter la forme du terrain. Ainsi, on observe le relief comme si une source de lumière éclairait celui-ci depuis un emplacement choisi (généralement en haut à gauche).

La première méthode analytique d'ombrage était juste fonction de l'angle (θ), mesuré en radian, formé par le rayon lumineux et la normale à la surface au point évalué (Wiechel, 1878) :

$$Ombrage = 255 * cos\theta$$

Par la suite, le hillshading analytique tel qu'implémenté dans les logiciels SIG actuels (Veronesi & Hurni, 2015), a évolué vers la formule suivante :

$$Ombrage = 255 * (cos(Z)cos(P)) + (sin(Z)sin(P)cos(Az - As))$$

- **P** est la pente du pixel ;
- **As** est l'aspect de la surface, c'est-à-dire la mesure de son orientation. Cette variable, dite "circulaire", est mesurée selon un gisement (angle mesuré dans le sens horaire par rapport au Nord cartographique). Notons que le nord cartographique est confondu avec le nord géographique au sein de cette projection et que celle-ci est conforme ;
- **Az** est l'azimut correspondant à l'angle du rayon lumineux. Dans le cadre de la projection de Mercator, cet azimut correspond au gisement ;
- **Z** est le zénith, angle d'inclinaison au-dessus de l'horizon du vecteur lumineux.

Les angles sont exprimés en radians.

On remarque donc l'influence de la pente et des facteurs liés à l'éclairement. Ainsi, un pixel présentant une forte pente aura un ombrage s'approchant du blanc (0) s'il est "exposé" à la lumière et, à l'inverse, s'approchant du noir s'il est "masqué" à la lumière. L'exposition du pixel dépendant de l'orientation de son vecteur normal par rapport à celle du vecteur lumineux incident.

Notons également que seules les ombres propres du relief sont prises en compte et non les ombres ombres portées (Figure 20. Par ailleurs, les terrains de pente nulle ne présente pas d'orientation et donc pas d'aspect.

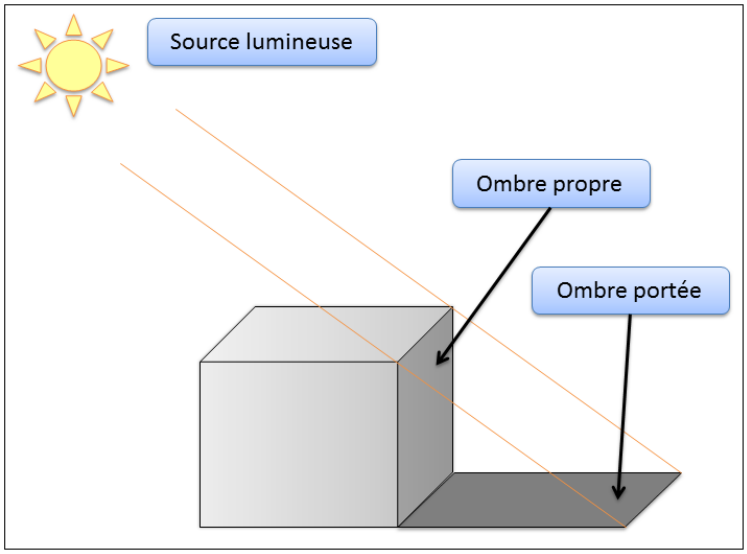

FIGURE 20 – Différentes ombres d'un objet : ombre propre et ombre portée

La commande "hillshade" utilisée est incluse au sein des outils fournis avec gdaldem.

```
gdaldem hillshade srtm_belgium_warped.tif srtm_belgium_hillshade.tif
```

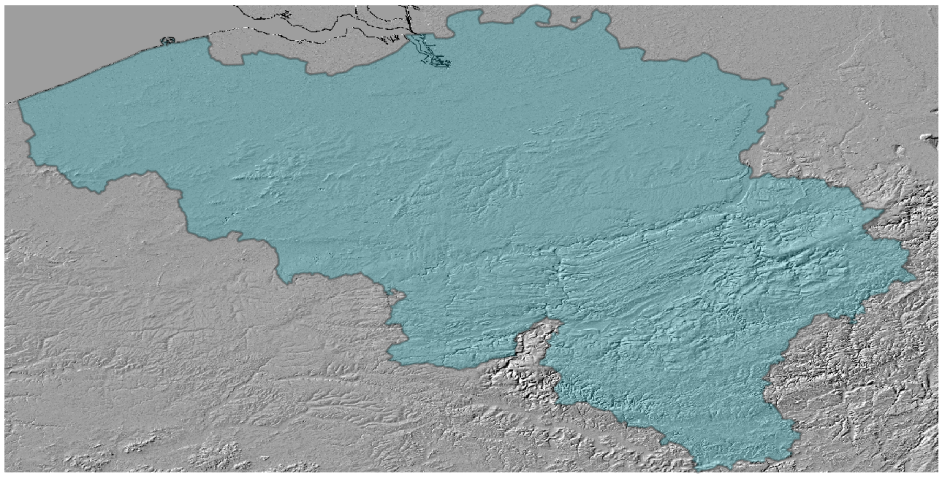

FIGURE 21 – Hillshading du SRTM pour le territoire belge

Suite aux faibles variations du relief présentes sur le territoire belge, une exagération verticale de 4 a été utilisée par le biais du paramètre "z" de la commande :

```
gdaldem hillshade -z 4 srtm_belgium_warped.tif srtm_belgium_hillshade.tif
```

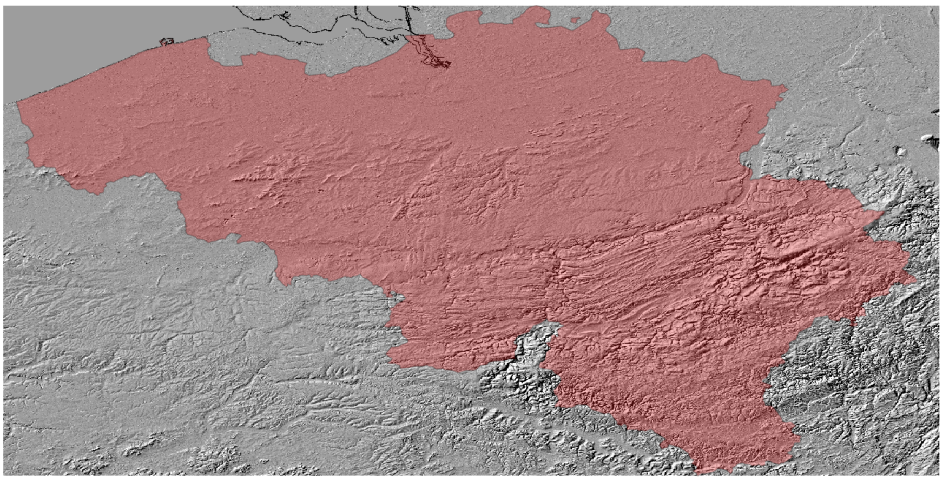

FIGURE 22 – Hillshading du SRTM pour le territoire belge avec une exagération verticale de 4

Découpe du hillshading pour le territoire belge

Afin de restreindre le hillshading à la zone étudiée, une découpe selon les limites du territoire belge a été réalisée par le biais d'un masque. Ce masque, basé sur la couche vectorielle des limites administratives, a permis d'attribuer une valeur de type "NODATA" aux pixels en dehors du territoire, les rendant transparents à l'affichage, bien que ceux-ci restent présents au sein du fichier .tiff. Cette opération est possible en intervenant sur le canal alpha de l'image qui agira sur celle-ci comme un masque binaire.

A cet effet, c'est à nouveau l'outil "gdalwarp" qui est sollicité :

```
gdalwarp -co COMPRESS=DEFLATE -dstalpha -cutline belgium.shp
        srtm_belgium_hillshade.tif srtm_belgium_hillshade_cut.tif
```

- **co COMPRESS=DEFLATE** définit une méthode de compression pour le traitement de fichiers volumineux. La commande est exécutée moins rapidement, car elle nécessite plus de lectures/écritures. Par contre, celle-ci consommera moins d'espace en mémoire ;
- **dstalpha** permet de créer une partie "nodata" transparente en dehors des limites ;
- **cutline** précise les limites du découpage.

FIGURE 23 – Hillshading du SRTM découpé selon les limites administratives du territoire belge avec une exagération verticale de 4

5.5 Visualisation conjointe : OSM et SRTM

5.5.1 Etat de l'art de l'affichage de données SRTM combinées aux données OSM

L'affichage de l'orographie conjointement à des données OSM est dors et déjà fort répandu. On peut citer notamment la plateforme "The OpenCycleMap", qui propose une "slippy map" basées sur les données OSM et comportant des itinéraires cyclables, ainsi que les courbes de niveau et un ombrage du relief. Tous deux sont obtenus obtenus à partir des données SRTM (Gravitystorm Limited, s.d.).

Un autre exemple est celui du portail "OpenTopoMap" ou son équivalant français "FranceTopo". Ceux-ci offrent des cartes selon un style inspiré des cartes topographiques en présentant également des courbes de niveau ainsi qu'un ombrage issus des MNE SRTM/ASTER. (Philibert-Caillat, s.d.; OpenTopoMap, s.d.).

Un dernier exemple, qui reprend en détail toutes les opérations de mise en place de la plateforme nécessaire à une application de type "slippy map" côté serveur, est celui du projet "OpenTrackMap". Celui-ci reprend des parcours de randonnée sur des fonds de cartes OSM/hillshading SRTM (Bartoň, 2010).

5.5.2 Ajout du hillshading au sein du fichier de style

Afin de permettre à Mapnik d'exploiter l'image du hillshading lors de son opération de rendu, il est essentiel de lui spécifier l'emplacement de ce fichier, ainsi que le type qui lui est attribué. Ces précisions s'effectuent au sein du fichier de style au format XML, en ajoutant un style comprenant une **règle** quant au rendu et une **couche**, ainsi que ses paramètres intrinsèques :

```
<Map srs="+proj=merc +lon_0=0 +k=1 +x_0=0 +y_0=0 +ellps=WGS84
        +datum=WGS84 +units=m +no_defs">
  <Style name="hillshade style">
    <Rule name="rule 1">
      <RasterSymbolizer opacity="1" scaling="bilinear" mode="normal" />
    </Rule>
  </Style>
  <Layer name="hillshade">
    <StyleName>hillshade style</StyleName>
    <Datasource>
      <Parameter name="type">gdal</Parameter>
      <Parameter name="file">
      /home/benjamin/SRTM/srtm_belgium_hillshade_z4_cut.tif</Parameter>
    </Datasource>
  </Layer>
</Map>
```

La couche (**layer**) est de type raster (pris en charge par GDAL) et stockée dans un fichier (GeoTIFF) dont on spécifie l'emplacement. Ensuite, le **style** permettra de représenter l'image selon une règle (rule) de type "RasterSymbolizer" dont on peut définir l'opacité et en jouant avec la transparence. Notons qu'il est impératif de réaliser cette opération pour certaines couches de données suite à l'insertion du hillshading. En effet, ces couches cachent celle de l'orographie, entrainant des discontinuités peu esthétiques, sous forme de rupture nette, dans la représentation (Figure 24). Néanmoins, si l'on omet ces défauts de transparence, le rendu opéré permet de percevoir le relief et ses variations de manière assez nette.

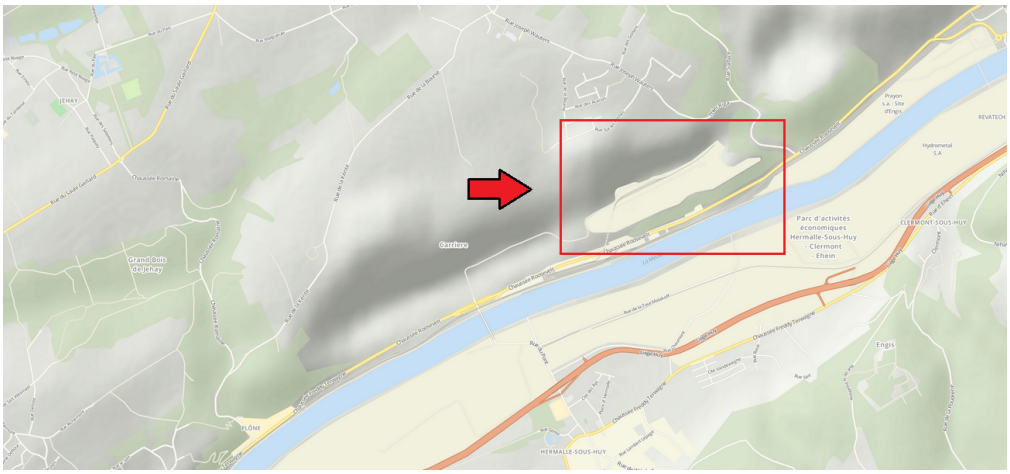

FIGURE 24 – Exemple de discontinuité lors du rendu combinant données OSM et hillshading du SRTM.

FIGURE 25 – Visualisation du rendu combinant données OSM et estompage du SRTM.

5.5.3 Implémentation dans un script python

Il est possible de faire appel à Mapnik depuis un script Python afin de générer une carte au format image. Cette approche a été utilisée afin de tester, au préalable, le rendu incorporant les données OSM et SRTM. Ce rendu est, par la suite, réalisé automatiquement au sein de la plateforme de tuilage. Au sein de ce script, sont précisés :

- **Les dimensions de l'image**. Elles ont été fixées afin de réaliser une tuile rectangulaire (500x1000 pixels) englobant le territoire ciblé ;
- **Le territoire recouvert par la carte**. On parle ici d'une "bounding box" dont l'étendue est définie en latitude et longitude selon le référentiel WGS84 ;
- **La projection de destination**. En effet, Mapnik est capable de reprojeter les données en sollicitant directement la bibliothèque GDAL/OGR. Dans le cas qui nous concerne, c'est la projection "Mercator sphérique" qui a été utilisée ;
- **La définition du style**. Celui-ci est récupéré en consultant le fichier xml associé.

Voici, à titre d'exemple, une tuile générée pour le territoire belge et dont le script figure dans l'annexe C.

FIGURE 26 – Tuile centrée sur Liège selon le style OSM Bright combiné à un hillshading du SRTM

5.5.4 Implémentation au sein d'un serveur web Apache

Une fois le fichier de style correctement réalisé et les différents programmes intervenant lors du rendu installés, il est possible de configurer le serveur web Apache, afin de servir les tuiles de manière automatique à la requête de clients.

Dans un premier temps, les différents **processus** ont été **démarrés manuellement** depuis des terminaux distincts afin de suivre en temps réel les opérations effectuées par Renderd (requête de tuiles selon les déplacements côté client, temps d'exécution, stockage en mémoire cache, etc.) puis selon un **processus automatique** exécuté dès le démarrage du système et en arrière-plan. Les détails de ces opérations sont repris dans l' Annexe D.

Notons que, dans un intérêt d'économie de mémoire ainsi que de performances, un seul style de rendu a été implémenté au sein du serveur : le style associé aux vol VFR de jour. D'autres styles sont néanmoins accessibles à l'utilisateur au sein du prototype. Toutefois, les tuiles de ceux-ci sont générées depuis les serveurs de Mapbox (et non localement) et accessibles par le biais de leur API (Mapbox.js).

6 Interface client

6.1 Environnement client : interface web

L'interface qui permet a l'utilisateur d'exploiter ce prototype se présente sous la forme d'une **interface web**. Celle-ci permet des interactions homme-machine grâce à des pages web accessibles depuis un navigateur. Un exemple de code pour la page contenant la carte figure dans l'Annexe H.

Une des évolutions possible de cette interface serait de l'incorporer à une application mobile ou de permettre un transfert d'informations entre cette interface web, consultée lors de la préparation du vol, et une application qui offrirait ensuite un suivi durant le navigation. En effet, il n'est pas nécessaire, dans ce cas, que l'application comporte toutes les fonctionnalités proposées par l'interface web mais constitue plutôt un complément axé sur la mobilité dont la compatibilité est assurée.

6.1.1 WEB 2.0 : Javascript, JQuery et AJAX

Ce prototype fonctionne entièrement selon le paradigme client-serveur (Section 3) et sa réalisation se base sur les usages communément associés au "Web 2.0". Ce terme générique symbolise la mise en place d'interfaces facilitant les échanges ainsi que les interactions avec les informations. Bien que construit avec des technologies déjà existantes dans le "Web 1.0", celui-ci est axé sur une simplicité d'utilisation ainsi qu'une certaine indépendance de la plateforme d'exécution, ce qui impliquera l'utilisation de mécanismes plus complexes tels que **Ajax**.

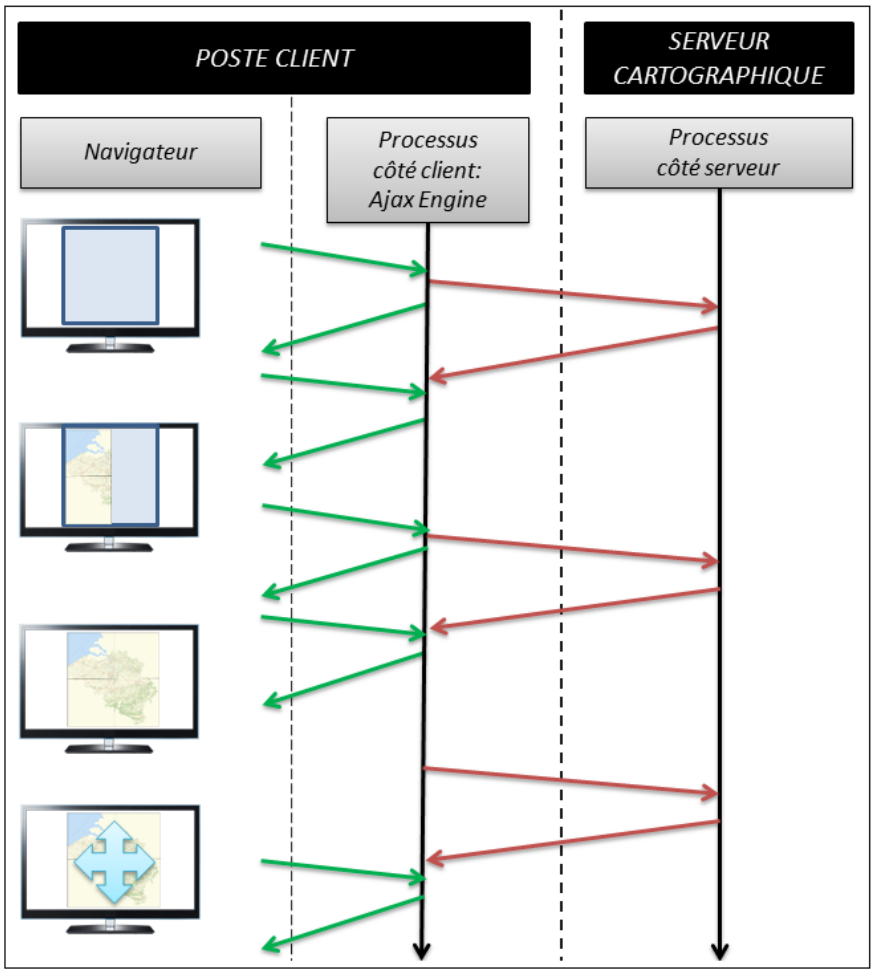

FIGURE 27 – Principe de la communication client-serveur asynchrone grâce à la technologie Ajax

Introduit dans le domaine de la cartographie en ligne par Google Maps à partir de 2004 (Peterson, 2012), **le mécanisme Ajax** permet aux pages web d'exécuter des tâches de *gestion de données* et d'*envoi de requêtes* aux serveurs web à l'arrière plan, *sans que l'utilisateur ne s'en rende compte* (Nixon, 2015). Cet ensemble de méthodes JavaScript veille au transfert de données entre le client et le serveur, sans imposer de recharger la page. Le cas d'une API cartographique en est un bon exemple : les sections de la carte sont téléchargées depuis le serveur de manière asynchrone, sans que cela ne soit visible par l'utilisateur (Figure 27). Parfois, certaines données sont téléchargées de manière à anticiper les actions de l'utilisateur et veiller à la fluidité de la navigation en évitant les interruptions. C'est généralement le cas des tuiles jouxtant la fenêtre d'affichage.

Suite à l'intérêt nouveau porté aux technologies telles que Ajax, la bibliothèque **JQuery** a vu le jour. Celle-ci regroupe plusieurs fonctionnalités du JavaScript, notamment Ajax, afin de faciliter l'intégration de processus exécutés côté client (The jQuery Foundation, 2017). Ces processus sont directement insérés dans le code HTML des pages web sous forme de scripts **JavaScript** et c'est d'ailleurs sur ce principe que fonctionne la bibliothèque **Mapbox.js**, elle-même inspirée de la bibliothèque **Leaflet**, utilisée au sein de ce prototype (Section 6.1.2).

6.1.2 Leaflet et Mapbox API

Afin d'afficher des cartes en ligne incluant les données fournies par le serveur réalisé, la bibliothèque JavaScript **Mapbox.js** a été utilisée au sein des pages web. Celle-ci est directement construite sur la bibliothèque JavaScript **Leaflet** (Mapbox, 2017b) et est développée librement par la société Mapbox, intervenant récurrent en matière d'affichage de données OSM. C'est notamment la société Mapbox qui a procédé au relooking du portail d'OpenStreetMap [34] en 2013 selon le nouveau standard "cartoCSS" dont elle est à l'origine (OpenStreetMap Wiki, 2016a). C'est également Mapbox qui maintient le développement de Mapnik sur la plateforme github.

Le rôle de l'API Mapbox.js est donc d'**afficher une carte interactive et dynamique** à l'utilisateur au sein d'une page web. Celle-ci doit également fournir des outils propres à l'exécution de tâches côté client. Ainsi, les fonctionnalités mentionnées dans la Section 2.5 ont été mises en œuvre par le biais de cette API et de greffons de Leaflet.

Par ailleurs, Mapbox propose un environnement de développement mobile par le biais des librairies Mapbox GL (Mapbox, 2017). Celles-ci permettent d'accéder aux styles définis à l'aide de Mapbox Studio au sein d'une application et prennent également en charge les services de tuilage traditionnels ("slippy map") supportés par Leaflet et Mapbox.js.

Dès lors, cet **environnement** open-source est **opportun pour le développement de services web**, utilisant Mapbox.js, qui sont **compatibles avec des services embarqués au sein d'une application mobile**, réalisés quant à eux avec les outils de Mapbox GL.

6.1.3 Habillage et exhaustivité du prototype

Par soucis d'esthétisme et d'exhaustivité, la carte fournie grâce à l'API Mapbox a été incluse au sein d'un site web complet comprenant différentes pages. Celles-ci constituent l'interface du prototype et sont organisées comme suit :

— Page d'accueil,
— Carte interactive,
— Résumé du projet et de son contexte,
— Tutoriel et initiation à l'utilisation de l'interface.

34. http ://www.openstreetmap.org

6.2 Fonctionnalités du client du prototype

6.2.1 Affichage des fonds de cartes

L'affichage, par le biais de l'API, s'effectue en créant un **objet "Map"** défini dans la bibliothèque Leaflet. Dès lors, il est possible de spécifier certains paramètres d'affichage de la carte par l'intermédiaire de cet objet : le centre initial de la carte ainsi que son niveau de zoom, l'ajout d'une échelle graphique ou encore l'ajout/retrait de certaines couches de tuiles ou vectorielles. Il est également possible de réaliser certaines opérations selon les valeurs qu'ont ses paramètres lors de la navigation. Par exemple, on affiche certaines données lorsque le niveau de zoom associé à l'objet map dépasse un certain seuil.

L'implémentation des différents fonds de cartes s'effectue en définissant des **couches** (layers). Celles-ci peuvent être de différents types selon leur origine. Ainsi, si elles sont générées localement, on utilise la méthode "Tilelayer" de Leaflet. En revanche, si elles sont générées depuis les serveurs Mapbox, on les récupère par le biais de la méthode "StyleLayer" de Mapbox.js. Cette méthode permet de récupérer directement des tuiles générées à la volée selon un style créé via l'interface "Mapbox Studio". Les accès à ces tuiles sont autorisés grâce à une clé liée à l'utilisateur.

Par ailleurs, l'affichage de ces fonds de cartes, qui ne peuvent être visualisés simultanément, est géré par l'utilisateur au sein d'un menu de sélection, le **"layer control"**. Au sein de l'API, les couches tuilées côtoient également des couches vectorielles qui peuvent leur être superposées. Par conséquent, ce menu permet de sélectionner les couches que l'on souhaite afficher au sein de la carte et, parmis elles, ne peut figurer qu'une seule couche tuilée.

Les différents **fonds de cartes** proposés actuellement sont :
— Des tuiles **Mapbox** pour les vols **VFR de jour** ;
— Des tuiles **Mapbox** pour les vols **VFR de nuit** ;
— Des tuiles générées **localement** pour les vols **VFR de jour**.

De plus, des **couches vectorielles** sont également affichables :
— **Les données METAR** des stations sous forme de marqueurs (entités ponctuelles) interrogeables afin d'en obtenir les paramètres ;
— **Les règles de vol** associées aux différentes zones d'influence des stations météo sur base des polygones de Voronoï ;
— **La trajectoire** dessinée par l'utilisateur à l'aide des outils intégrés de dessin lorsqu'il planifie son vol ;
— **Les règles de vol le long de la trajectoire** prévue, extraites par intersection entre les deux couches précédentes.

6.2.2 Récupération des données METAR

Principe

Pour rappel, les données METAR sont stockées dans une base de donnée PostGIS, côté serveur, et accessibles par le biais de GeoServer (Figure 5). Dès lors, il convient de les récupérer au sein de l'interface, côté client, en effectuant une requête selon le mécanisme AJAX, afin que la navigation ne soit pas interrompue lorsque l'utilisateur demande à afficher ces données.

Afin de ne pas transférer les données METAR pour l'ensemble du globe, celles-ci sont **sélectionnées sur base de la fenêtre d'affichage** ("bounding box"). Ainsi, seules les données METAR situées au sein de ce périmètre sont sollicitées. Toutefois, cette **opération** doit être **renouvelée à chaque fois que l'utilisateur déplace la bounding box** lors de sa navigation. Par conséquent, des interactions client-serveur sont inévitables tout au long des déplacements au sein de la carte.

Cette approche serait à éviter au sein d'un appareil mobile lors d'un vol car celui-ci ne bénéficie pas nécessairement d'une connexion performante. Il est donc préférable pour le pilote d'effectuer la **consultation des données METAR** lorsqu'il est **au sol**, bénéficiant ainsi d'une connexion internet fiable. Par la suite,

il lui est proposé de récupérer l'échantillon de données METAR correspondant aux stations le long de sa trajectoire (Section 6.2.4). Ainsi, il peut récupérer les données correspondantes à son vol préalablement à celui-ci et elles seraient stockées directement au sein de l'application de type EFB (Figure 28). Par ailleurs, il est intéressant, d'envisager un **mécanisme de mise à jour** des données METAR **lors du vol**. Cette opération peut être effectuée sur base des identifiants des stations concernées, minimisant ainsi le flux de données à faire transiter par le réseau lors du vol.

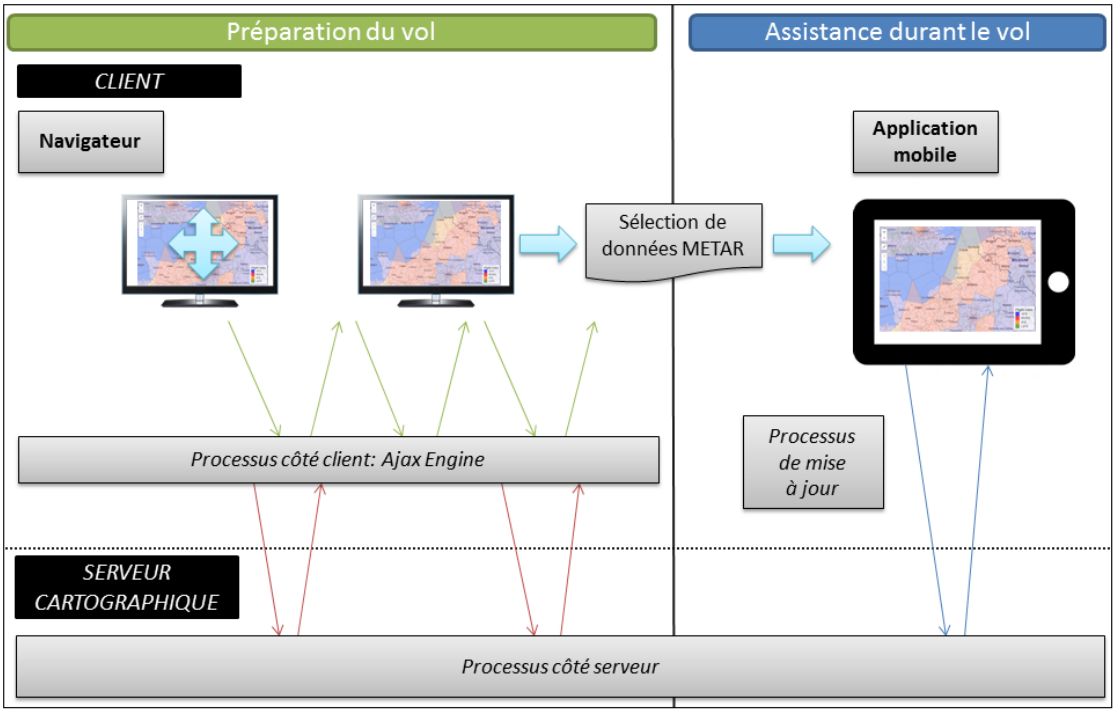

FIGURE 28 – Communication client-serveur lors de la préparation du vol et lors du vol

Le prototype réalisé lors de ce mémoire se situe en amont du vol, lors de la phase de préparation. Par conséquent, une solution d'export des données a également été implémentée.

Réalisation : le greffon leaflet-ajax

Afin de permettre au client de récupérer dynamiquement les données METAR ainsi que les polygones de Voronoï associés, un greffon spécifique de Leaflet a été mis en oeuvre : "leaflet-ajax". Celui-ci permet de récupérer des données vectorielles au format GeoJSON par le biais d'un appel Ajax (Metcalf, 2016). Le format GeoJSON, autre format texte que l'habituel format XML des appels Ajax, permet l'échange de géométries et de leurs attributs selon la norme JSON. Il est, par ailleurs, le format standard utilisé par la bibliothèque Leaflet.

La requête effectuée est une requête "getFeature" selon le service "WFS" et est envoyée vers GeoServer, afin de récupérer des données au format textuel. Les paramètres de la requête, communiqués par le biais de l'url, sont les suivants :

service : 'WFS', *version : '1.0.0',* *request : 'getFeature',* *typeName : 'OSM :m_metar',* *outputFormat : 'application/json'*	*service : 'WFS',* *version : '1.0.0',* *request : 'getFeature',* *typeName : 'OSM :m_voronoi',* *outputFormat : 'application/json'*

FIGURE 29 – Paramètres des requêtes "getFeature" effectuées vers GeoServer afin de récupérer les données METAR et les polygones de Voronoi associés au format GeoJSON par le biais d'un service WFS

Un paramètre variable a été ajouté à cette requête : **la fenêtre d'affichage**. Ainsi, seules les données localisées dans une zone dont l'étendue géographique correspond à celle des données affichées à l'écran sont requises. Ce paramètre est défini de la manière suivante pour les deux types de requête :

bbox : map.getBounds().toBBoxString()

Par conséquent, une requête complète se présente sous la forme :

```
http ://[Adresse IP] :8080/geoserver/OSM/ows ?
service=WFS&
version=1.0.0&
request=getFeature&
typeName=[Couche de données]&
outputFormat=application/json&
bbox=[long2][long1][lat2][lat1]
```

FIGURE 30 – Requête "getFeature" vers GeoServer afin de récupérer les données au sein d'une "bounding box" au format GeoJSON par le biais d'un service WFS

Les résultats de ces requêtes sont incorporés dans une couche vectorielle de type GeoJSON, directement interprétée par Mapbox.js. Les données METAR, représentées de manière ponctuelle, sont interrogeables par l'utilisateur (Figure 31).

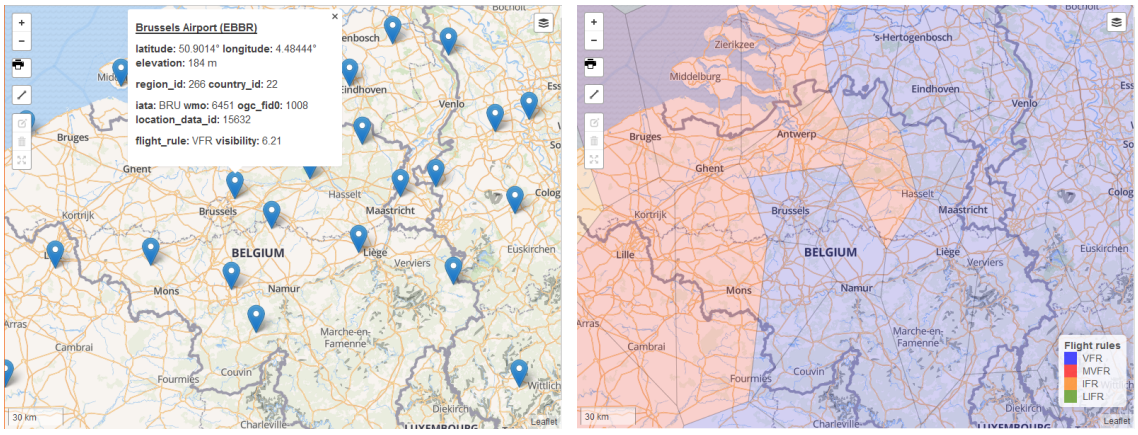

FIGURE 31 – Affichage des données METAR et règles de vol associées au sein de la fenêtre d'affichage

6.2.3 Définition d'une route de navigation aérienne

Principe

Afin d'orienter sa recherche en fonction de ses projets de vol, l'utilisateur a la possibilité d'entrer une trajectoire par l'intermédiaire d'un outil de dessin. Cette trajectoire est réalisable sous forme d'une polyligne

en pointant à l'écran les points de passage (sommets) successifs.

Toutefois, cette polyligne est représentée, dans un premier temps, de manière linéaire dans le système de coordonnées de la carte, c'est-à-dire le pseudo-Mercator. Or, les trajectoires (orthodromies) dans la réalité correspondent à des **arcs de grands cercles**, représentés de manière arquée dans le système de projection pseudo-Mercator. Ainsi, il convient de transformer cette polyligne en une succession d'arcs, à partir des points de passages entrés, afin d'approcher au mieux la trajectoire envisagée (Figure 32).

Bien que cette approche ne présente pas de différence notable sur des trajectoires courtes, les écarts peuvent, toutefois, se montrer conséquents pour de longues distances. En effet, ceux-ci augmentent d'autant plus que la direction de leur trajectoire s'écarte du méridien du point d'origine.

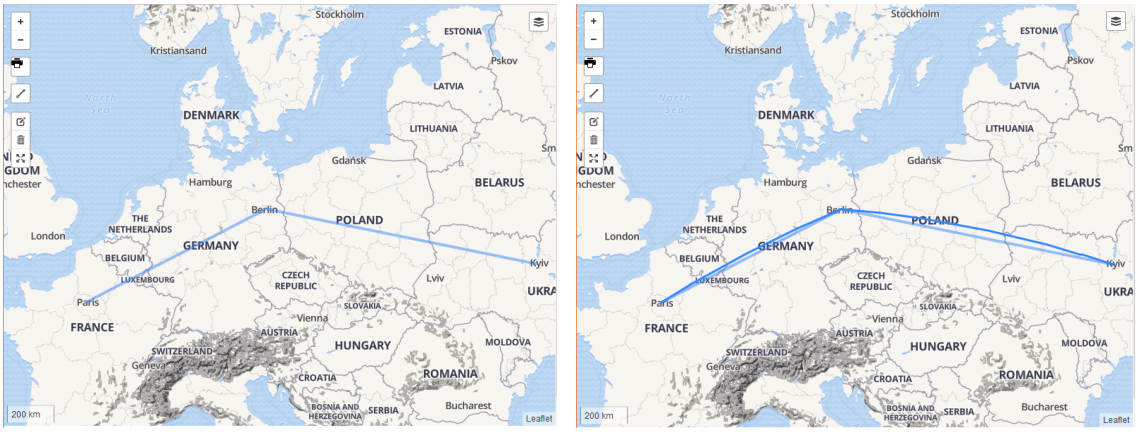

FIGURE 32 – Dessin de trajectoires et d'arcs de grand cercle

Réalisation : le greffon leaflet-draw

Les outils de dessin et d'édition ont été ajoutés en utilisant le greffon "leaflet-draw". Celui-ci permet de bénéficier de solutions de dessin pré-construites directement disponibles au sein d'**une barre d'outils d'édition configurable** (Toye, 2017). Celle-ci est définie en tant qu'objet "L.Control.Draw" dont il ne reste qu'à spécifier les différents paramètres :

— **emplacement** des outils,
— **outils d'édition** disponibles (suppression, couches autorisées à l'édition, etc.),
— **outils de création** de géométries (points, polylignes, cercles, polygones, etc.) proposés ainsi que leur métrique.

Réalisation : le greffon leaflet-arc

La génération d'arcs de grands cercles à partir de leurs extrémités est réalisée par le greffon "leaflet-arc" (Gusev, 2016). Celui-ci permet d'ajouter des objets de type "Arc" directement hérités de l'objet linéaire basique, "Polyline". En effet, étant donné que Leaflet ne prend pas en charge directement les géométries circulaires, ceux-ci ne sont pas réellement des arcs mais plutôt une succession de segments linéaires dont la forme approchera celle de l'arc de cercle.

Les paramètres qu'il est possible de spécifier, en plus des extrémités de l'arc, sont :
— le **nombre de sommets intermédiaires**,
— l'**offset** permettant de gérer la division des segments traversant la ligne de changement de date.

Dès lors, cet objet "Arc" est implémenté au sein d'une fonction "getGreatCircle", **qui a pour rôle de générer des approximations d'arcs à partir des sommets entrés lors du dessin de la trajectoire** et selon une certaine densité de points par mesure de distance. Ainsi, pour ce prototype, les arcs sont construits avec une densité d'un sommet par kilomètre et doivent comporter minimum 2 sommets : leurs extrémités.

6.2.4 Définition d'un couloir de vol et sélection de données météorologiques

Principe

La mise en place d'une certaine marge de sécurité bordant la trajectoire a été réalisée en définissant un **couloir de vol** qui présente un espace tampon (buffer) de part et d'autre de la trajectoire initiale. En effet, il est difficile pour le pilote, dans la pratique, de s'en tenir à une trajectoire correspondant parfaitement à celle prévue initialement. Les raisons de ces écarts potentiels à l'arc de grand cercle de la trajectoire sont multiples mais, parmi elles, on peut citer notamment : la réalisation du vol selon des tronçons à cap constant, le contournement de certaines zones non autorisées ou encore le choix de trajectoires en fonction du vent.

Par ailleurs, la définition d'un couloir de vol permet de réaliser une **sélection des règles de vol en vigueur à proximité de la trajectoire prévue** selon une marge choisie par l'utilisateur. Ainsi, l'utilisateur est capable d'appréhender les conditions de vol et ce, même pour des zones non traversées a priori mais néanmoins avoisinantes à la trajectoire. En effet, les paramètres météorologiques relatifs à ces zones adjacentes peuvent aider le pilote à déduire des informations selon un contexte plus général.

Notons que, lors de la sélection des polygones affectés des règles de vol et situés le long de la trajectoire, c'est un **algorithme de localisation de points** de type **"point-in-polygon"** qui est implémenté. Celui-ci a pour objectif d'évaluer si un point donné est inclus au sein d'un polygone. Ainsi, les points qui constituent les sommets des approximations d'arc de la trajectoire, de même que ceux relatifs aux bords du corridor, sont testés par rapport aux polygones de Voronoï.

Il est donc intéressant, au sein de la couche contenant ces polygones, de ne comporter, au préalable, que ceux contenus au sein de la bounding box et non la totalité de ceux-ci. En effet, chaque point testé sera confronté tour à tour aux polygones, afin de déterminer si il est compris ou non au sein de ces derniers. Cela implique une multitude d'opérations qui, suite à l'architecture déployée, sont exécutées côté client. Dès lors, il est avantageux d'exclure des prétendants potentiels les polygones non contenus au sein de la bounding box, afin de procéder à une **première opération de tri**. Cela implique, toutefois, de *visualiser à l'écran l'ensemble de la trajectoire* lors de l'opération, afin de s'assurer que tous les polygones potentiellement repris lors de la sélection finale soient conservés.

Ensuite, seuls les polygones repris au sein de la fenêtre d'affichage sont considérés comme prétendants au second tri : **la sélection sur base de l'inclusion de points de la trajectoire** ou du couloir de vol associé peut désormais être effectuée. Il en résulte un ensemble de polygones traversés par la trajectoire ou sa zone tampon et celui-ci est exportable au sein du prototype, afin d'être téléchargé puis embarqué dans une application d'aide lors de la navigation. Dans le cas présent, c'est le format GeoJSON qui est utilisé pour l'export car, comme mentionné précédemment, il est sous forme textuelle et nativement supporté par Leaflet et les différentes plateformes Mapbox.

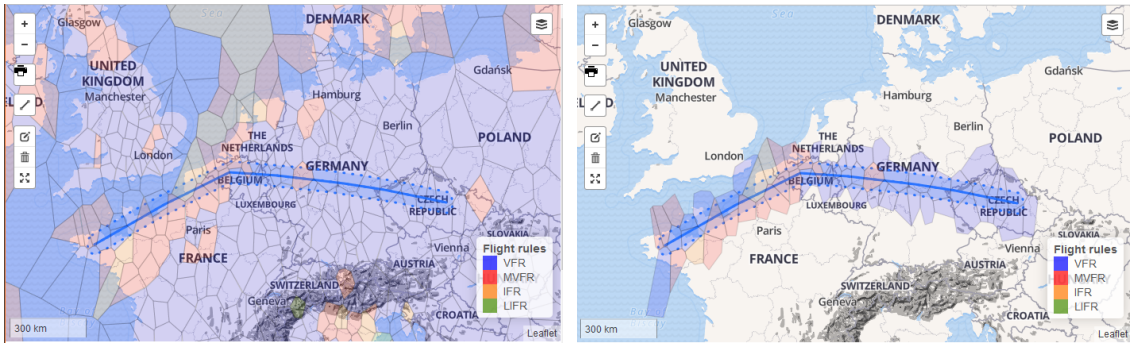

FIGURE 33 – Sélection des règles de vol croisant le couloir de vol (droite) parmi l'ensemble des règles de vol de la fenêtre d'affichage (gauche)

Réalisation : le greffon Leaflet-buffer

La définition d'un couloir de vol s'effectue grâce au greffon "Leaflet-buffer" qui permet d'ajouter une zone tampon autour de géométries (Skeate, 2017). Cette fonctionnalité est directement ajoutée aux outils de dessin de "Leaflet-draw" et précise le rayon de la zone tampon à l'utilisateur lors de son exécution. Notons toutefois que ce greffon n'implémente pas un corridor dans le système de coordonnées de la carte mais bien dans le système de coordonnées d'origine. Il en résulte une déformation de la forme une fois celle-ci projetée en pseudo-Mercator.

En effet, le **corridor symétrique**, tel qu'il serait sans distorsion des longueurs résultantes d'une (re)projection, est construit à partir d'un cercle à chaque sommet de la polyligne puis en traçant des parallèles de part et d'autre des segments (à distance égale au rayon du cercle) jusqu'à ce qu'elles intersectent les cercles aux extrémités (Figure 34, côté droit). Dans ce cas de figure, les mesures sont considérées comme équivalentes selon les différentes directions : le rayon a même longueur selon X et selon Y. Or, suite à la projection utilisée dans le cas présent, une déformation des distances est à prendre en ligne de compte.

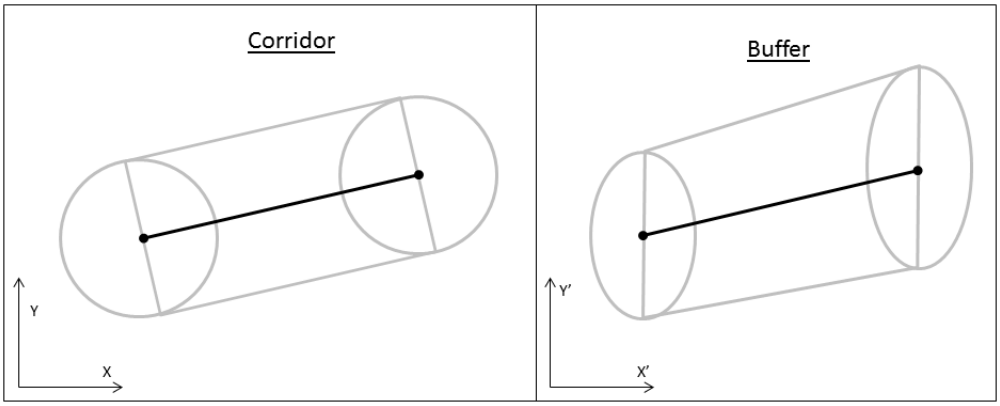

FIGURE 34 – Construction d'un "buffer" et d'un corridor à partir d'un segment de droite

La représentation d'un "**buffer**" telle qu'implémenté est le fruit d'une projection d'un corridor. Sa construction est basée sur les principes suivants :

La **partie entourant les sommets** est construite à partir d'un cercle projeté. Il en résulte une forme elliptique prenant en compte les déformations de distance en X et en Y. Dès lors, ces ellipses sont construites à partir de demi axes correspondants à une même longueur réelle mais dont la représentation n'est pas identique selon X et selon Y dans le plan de projection. Par conséquent, leur forme s'apparente à celle d'un cercle lorsqu'on est à l'équateur et celui-ci s'aplatit de plus en plus à mesure que l'on s'approche d'un pôle (Figure 35).

Par la suite, les **bordures rectilignes** du buffer sont réalisées en reliant les demi-grand axes des ellipses situées aux extrémités. Les valeurs d'abscisse de ces raccords sont identiques à celles du segment d'origine, toutefois, leurs valeurs d'ordonnée ne résultent pas d'une simple translation. En effet, suite à la distorsion des distances, les bordures du buffer ne sont pas exactement parallèles au segment d'origine car elles joignent les demi-grand axes des ellipses situées aux extrémités. Or, bien qu'issus d'une même distance réelle, ceux-ci ne sont pas nécessairement de même longueur au sein de la carte. Il en résulte une forme légèrement conique pour le tronçon (Figure 34, côté gauche).

	Corridor	**Buffer**
Sommet	*Cercle* de rayon fixé (même valeur en X et en Y)	*Ellipse* dont les demi axes (orientés selon X et Y) correspondent chacun à une même distance réelle mais différente en X et en Y une fois projetée
Bordure rectiligne	*Segment parallèle* à l'original et s'arrêtant aux points de tangence avec les cercles construits aux sommets	*Raccord* entre les extrémités des demi-grand axes

TABLE 8 – Comparaison de la construction d'un buffer et d'un corridor

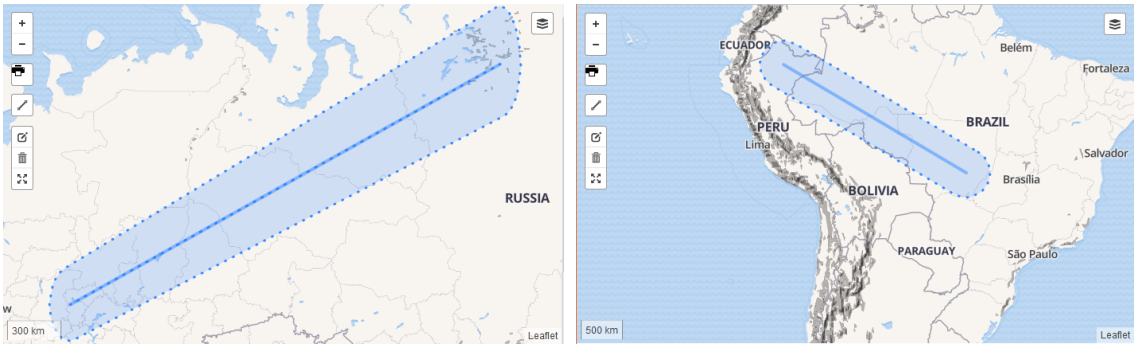

FIGURE 35 – Dessin d'un "buffer" à proximité du pôle (gauche) ou de l'équateur (droite)

Réalisation : le greffon pip (point in polygon)

Le greffon "pip" de Leaflet permet de **récupérer les polygones d'une couche incluant les points d'une autre couche** (MacWright, 2017). Il est lui-même basé sur la librairie "Point_in_polygon" qui effectue le test d'inclusion à proprement parler et fournit une réponse au format booléen pour chaque comparaison point/polygone : "true" ou "false" (Halliday, 2017). Cette dernière bibliothèque implémente un algorithme du "nombre de passage" (crossing number), aussi appelé algorithme du fil à plomb. Celui-ci est lui-même basé sur le *théorème de Jordan* (Sunday, 2012; Bretto, 2012). Le principe de cet algorithme est le suivant :

1. On trace une demi-droite depuis le point évalué ;
2. On calcule le nombre d'intersections (n) que présente cette demi-droite avec le polygone ciblé ;
3. Sur base du nombre d'intersections, on déduit si le point est inclus ou non au sein du polygone (Figure 36). En effet, si n est pair, le point n'est pas inclus. A l'inverse, si n est impair, le point est inclus au polygone.

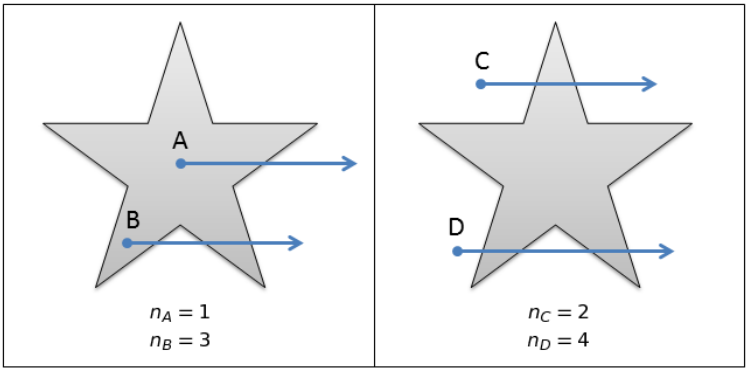

FIGURE 36 – Principe de la méthode *crossing number* : points inclus (A, B) et non inclus (C, D) à un polygone

6.2.5 Fonctionnement étape par étape de l'interface client du prototype

La préparation d'un vol d'école "Paris-Bruxelles" selon des conditions VFR s'effectue de la manière suivante au sein du client de ce prototype :

1. Accès à la page web où se situe la carte et choix des fonds de cartes ;

FIGURE 37 – Étape 1 : accès à la page contenant la carte

2. Définition des points de passage de la trajectoire prévue par le biais de l'outil de dessin ;

FIGURE 38 – Étape 2 : définition des points de passage de la trajectoire

3. Génération automatique (à la demande) de la trajectoire sous forme d'arc de grand cercle et suppression des polylignes de dessin ;

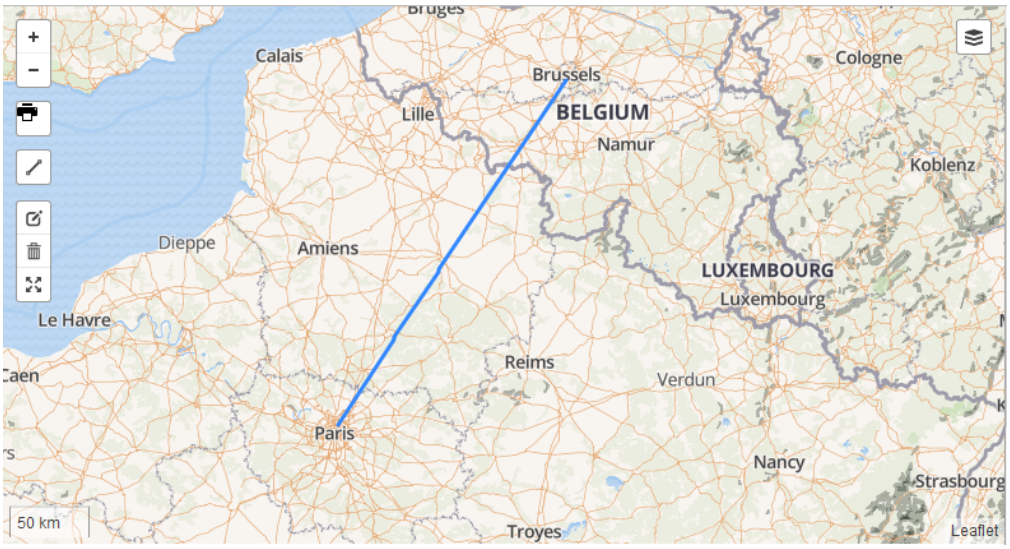

FIGURE 39 – Étape 3 : génération des arcs de grand cercle

4. Définition d'un couloir de vol selon une distance au choix ;

FIGURE 40 – Définition d'un couloir de vol de 15 km autours de la trajectoire

5. Consultation des données METAR et des règles de vol. Si les conditions ne permettent pas de voler selon des conditions VFR, la trajectoire peut être éditée (retour à l'étape 2) ;

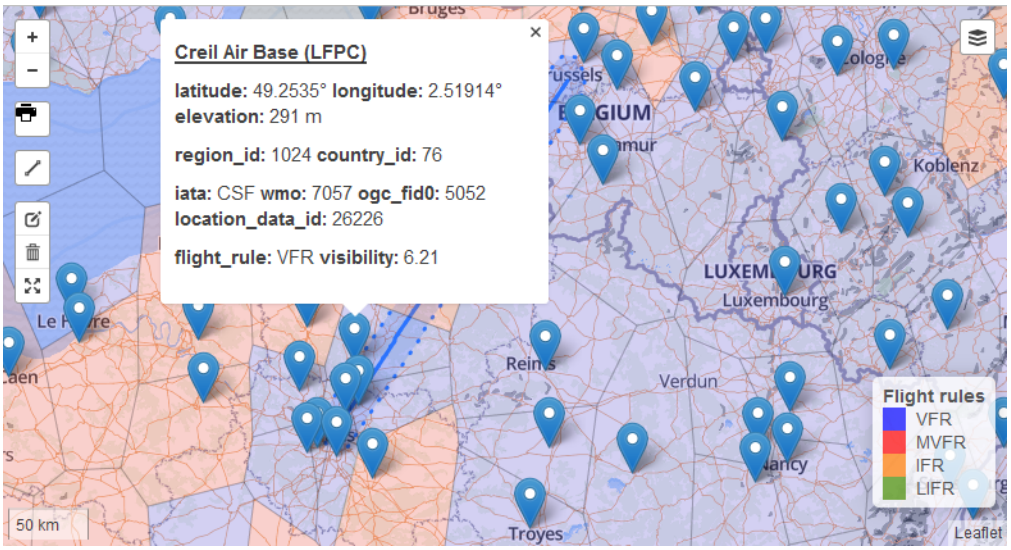

FIGURE 41 – Consultation des données METAR pour l'aéroport de Creil et des différentes règles de vol

6. Sélection des règles de vol en vigueur au sein du couloir de vol ;

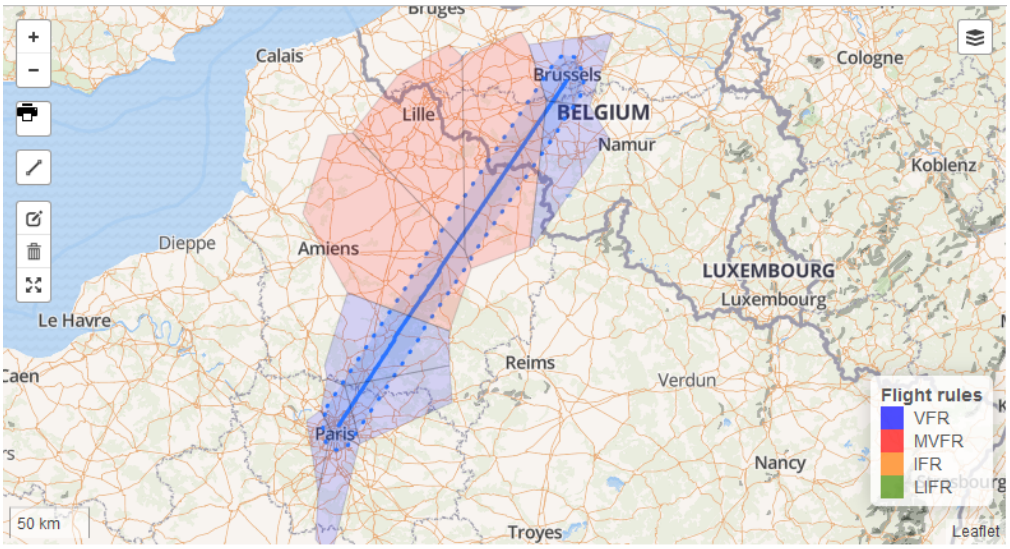

FIGURE 42 – Sélection des règles de vol

7. Export des géométries associées à la sélection de règles de vol.


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

{"type":"Feature","id":"m_voronoi.1469",
"geometry":{"type":"Polygon","coordinates":[[[2.077572,50.083545],
        [2.296611,50.515712],[2.505659,50.524098],[3.473092,49.882125],
        [3.526972,49.781076],[3.343296,49.429779],[2.461804,49.648949],
        [2.077572,50.083545]]]},
"geometry_name":"geometry",
"properties":{"id":1469,"ogc_fid0":1469,"fr":"MVFR",
        "latitude":49.9715,"longitude":2.69766,"icao":" LFAQ"}},

{"type":"Feature","id":"m_voronoi.1470",
"geometry":{"type":"Polygon","coordinates":[[[3.473092,49.882125],
        [2.505659,50.524098],[2.90719,50.855916],[3.338243,51.009674],
        [3.453196,50.943588],[3.473092,49.882125]]]},
"geometry_name":"geometry",
"properties":{"id":1470,"ogc_fid0":1470,"fr":"MVFR",
"latitude":50.5619,"longitude":3.08944,"icao":" LFQQ"}},

(...)

]}
```

7 Travaux complémentaires

Comme évoqué précédemment au sein de la Section 2.6, ce prototype avait une vocation de prospective et les **éventualités d'améliorations** qu'il laisse entrevoir sont multiples. En effet, les solutions qui ont été déployées peuvent chacune être creusée. Néanmoins, il a fallut délimiter les fonctionnalités couvertes par cette solution sans toutefois exclure des perspectives plus approfondies au sein d'une architecture similaire voire plus complexe.

Ainsi, les expectatives qui sont le fruit de ce mémoire ont été organisées en fonction de l'emplacement auquel elles interviennent au sein de l'architecture : le côté serveur, le côté client de l'interface web ou encore l'application mobile associée à ce prototype et dont elle serait le complément mobile.

7.1 Perspectives du client web

Finalisation des styles de fonds de cartes

Une première amélioration serait la **finalisation des styles** définissant les fonds de cartes. La définition de styles complets et exhaustifs serait un plus et permettrait de s'affranchir totalement des serveurs de Mapbox pour le rendu des données OSM. Dès lors, il serait également possible de pré-générer des jeux de tuiles destinés à être embarqués au sein d'une application mobile de navigation, permettant ainsi la navigation hors ligne.

Intégration des données spécifiques à l'aéronautique

La prise en compte de **données spatiales spécifiques à l'aéronautique** serait une plus-value indéniable. En effet, des informations telles que celles figurant sur les *cartes VAC*, qui décrivent un aérodrome de manière détaillée, ainsi que les informations quant à une approche à vue de celui-ci, sont des éléments indispensables. Leur incorporation au sein de la plateforme en ferait une alliée précieuse du pilote (voir Section 2.6).

Concrètement, ces cartes contiennent des informations relatives aux pistes, aux obstacles, aux trajectoires de départs/arrivées, au parcage, à l'altitude de l'aérodrome, la fréquence radio, les repères au sol, etc. Dès lors, la centralisation de leur contenu au sein d'une base de données spatiales, accessible depuis une plateforme englobant également les données météorologiques ainsi que des fonds de cartes spécifiques, serait un atout.

Complétion des données météorologiques

L'intégration des **données TAF et/ou GRIB files** au sein de la plateforme parachèverait la *centralisation de données météorologiques* au sein d'une même plateforme. Effectivement, la stratégie adoptée pour les données de type METAR, par le biais d'un service WFS, peut s'appliquer aux autres données météorologiques.

Les données de type **TAF** peuvent être gérées de manière similaire aux données de type METAR car elles partagent le même principe de fonctionnement excepté le fait qu'il s'agisse de prévisions et non plus d'observations. Par conséquent, une couche ponctuelle ainsi qu'une autre couche sous forme de polygones de Voronoï matérialisant leur zone d'influence peuvent être implémentées.

Les données **GRIB**, quant à elles, sont fournies selon un échantillonnage systématique (à 3 dimensions pour certaines variables) qu'il serait intéressant de disposer en surcharge de la carte. Toutefois, la gestion des différentes altitudes et variations au cours du temps des paramètres constitue un vrai challenge. Par ailleurs, l'interprétation de ces données météorologiques ainsi que l'interpolation des phénomènes spatialement continus qu'elles modélisent seraient des perspectives de travail dans le domaine de la météorologie. Assurément, leur interpolation pourrait s'avérer intéressante lors de la réalisation d'outils d'aide à la décision mettant en rapport ces phénomènes météorologiques et la consommation des avions (voir ci-après).

Réalisation d'un outil d'aide à la décision

Une fois les données météorologiques et leurs interpolation intégrées afin de bénéficier d'une approximation de certains phénomènes, il est envisageable de construire des **outils d'aide à la décision** destinés au pilote. Effectivement, on peut envisager le cas d'une simulation de vol à partir d'une trajectoire spécifiée

qui évaluerait la quantité de carburant consommée à partir du type d'avion et des données relatives aux estimations de vent (direction et vitesse). Ainsi, il serait possible de suggérer des modifications de trajectoire au pilote afin de minimiser sa consommation. En effet, sans nécessairement devoir entrer en détail dans les calculs d'aérodynamisme, il serait déjà possible de suggérer des paliers de vol ou des directions où le vent (vitesse et direction) est plus favorable.

Par ailleurs, l'ACARE[35] travaille actuellement sur un projet de **réduction des émissions de** CO_2 de l'industrie aéronautique européenne. Au sein de cette problématique, s'intègre le programme SESAR[36] destiné à construire un "ciel unique européen" et dont une des préoccupations est l'optimisation des routes aériennes (SESAR Joint Undertaking, s.d.). A cet effet, une plateforme d'évaluation des systèmes de transports aériens est utilisée : la plateforme IESTA. Celle-ci permet, entre-autre, de procéder à des simulations par le biais du programme SimSky (Huynh, s.d.). Dès lors, elle offre la possibilité de développer des simulations de vol à partir de modèles pré-enregistrés correspondant à diverses situations. Ainsi, l'ajout de modules de simulation de consommation serait une amélioration qui, bien que conséquente en développement, proposerait une solution à une problématique d'actualité.

7.2 Perspectives côté serveur

Cohabitation de plusieurs styles de rendu au sein de Renderd

Jusqu'à présent, le rendu effectué par Mapnik était effectué sur base d'un seul style, les autres étant implémentés sur les serveurs de Mapbox et accessibles par le biais de l'API Mapbox.js. Toutefois, il est possible de définir plusieurs styles au sein du fichier de configuration de Renderd (détaillé dans la Section B.3.1 de l'Annexe B). Ainsi, les tuiles seraient générées dans des répertoires différents et accessibles depuis des URL distinctes. Cette approche permet de s'affranchir des données Mapbox, qui ne sont accessibles qu'en ligne. De plus, les tuiles Mapbox ne pourraient être utilisées pour générer directement des tuiles destinées à êtres incorporées à une application de navigation accessible également hors ligne.

Optimisation du processus de rendu

L'architecture informatique en place pour ce prototype n'a pas été évaluée sur un ordinateur offrant des performances adéquates à un "serveur". En effet, les temps d'attente et d'exécution sont trop élevés pour une application "dynamique" alors que le serveur n'a été testé jusqu'à présent que pour une connexion à la fois. Dès lors, il conviendrait de tester la configuration actuelle sur un serveur pour une implémentation réelle. Ensuite, une fois la plateforme déployée sur un serveur, il convient d'en configurer les intervenants pour en optimiser le rendement. Par conséquent, les programmes PostgreSQL, Renderd, Mod_Tile ainsi que la gestion de la mémoire cache sont à configurer minutieusement.

De plus, les données raster de l'estompage du SRTM figurent dans un même fichier GeoTIFF au sein d'un répertoire sur le serveur. Or, pour une implémentation plus conséquente, il serait adéquat de stocker des images de cet estompage au sein d'une base de données spatiales et éviter de ne travailler qu'avec un seul fichier d'autant plus volumineux que le territoire couvert est vaste.

Affectation de certains processus au serveur plutôt qu'au client

La configuration actuelle de ce prototype a montré certaines lacunes lorsqu'il s'agit de calculer la sélection de polygones selon la trajectoire et son couloir de vol (Section 6.2.4). En effet, cette opération est exécutée par le navigateur du côté client car elle est écrite en JavaScript. Dès lors, ce sont les ressources du client qui sont sollicitées et non celles du serveur, qui est généralement plus performant et donc plus apte à réaliser des processus "gourmands" en ressources. Par conséquent, il serait plus opportun de réaliser cette opération du côté serveur par le biais d'un mécanisme tel qu'Ajax, afin d'alléger la charge au client (en présence d'un serveur plus performant que lors de la réalisation de ce prototype). Notons toutefois que les requêtes et résultats de ces processus doivent être communiqués entre le client et le serveur, ce qui implique une réflexion pour réaliser un **compromis entre performances et échanges de données**.

35. Advisory Council for Aviation Research and Innovation in Europe
36. Single European Sky Air Traffic Management Research

7.3 Perspectives au sein d'une application mobile

Adaptations pour réaliser des données embarquées

L'inclusion de données disponibles hors connexion est à prévoir au sein d'une éventuelle application mobile de navigation. En effet, il est préférable d'éviter le chargement de tuiles par l'intermédiaire d'un serveur à distance et favoriser un jeu de tuiles pré-générées stocké au sein de l'application. Dès lors, il convient de définir des jeux de tuiles (réalisés par pays par exemple) accessibles hors ligne et leurs critères de réalisation (quels styles de rendu, quels niveaux de zoom, etc.).

Les données issues de la sélection des polygones contenant les règles de vol doivent être exportables depuis l'interface web à destination de cette application mobile, afin de bénéficier de ces données lors de la navigation. Par ailleurs, un mécanisme de mise à jour de la sélection peut également être envisagé si l'appareil mobile bénéficie d'un accès à internet.

Rendu 3 dimensions

Comme mentionné dans la Section 2.6, le rendu 3 dimensions a été envisagé initialement par ESNAH. Celui-ci est réalisable en recourant à un "game engine" au lieu du "mapping engine" habituel lié aux API de cartographie. Dès lors, ce moteur de rendu aurait permis de réaliser un aplat des données OSM sur les variations (selon 3 dimensions) du relief issues des données SRTM. Toutefois, cette réalisation exploratoire requière des connaissances spécifiques au domaine de la création de jeux et les données géo-spatiales ne sont pas nativement prises en charge au sein de la plateforme. Par conséquent, l'usage du kit de développement "Mapbox Unity" lors de la réalisation d'une application mobile serait une solution à ce problème. En effet, celui-ci permet un rendu 3D et prend en charge les données géographiques de type vectoriel au format GeoJSON ou encore raster au format GeoTIFF. Néanmoins, cette amélioration serait à implémenter au sein d'une application mobile de navigation et montre peu d'intérêt au sein d'une interface de préparation de vol.

8 Conclusions

Démarche

La démarche suivie lors de ce mémoire a été d'élaborer une solution répondant à une question de recherche originale. Cette question a été suscitée par une collaboration avec la société ESNAH, spécialisée dans les systèmes d'aide à la navigation. La solution est matérialisée sous forme d'un prototype selon une architecture client-serveur. Elle a pour mission de permettre aux pilotes de préparer leurs vols préalablement au dépôt de ceux-ci. Cette préparation s'effectue au sein d'une interface client capable de proposer certaines fonctionnalités, dont certaines sont construites à partir de services web fournis par le serveur qui leur est dédié. Ces fonctionnalités, élaborées spécifiquement en considération du contexte aéronautique, sont les suivantes :

1. Consultation de différents fonds de cartes selon le type de vol planifié par le pilote.

2. Consultation de données météorologiques de type METAR ainsi que les règles de vol associées. Ces règles de vol ont été affectées aux polygones de Voronoï générés à partir des entités ponctuelles que sont les stations émettrices.

3. Définition d'une route de navigation aérienne. Celle-ci est générée à partir d'une trajectoire dessinée par l'utilisateur au sein de l'interface client. De plus, elle permet d'extraire une sélection de règles de vol sur base du couloir qu'elle définit.

Cette solution est destinée à être complémentaire ou directement intégrée à une application de navigation en temps réel. Elle permet, effectivement, de récupérer les données météorologiques selon le couloir de vol planifié. Cette sélection de données pourrait donc être importée et visualisée en surcharge des fonds de cartes au sein de l'application de navigation.

Les fonds de cartes originaux, générés au sein du serveur web déployé à partir de données OSM et SRTM, pourraient également être intégrés au sein de cette application de navigation. En effet, ils constituent un support adapté à ce contexte particulier mais leur intégration nécessiterait une adaptation, afin de pouvoir en bénéficier également hors-ligne. Toutefois ces fonds de cartes originaux ne sont disponibles que pour un territoire défini au sein de ce prototype. Les cartes proposant l'ensemble du globe sont, quant à elles, issues de services web externes fournis par Mapbox.

Architecture informatique

Une configuration spécifique à la thématique a été mise en place au sein de ce prototype, plus particulièrement du côté serveur. Celui-ci est organisé en deux strates distinctes.

◇ **La strate de données**

- Le premier jeu de données à considérer au sein du serveur est celui des données OSM. Celles-ci sont importées dans une base de données PostGIS qui sera directement consultée par le moteur de rendu réalisant les tuiles.

- Le second jeu de données est celui consacré à l'orographie. Un estompage (hillshading) est exécuté à partir de données raster SRTM. C'est ce hillshading qui sera exploité lors du rendu par Mapnik.

- Le dernier jeu de données est celui des données météorologiques de type METAR. Dans un premier temps, celles-ci sont récupérées au format XML à l'aide d'une requête de type GET. Cette requête est à effectuer toutes les 30 minutes, afin d'obtenir les mises à jour de ces données. Ensuite, elles sont "parsées" par le biais d'un script python, afin de les importer provisoirement dans une base de données SQLite. Ce format a été choisi afin de correspondre aux traitements effectués au sein des développements d'ESNAH. En effet, celui-ci est propice au stockage de données au sein d'une base directement contenue sur un appareil mobile. Par la suite, ces données sont importées dans la même base de données PostGIS que les données OSM.

◇ **La strate application**

- La première mission affectée au serveur au sein de cette strate est de desservir des tuiles générées spécifiquement pour ce contexte. Dans ce but, différents programmes ont été combinés, afin de fournir ce service par le biais d'un serveur web. Le premier programme est le moteur de rendu cartographique Mapnik. Celui-ci permet de générer des tuiles selon des critères de rendu particuliers, élaborés selon le contexte qui nous occupe. Les deux autres sont le daemon Renderd et le module Apache Mod_Tile. Ils permettent, une fois associés à Mapnik, de réaliser une application de tuilage de type carte glissante. Cette carte est, dès lors, consultable au sein d'une interface web pourvue d'une bibliothèque adéquate, prenant en charge les fonds cartographiques tuilés.

- La seconde mission réalisée par le serveur au sein de cette strate est la gestion des données de type METAR. Ces données sont stockées au sein de la base de données PostGIS et leur gestion est effectuée par le biais de Geoserver. Celui-ci permet de diffuser ces données aux clients par le biais d'un service WFS.

Traitement des données météorologiques

Les données METAR sont distribuées selon un ensemble d'entités ponctuelles correspondant aux stations d'émission. Chaque station ayant une observation METAR d'application à un instant donné. Des polygones de Voronoï ont été générés à partir de ces stations, afin que chaque point de l'espace soit associé à la station la plus proche. A ces polygones sont affectés les mêmes attributs que les données METAR ponctuelles. Cependant, c'est l'attribut de la règle de vol en vigueur qui est exploité pour réaliser une classification. Celle-ci permet d'indiquer au pilote la règle de vol de la station la plus proche lorsqu'il consulte l'interface client. Les deux types de géométries sont ensuite stockées au sein de la même base de données PostGIS que les données OSM. Ces opérations devant être réalisées de manière automatique toute les 30 minutes, leur implémentation s'effectue par le biais d'un script python dont l'exécution systématique est programmable au sein du serveur.

Planimétrie et orographie

Bien qu'utilisé au sein d'une interface web jusqu'à présent, les fonds de cartes ont pour objectif final d'être embarqués au sein d'une application de navigation aérienne. Par conséquent, ceux-ci sont réalisés selon une classification spécifiquement élaborée pour le contexte aéronautique. De plus, l'orographie est également présente sous forme d'estompage, afin de fournir au pilote une perception du relief et de ses variations.

Les fonds de cartes sont réalisés en associant des données OSM et des données raster SRTM lors du rendu :

- Les données OSM sont des données vectorielles organisées sous forme de nœuds, chemins et relations. Elles sont affectées de "tags" qui permettent une classification de celles-ci. Une classification spécifique a d'ailleurs été réalisée sur base de critères propres à l'aéronautique. Elle comporte trois styles différents selon le mode de vol envisagé (VFR de jour, VFR de nuit, IFR) ;

- Les données altimétriques SRTM ont été préférées aux données ASTER, suite aux corrections plus approfondies qui sont disponibles. Elles sont utilisées afin de réaliser un estompage d'ombre visant à générer une image en simulant un éclairement oblique du relief. Cet estompage permet de percevoir le relief et ses variations. Lors de la réalisation du rendu à partir des données du territoire belge, une exagération verticale a été utilisée, afin d'amplifier les variations du relief peu marqué de la Belgique.

Bien que la définition des styles associés au rendu n'a pas été totalement finalisée, des résultats assez encourageants sont obtenus lors de la visualisation conjointe des données. La perception du relief et de ses variations est assez aisée grâce à l'exagération verticale utilisée. Toutefois, le fichier de style définissant les règles du rendu devrait encore être amélioré pour un rendu totalement abouti. En effet, la transparence de certaines couches doit encore y être modifiée afin de laisser transparaître l'orographie sous-jacente.

Interface client

L'implémentation des différentes solutions a été réalisée en créant une interface client. Celle-ci est construite sur base d'une interface web combinant HTML et JavaScript. Elle est toutefois susceptible d'être adaptée à un environnement mobile fonctionnant sur le même principe, à la seule exception que certaines données devront être embarquées au sein de l'appareil pour permettre un mode hors-ligne.

Cette interface web utilise la bibliothèque JavaScript Mapbox.js, directement inspirée de la bibliothèque Leaflet. Dès lors, elle est capable d'afficher des cartes interactives et dynamiques. Parmi les cartes proposées, figure la carte glissante générée sur le serveur de ce prototype. D'autres cartes sont également disponibles par le biais de l'API Mapbox. Celles-ci présentent l'avantage de couvrir l'entièreté du globe, contrairement au prototype implémenté qui est limité au territoire belge.

Les fonctionnalités proposées au sein de cette interface sont les suivantes :
- Affichage des différents fonds de cartes et des couches vectorielles disponibles en surcharge ;
- Récupération des données METAR, ainsi que les polygones de Voronoï associés, au sein de la fenêtre d'affichage sans devoir interrompre la navigation ;
- Définition d'une route de navigation aérienne ;
- Construction d'un couloir de vol et sélection des données météorologiques le long de celui-ci ;
- Export des géométries récupérées après sélection.

Certaines de ces fonctionnalités pourraient être modifiées, afin d'être exécutées par le serveur plutôt que par le client. En particulier, le test d'inclusion de points au sein de polygones, réalisé lors de la sélection des données météorologiques, est susceptible de s'interrompre selon les performances du poste client. Un compromis entre performances et échanges de données est à élaboré, dans ce cas.

Travaux complémentaires

Bien que le prototype répond aux attentes fixées après réévaluation des ambitions initiales, il reste une série de perspectives dont certaines se sont révélées au cours du développement. Les principales sont reprises ci-après.

◇ **Les perspectives côté client**

Une première perspective serait la finalisation des styles associés aux fonds de cartes. En effet, la transparence n'étant pas définie pour chacune des couches, certaines classes d'entités géographiques masquent l'orographie sous-jacente. Par ailleurs, une implémentation complète de la classification spécifique à l'aéronautique reste à réaliser.
Une autre perspective importante serait l'intégration des données aéronautiques au format AIXM. Celles-ci comportent des données aéroportuaires détaillées ainsi que des informations relatives aux approches des aéroports.

◇ **Les perspectives côté serveur**

Une première amélioration, incontournable pour l'élaboration d'une plateforme totalement indépendante de services web externes, serait la cohabitation de plusieurs styles de rendu au sein du daemon Renderd. Ceci permettrait de s'émanciper des serveurs de Mapbox et de pouvoir générer des tuiles selon une projection de Mercator "vraie" dont la conformité est assurée.

◇ **Les perspectives au sein d'une application mobile**

L'intégration des données fournies (fonds de cartes et météorologiques) au sein d'une application de navigation serait une perspective. Celle-ci permettrait d'intégrer des données générées spécifiquement selon les besoins associés au contexte. Elle sera, par ailleurs, certainement exploitée par ESNAH.
La réalisation de données embarquées serait une perspective incontournable dans le cas d'une application de navigation. En effet, il serait préférable de pouvoir bénéficier des fonds de carte lorsque l'appareil supportant l'application est hors-ligne.

Références

Eurocontrol (2017). Aixm - aeronautical information exchange model. Consulté le 19/05/2017.
URL http://www.aixm.aero/

Neogeo-online (2012). Safe software | fme | integrate data, applications, web services. Consulté le 24/05/2017.
URL http://www.neogeo-online.net/blog/archives/1727/

Safe Software (2017). Safe software | fme | integrate data, applications, web services. Consulté le 19/05/2017.
URL https://www.safe.com/

AGG Project (2006). Anti-grain geometry - reference manual. Consulté le 25/05/2017.
URL http://antigrain.com/doc/index.html

Aitchison, A. (2013). Importing DEM Terrain Heightmaps for Unity using GDAL.
URL https://alastaira.wordpress.com/2013/11/12/importing-dem-terrain-heightmaps-for-unity-using-gdal/

Avsoft (s.d.). AvPlan EFB. Consulté le 13/05/2017.
URL http://www.avplan-efb.com/

Bartoň, R. (2010). Custom OpenStreetMap Rendering – OpenTrackMap Experience. *Geoinformatics FCE CTU*, *4*(0), 5–28.
URL https://ojs.cvut.cz/ojs/index.php/gi/article/view/2715

Bartok, J., Habala, O., Bednar, P., Gazak, M., & Hluchý, L. (2010). Data mining and integration for predicting significant meteorological phenomena. *Procedia Computer Science*, *1*(1), 37–46.

Bikecitizens (2017). Bikecitizens - urban independence. Consulté le 07/06/2017.
URL http://www.bikecitizens.net/

Bogdos, N., & Manolakos, E. S. (2013). A tool for simulation and geo-animation of wildfires with fuel editing and hotspot monitoring capabilities. *Environmental Modelling and Software*, *46*, 182–195.

Boots, B. N. (1986). *Voronoi (Thiessen) Polygons*, vol. 45. Geo books.

Bretto, A. (2012). *Eléments de théorie des graphes.*. Collection IRIS. Dordrecht : Springer.

Bukowski, B. (2007). China Airlines B747-400 B-18210.
URL https://commons.wikimedia.org/wiki/File:Boeing_747-400_Dreamliner_CI_B-18210.jpg#

CGIAR CSI (s.d.). Srtm 90m digital elevation database v4.1. Consulté le 04/06/2017.
URL http://www.cgiar-csi.org/data/srtm-90m-digital-elevation-database-v4-1

Comber, A., See, L., Fritz, S., Van Der Velde, M., Perger, C., & Foody, G. (2013). Using control data to determine the reliability of volunteered geographic information about land cover. *International Journal of Applied Earth Observations and Geoinformation*, *23*, 37–48.

Communauté francophone d'utilisateurs d'Ubuntu (2017a). Cron - documentation ubuntu. Consulté le 24/05/2017.
URL https://doc.ubuntu-fr.org/cron

Communauté francophone d'utilisateurs d'Ubuntu (2017b). Documentation ubuntu francophone : Linux mint. Consulté le 20/05/2017.
URL https://doc.ubuntu-fr.org/linux_mint

Communauté francophone d'utilisateurs d'Ubuntu (2017c). Documentation ubuntu francophone : Serveur. Consulté le 20/05/2017.
URL https://doc.ubuntu-fr.org/serveur

Conte, D., Miglietta, M. M., & Levizzani, V. (2011). Analysis of instability indices during the development of a Mediterranean tropical-like cyclone using MSG-SEVIRI products and the LAPS model. *Atmospheric Research*, *101*(1), 264–279.

Coward, D. (2001). *Java^{TM} Servlet Specification - version 2.3*. Sun Microsystem.
URL http://download.oracle.com/otn-pub/jcp/7840-servlet-2.3-spec-oth-JSpec/servlet-2_3-fcs-spec.pdf?AuthParam=1495970843_eba8554fc2b23f014ff17887fc75848b

Czogalla, O. (2015). Smart phone based indoor navigation for guidance in public transport facilities. *IFAC-PapersOnLine*, *48*(10), 233–239.

Doumit, J. A. (2013-2014). Comparison of srtm dem and aster gdem derived digital elevation models with elevation points over the lebanese territory. *HANNON, Lebanese journal of Geography*, *27*, 7–28.

Eastcott, D. (s. d.). *MAPNIK - XML Schema Reference for the Map Definition File*. Consulté le 06/06/2017.

ESNAH sprl (2017). Esnah - aeronautical navigation systems. Consulté le 01/06/2017.
URL http://www.esnah.com/esnah/index.php?lang=fr

Farr, T. G., Rosen, P. A., Caro, E., Crippen, R., Duren, R., Hensley, S., Kobrick, M., Paller, M., Rodriguez, E., Roth, L., Seal, D., Shaffer, S., Shimada, J., Umland, J., Werner, M., Oskin, M., Burbank, D., & Alsdorf, D. (2007). The Shuttle Radar Topography Mission. *Reviews of Geophysics*, *45*(2).

Federal Aviation Administration - U.S.Department of Transportation, Flight Standards Service (2009). *Advanced Avionics Handbook*. Consulté le 13/05/2017.
URL https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/advanced_avionics_handbook/

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. thesis, University of California, Irvine.

Fleischmann, P., Pfister, T., Oswald, M., & Berns, K. (2016). Using openstreetmap for autonomous mobile robot navigation. In *International Conference on Intelligent Autonomous Systems*, (pp. 883–895). Springer.

GDAL (2017a). Gdal - geospatial data abstraction library. Consulté le 26/05/2017.
URL http://www.gdal.org/

GDAL (2017b). Gdal : ogr2ogr. Consulté le 23/05/2017.
URL http://www.gdal.org/ogr2ogr.html

GDAL (s.d.). Gdal : gdalwarp. Consulté le 05/06/2017.
URL http://www.gdal.org/gdalwarp.html

Goetz, M. (2012). Using crowdsourced indoor geodata for the creation of a three-dimensional indoor routing web application. *Future Internet*, *4*(2), 575–591.

Gravitystorm Limited (s.d.). Opencyclemap. Consulté le 04/06/2017.
URL http://francetopo.fr/

Groen, E., Jansen, C., Van Erp, J., & Van Veen, H.-J. (2008). Tactile Displays in the Cockpit : Developments in the Netherlands. American Institute of Aeronautics and Astronautics.
URL http://arc.aiaa.org/doi/10.2514/6.2008-7154

Gusev, A. (2016). Github - leaflet.arc : Leaflet.js plugin for drawing great circle arcs using arc.js. Consulté le 09/06/2017.
URL https://github.com/MAD-GooZe/Leaflet.Arc

Halliday, J. (2017). Github - point-in-polygon : determine if a point is inside a polygon. Consulté le 09/06/2017.
URL https://github.com/substack/point-in-polygon

Hombiat, A., Villanova-Oliver, M., & Gensel, J. (2015). Un nouveau méta-modèle pour rapprocher la folksonomie et l'ontologie d'OSM. In *CEUR Workshop Proceedings*, vol. 1535, (pp. 103–119). CEUR-WS.

Houghton, A., Prudent, N., Scott, J. E., Wade, R., & Luber, G. (2011). Climate change-related vulnerabilities and local environmental public health tracking through GEMSS : A web-based visualization tool. *Applied Geography*.

Huynh, N. (s.d.). Onera - simsky - le coeur informatique de la plateforme iesta. Consulté le 11/06/2017. URL http://www.onera.fr/fr/dcps/simsky

ICAO (2005a). Meteorological service for international air navigation (annex 3). *Convention on International Civil Aviation*, (pp. 6–1 6–3).

ICAO (2005b). Rules of the air (annex 2). *Convention on International Civil Aviation*, (10), 34.

ICAO (2005c). Rules of the air (annex 2). *Convention on International Civil Aviation*, (10), 36–37.

ICAO (2005d). Rules of the air (annex 2). *Convention on International Civil Aviation*, (10), 27–29.

International Association of Oil And Gas Producers (2017). Epsg geodetic parameter dataset - epsg :3857. Consulté le 25/05/2017.
URL http://www.epsg-registry.org/report.htm?type=selection&entity=urn:ogc:def:crs:EPSG::3857&reportDetail=short&style=urn:uuid:report-style:default-with-code&style_name=OGP%20Default%20With%20Code&title=EPSG:3857

ISO (2005). *ISO 19128 :2005. Geographic information - Web map server interface*. International Organization for Standardization.

IVAO (2005). Meteorological service for international air navigation (annex 3). *Convention on International Civil Aviation*, (pp. 4–1).

IVAO (2014a). *Les règles de vol IFR*. International Virtual Aviation Organisation (division France).

IVAO (2014b). *Les règles de vol VFR*. International Virtual Aviation Organisation (division France).

Jet Propulsion Laboratory (NASA) (2017). Nasa - shuttle radar topography mission. Consulté le 22/05/2017. URL https://www2.jpl.nasa.gov/srtm/

Kalašová, A., & Krchová, Z. (2013). Telematics applications and their influence on the human factor. *Transport Problems*, *8*(2), 89–94.

Kresse, W. (2012). *Springer Handbook of Geographic Information.*. Berlin, Heidelberg : Springer Berlin Heidelberg.

Ky, P. (2017). Annual safety review. Tech. rep., European Aviation Safety Agency.

Larousse (s.d.). Définitions : avionique - Dictionnaire de français Larousse. Consulté le 18/05/2017. URL http://www.larousse.fr/dictionnaires/francais/avionique/7103

Le Monde (2017). Cinq morts dans un accident d'avion au Portugal, dont trois Français. *Le Monde.fr*. URL http://www.lemonde.fr/europe/article/2017/04/17/cinq-morts-dans-un-accident-d-avion-au-portugal-5112568_3214.html

MacWright, T. (2016). The end of cartocss | mapbox. Consulté le 26/05/2017. URL https://www.mapbox.com/blog/the-end-of-cartocss/

MacWright, T. (2017). Github - leaflet-pip : point in polygon intersections for leaflet. Consulté le 09/06/2017. URL https://github.com/mapbox/leaflet-pip

Mapbox (2016). Github - mapbox/osm bright : A carto template for openstreetmap data. Consulté le 26/05/2017. URL https://github.com/mapbox/osm-bright/

Mapbox (2017). Github - mapbox/carto : fast css-like map stylesheets. Consulté le 26/05/2017. URL https://github.com/mapbox/carto

Mapbox (2017). Mapbox gl js api. Consulté le 09/06/2017.
URL https://github.com/mapbox/leaflet-pip

Mapbox (2017a). Unity | mapbox. Consulté le 07/06/2017.
URL https://www.mapbox.com/unity/

Mapbox (2017b). v3.1.1 javascript library : All. Consulté le 07/06/2017.
URL https://www.mapbox.com/mapbox.js/api/v3.1.1/

Masson, A. (2016). United States Patent : 9511877 - Electronic kit bag.
URL http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&u=%2Fnetahtml%2FPTO%2Fsearch-adv.htm&r=1&p=1&f=G&l=50&d=PTXT&S1=%28masson.INNM.+AND+angela%29&OS=in/masson+AND+angela&RS=%28IN/masson+AND+angela%29

Metcalf, C. (2016). Github - calvinmetcalf/leaflet-ajax : plugin for leaflet for ajax. Consulté le 08/06/2017.
URL https://github.com/calvinmetcalf/leaflet-ajax

Ministère Francais Des Transports De L'Equipement Du Tourisme Et De La Mer (2006a). Règles de l'air et services de la circulation aérienne. *Journal officiel électronique des lois et décrets*, (3), 37.

Ministère Francais Des Transports De L'Equipement Du Tourisme Et De La Mer (2006b). Règles de l'air et services de la circulation aérienne. *Journal officiel électronique des lois et décrets*, (3).

Ministère Francais Des Transports De L'Equipement Du Tourisme Et De La Mer (2006c). Règles de l'air et services de la circulation aérienne. *Journal officiel électronique des lois et décrets*, (3), 29–31.

NASA Jet Propulsion Laboratory (s.d.). Aster global digital elevation map announcement. Consulté le 04/06/2017.
URL https://asterweb.jpl.nasa.gov/gdem.asp

National Oceanic and Atmospheric Administration (2017). Foia : Freedom of information act | national oceanic and atmospheric administration. Consulté le 31/05/2017.
URL http://www.noaa.gov/foia-freedom-of-information-act

Nav Canada (2009). Guide de décodage des metar / speci / lwis. Consulté le 15/05/2017.
URL https://flightplanning.navcanada.ca/cgi-bin/CreePage.pl?Page=weather-products&NoSession=NS_Inconnu&TypeDoc=user-guide&Langue=francais#metar

Navmii (2017). Navmiv products- navmii world. Consulté le 07/06/2017.
URL http://navmii.com/

Neis, P., & Zielstra, D. (2014). Recent Developments and Future Trends in Volunteered Geographic Information Research : The Case of OpenStreetMap. *Future Internet*, *6*(1), 76–106.

Nixon, R. (2015). *Développer un site web en PHP, MySQL et JavaScript*. O'Reilly Media, 4 ed.

NOAA National Weather Service (2017). Awc - aviation weather center -. Consulté le 31/05/2017.
URL https://www.aviationweather.gov/metar

OGC (2010). *OpenGIS® Web Map Tile Service Implementation Standard*. Open Geospatial Consortium. Reference number : OGC 07-057r7.

Open Geospatial Consortium (2014). Ogc web feature service 2.0 interface standard – with corrigendum. Consulté le 27/05/2017.
URL http://docs.opengeospatial.org/is/09-025r2/09-025r2.html

Open Source Geospatial Foundation (2017a). Geoserver. Consulté le 27/05/2017.
URL http://tomcat.apache.org/

Open Source Geospatial Foundation (2017b). Geotools documentation. Consulté le 27/05/2017.
URL http://docs.geotools.org/

Open Source Geospatial Foundation (2017c). Mapcache 1.4 - documentation de mapserver 7.0.5. Consulté le 28/05/2017.
URL http://mapserver.org/fr/mapcache/index.html

Open Source Geospatial Foundation (2017d). Wfs output formats - geoserver 2.12.x user manual. Consulté le 27/05/2017.
URL http://docs.geoserver.org/latest/en/user/services/wfs/outputformats.html#json-and-jsonp-output

Open Source Geospatial Foundation (2017e). Wfs references - geoserver 2.12.x user manual. Consulté le 27/05/2017.
URL http://docs.geoserver.org/latest/en/user/services/wfs/reference.html#wfs-getcap

Open Source Geospatial Foundation (2017f). Wfs settings - geoserver 2.12.x user manual. Consulté le 27/05/2017.
URL http://docs.geoserver.org/latest/en/user/services/wfs/webadmin.html#service-metadata

OpenStreetMap (2017). Github - openstreetmap/mod tile : Renders and serves map tiles using apache. Consulté le 26/05/2017.
URL https://github.com/openstreetmap/mod_tile

OpenStreetMap (2017). Openstreetmap - about. Consulté le 04/06/2017.
URL http://www.openstreetmap.org/about

OpenStreetMap Foundation (2012). Open database license (odbl) v1.0. Consulté le 21/05/2017.
URL https://opendatacommons.org/licenses/odbl/1.0/

OpenStreetMap Foundation (2017). Github - openstreetmap/osm2pgsql. Consulté le 21/05/2017.
URL https://github.com/openstreetmap/osm2pgsql

OpenStreetMap Wiki (2010). Tirex/renderd : Openstreetmap wiki. Consulté le 26/05/2017.
URL http://wiki.openstreetmap.org/wiki/Tirex/Renderd

OpenStreetMap Wiki (2015). Mapnik/hillshading using mapnik, gdal and srmt data. Consulté le 05/06/2017.
URL https://wiki.openstreetmap.org/wiki/Mapnik/Hillshading_using_Mapnik,_GDAL_and_SRMT_data

OpenStreetMap Wiki (2016a). Cartocss : Openstreetmap wiki. Consulté le 26/05/2017.
URL http://wiki.openstreetmap.org/wiki/CartoCSS

OpenStreetMap Wiki (2016b). Mod tile : Openstreetmap wiki. Consulté le 26/05/2017.
URL http://wiki.openstreetmap.org/wiki/Mod_tile

OpenStreetMap Wiki (2016c). Tile disk usage. Consulté le 25/05/2017.
URL http://wiki.openstreetmap.org/wiki/Tile_disk_usage

OpenStreetMap Wiki (2017a). Mapnik : Openstreetmap wiki. Consulté le 25/05/2017.
URL http://wiki.openstreetmap.org/wiki/FR:Mapnik

OpenStreetMap Wiki (2017b). Openstreetmap wiki : Plantet.osm. Consulté le 21/05/2017.
URL https://wiki.openstreetmap.org/wiki/Planet.osm

OpenStreetMap Wiki (2017c). Tags - openstreetmap wiki. Consulté le 04/06/2017.
URL http://wiki.openstreetmap.org/wiki/Tags

OpenTopoMap (s.d.). Opentopomap. Consulté le 04/06/2017.
URL https://opentopomap.org/

OSGeo (2012). Tile map service specification. Consulté le 25/05/2017.
URL http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification

OSM Foundation (2017). Oepnstreetmap foundation - main page. Consulté le 04/06/2017.
URL http://wiki.osmfoundation.org/wiki/Main_Page

OsmAnd (2017). Osmand - offline mobile maps and navigation. Consulté le 07/06/2017.
URL http://osmand.net/

Outdooractive GmbH (2017). Gps tracks : Touren-führer mit topo karten und gps daten (download gpx format) für mtb, bike, wandern und wanderung, fahrrad, mountainbike, skitouren, schneeschuh tour und touren. Consulté le 07/06/2017.
URL http://www.gps-tracks.com/

Peterson, M. P. (2012). *Online Maps with APIs and WebServices*. Lecture Notes in Geoinformation and Cartography. Berlin, Heidelberg : Springer Berlin Heidelberg.

Philibert-Caillat, C. J. (s.d.). Francetopo. Consulté le 04/06/2017.
URL http://francetopo.fr/

Pol, L. V. d. (s.d.). Belux vacc - charts. Consulté le 11/06/2017.
URL http://beluxvacc.org/charts/

Sample, J. T. (2010). *Tile-Based Geospatial Information Systems.*. Boston, MA : Springer US.

Sanjay, K. (s.d.). AvNav EFB. Consulté le 13/05/2017.
URL https://www.avnavefb.com/

SESAR Joint Undertaking (s.d.). Sesar joint undertaking | high performing aviation for europe. Consulté le 11/06/2017.
URL http://www.sesarju.eu/

Simon, A. (2015). Photographie : Diamond da40 diamond star D-ESCR de la société rent a flight.

Skeate, J. (2017). Github - leaflet.buffer : Create buffers around leaflet.draw shapes. Consulté le 09/06/2017.
URL https://github.com/skeate/Leaflet.buffer

SQLite (2017). About sqlite. Consulté le 23/05/2017.
URL https://www.sqlite.org/about.html

Stein, M., & Sandl, P. (2012). *Information ergonomics : A theoretical approach and practical experience in transportation*. Springer-Verlag Berlin Heidelberg.

Sunday, D. (2012). Inclusion of a point in a polygon. Consulté le 09/06/2017.
URL http://geomalgorithms.com/a03-_inclusion.html

Suwandana, E., Kawamura, K., Sakuno, Y., Kustiyanto, E., & Raharjo, B. (2012). Evaluation of ASTER GDEM2 in Comparison with GDEM1, SRTM DEM and Topographic-Map-Derived DEM Using Inundation Area Analysis and RTK-dGPS Data. *Remote Sensing*, *4*(8), 2419–2431.

The Apache Software Foundation (2017). Apache tomcat. Consulté le 27/05/2017.
URL http://tomcat.apache.org/

The Gaia-SINS federated project (2017). Spatialite : mod spatialite. Consulté le 30/05/2017.
URL https://www.gaia-gis.itossil/libspatialite/wiki?name=mod_spatialite

The Gaia-SINS federated project (s.d.). The gaia-sins federated project home-page. Consulté le 23/05/2017.
URL http://www.gaia-gis.it/gaia-sins/

The jQuery Foundation (2017). jquery. Consulté le 07/06/2017.
URL http://jquery.com/

Thomas, G. (2006). Bag of Tricks. *Air Transport World*, *43*(6), 46–48,50.

Tomàs Castellà, R. (2012). *Pedestrian navigation application based on OpenStreetMap*. B.S. thesis, Universitat Politècnica de Catalunya.

Toye, J. (2017). Github - leaflet.draw : Vector drawing and editing plugin for leaflet. Consulté le 09/06/2017. URL `https://github.com/Leaflet/Leaflet.draw`

Turiak, M., Novák-Sedláčková, A., & Novák, A. (2014). Portable electronic devices on board of airplanes and their safety impact. In *Communications in Computer and Information Science*, vol. 471, (pp. 29–37). Springer Verlag.

Unity Technologies (s.d.). Unity - game engine, tools and multiplateform. Consulté le 19/05/2017. URL `https://unity3d.com/fr/unity`

US Departement of Transportation Federal Aviation Administration (1999). *Aviation Weather Formats : METAR/TAF*.

USGS (2015). Shuttle radar topography mission (srtm) | the long term archive. Consulté le 04/06/2017. URL `https://lta.cr.usgs.gov/SRTM`

USGS (s.d.). Aster data : Unlimited public access. Consulté le 05/06/2017. URL `https://lpdaac.usgs.gov/user_resources/outreach_materials/aster_no_charge_promo`

Veronesi, F., & Hurni, L. (2015). A GIS tool to increase the visual quality of relief shading by automatically changing the light direction. *Computers & Geosciences*, *74*, 121–127. URL `http://linkinghub.elsevier.com/retrieve/pii/S0098300414002489`

Wang, Z., Zlatanova, S., Moreno, A., van Oosterom, P., & Toro, C. (2014). A data model for route planning in the case of forest fires. *Computers and Geosciences*, *68*, 1–10.

Wiechel, H. (1878). Theorie und darstellung der beleuchtung von nicht gesetzmässig gebildeten flächen mit rücksicht auf die bergzeichnung. *Civilingenieur*, *24*, 335–364.

World Meteorological Organization (s.d.). Guide to grib. Consulté le 19/05/2017. URL `http://www.wmo.int/pages/prog/www/WDM/Guides/Guide-binary-2.html`

World Wide Web Consortium (2014). Cross-origin resource sharing. Consulté le 12/06/2017. URL `https://www.w3.org/TR/cors/`

Yang, Y., Wilson, L. T., & Wang, J. (2010). Development of an automated climatic data scraping, filtering and display system. *Computers and Electronics in Agriculture*, *71*(1), 77–87.

Zaninetti, J. (2016). Zygrib - grib file viewer. Consulté le 19/05/2017. URL `http://zygrib.org/index.php?page=home`

A Minimums VMC de visibilité et de distance par rapport aux nuages

Classe d'espace aérien	**A* B C D E**	**F G Au-dessus du plus haut des 2 niveaux : 3000 ft AMSL ou 1000 ft AGL**	**F G À ou au-dessous du plus haut des 2 niveaux : 3000 ft AMSL ou 1000 ft AGL**
Distance par rapport aux nuages	**1 500 m** horizontalement **300 m** (1 000 ft) verticalement		**Hors des nuages** et **en vue de la surface**
Visibilité en vol	**8 km** à et au-dessus du FL100 (ou 10 000 ft si l'altitude de transition est supérieure à 10 000 ft) ; **5 km** au-dessous du FL100 (ou 10 000 ft si l'altitude de transition est supérieure à 10 000 ft) ;		La plus élevée des 2 valeurs : **1500 m** (**800 m** pour les hélicoptères) ou **distance parcourue en 30 secondes de vol**

TABLE 9 – Minimums VMC de visibilité et de distance par rapport aux nuages (Ministère Francais Des Transports De L'Equipement Du Tourisme Et De La Mer, 2006a)

B Configuration du serveur de tuiles

La mise en place d'un serveur de tuiles personnalisé a été effectuée dans le cadre de ce prototype. Sa réalisation est basée sur la version standard telle qu'implémentée sur le serveur de tuiles principal "OpenStreetMap.org".
L'installation est basée sur des tutoriels en ligne[37] rédigés pour un ordinateur fonctionnant sous Ubuntu Linux 14.04 LTS. La réalisation de ces opérations a été opérée sur un ordinateur avec Linux Mint Cinnamon 17.3 (également basé sur Ubuntu Linux 14.04).

B.1 Installation des logiciels

B.1.1 Installation de GDAL

Installation de GDAL/OGR avant les autres programmes afin que, lors de la compilation du logiciel de rendu Mapnik depuis la source, les dépendances soient correctement configurées et pouvoir faire appel à GDAL et Mapnik depuis un même script python. Il est préférable d'éviter la version de GDAL fournie avec QGIS car les dépendances ne s'établissent pas toujours correctement lors de l'installation de Mapnik. En effet, Mapnik fait appel à la libraire libtif de GDAL pour les traitements des données raster. Par contre, celle fournie par défaut avec la version de GDAL issue des dépôts "ubuntugis" ne supporte pas les fichiers supérieurs à 4 Go. Par ailleurs, un conflit entre différentes versions de GDAL peut empêcher le processus de s'effectuer. En cas de conflit, il convient de désinstaller toutes les versions présentes de GDAL et de Mapnik, pour ensuite réinstaller successivement GDAL et Mapnik depuis leur code source.

B.1.2 Installation des dépendances

Installation des différents composants par le biais de la commande "apt-get install" afin d'assurer les dépendances requises.

```
sudo apt-get install
        libboost-all-dev
        subversion
        git-core
        tar
        unzip
        wget
        bzip2
        build-essential
        autoconf
        libtool
        libxml2-dev
        libgeos-dev
        libgeos++-dev
        libpq-dev
        libbz2-dev
        libproj-dev
        munin-node
        munin
        libprotobuf-c0-dev
        protobuf-c-compiler
        libfreetype6-dev
        libpng12-dev
        libtiff4-dev
        libicu-dev
```

37. https://switch2osm.org/fr/servir-des-tuiles/mettre-en-place-manuellement-un-serveur-de-tuiles-14-04/ et http://adrien.caillot.free.fr/?p=5362

```
        libgdal-dev
        libcairo-dev
        libcairomm-1.0-dev
        apache2
        apache2-dev
        libagg-dev
        liblua5.2-dev
        ttf-unifont
        lua5.1
        liblua5.1-dev
        node-carto
```

B.1.3 Installation de PostgreSQL/PostGIS

Installation

```
sudo apt-get install
        postgresql
        postgresql-contrib
        postgis
        postgresql-9.3-postgis-2.1
```

Création de la base de données et de l'utilisateur

```
sudo -u postgres -i
createuser mapnik
createdb -E UTF8 -O mapnik gis
exit
```

Note importante : il est utile de créer un utilisateur identique à l'utilisateur Ubuntu (exemple : benjamin). Par conséquent, lors des accès à la base de données lors des générations de tuiles, Mapnik sollicitera ce nom d'utilisateur *par défaut* (ainsi qu'un dbname = 'gis') et tentera d'accéder à la base de données avec cet identifiant.
Toutefois, ces paramètres peuvent être édités dans le fichier "datasource-settings.xml.inc" (voir B.2.3).
Dans le cas présent **l'utilisateur utilisé pour la manipulation sera postgres** lui-même.

Mise en place de PostGIS

```
sudo -u postgres psql
\c gis
CREATE EXTENSION postgis;
ALTER TABLE geometry_columns OWNER TO username;
ALTER TABLE spatial_ref_sys OWNER TO username;
\q
exit
```

Attention à modifier "username" en conséquence.

B.1.4 Installation de osm2pgsql

Compilation depuis la source pour bénéficier de la dernière version :

```
mkdir ~/src
cd ~/src
```

```
git clone git://github.com/openstreetmap/osm2pgsql.git
cd osm2pgsql
./autogen.sh
./configure
make
sudo make install
```

B.1.5 Installation de Mapnik

Facultatif : suppression d'une éventuelle version antérieure :

```
sudo apt-get purge libmapnik* mapnik-* python-mapnik
```

Vérification que la version présente de boost est assez récente (>1.46) :

```
apt-cache policy libboost-dev
```

Si la version est antérieure à la version 1.47 :

```
sudo add-apt-repository ppa:mapnik/boost
sudo apt-get update
sudo apt-get install
        libboost-dev
        libboost-filesystem-dev
        libboost-program-options-dev
        libboost-python-dev
        libboost-regex-dev
        libboost-system-dev
        libboost-thread-dev
```

Compilation de mapnik depuis la source :

```
git clone https://github.com/mapnik/mapnik mapnik-2.3.x -b 2.3.x --depth 10
cd mapnik-2.3.x
./configure && make && sudo make install
```

B.1.6 Installation de Mod_Tile et Renderd

```
cd ~/src
git clone git://github.com/openstreetmap/mod_tile.git
cd mod_tile
./autogen.sh
./configure
make
sudo make install
sudo make install-mod_tile
sudo ldconfig
```

B.2 Configuration des feuilles de style

Le rendu sera basé sur le thème d'OSM Bright écrit en CartoCSS. Ce langage a pour avantage d'être facile à interpréter car il est proche du CSS utilisé en design web et peut être lu par des utilitaires, tels que Tilemill, Kosmtik, Sputnik ou Mapbox Studio. C'est d'ailleurs ce dernier qui sera utilisé pour créer le style final spécifique au contexte aéronautique demandé.

B.2.1 Téléchargement de OSM Bright

Répertoire de travail
Les données sont téléchargées dans le répertoire /usr/local/share/maps/style.
ATTENTION : ce répertoire nécessite les droits de super-utilisateur (sudo).

```
mkdir -p /usr/local/share/maps/style
```

Shapefile des limites administratives et bordures de côtes

```
cd /usr/local/share/maps/style
wget https://github.com/mapbox/osm-bright/archive/master.zip
wget http://data.openstreetmapdata.com/simplified-land-polygons-complete
    -3857.zip
wget http://data.openstreetmapdata.com/land-polygons-split-3857.zip
wget http://www.naturalearthdata.com/http
    //www.naturalearthdata.com/download/10m
    /cultural/ne_10m_populated_places_simple.zip
```

Déplacement dans le répertoire de travail

```
unzip '*.zip'
mkdir osm-bright-master/shp
mv land-polygons-split-3857 osm-bright-master/shp/
mv simplified-land-polygons-complete-3857 osm-bright-master/shp/
mv ne_10m_populated_places_simple osm-bright-master/shp/
```

Création des fichiers d'index des shapefile

```
cd osm-bright-master/shp/land-polygons-split-3857
shapeindex land_polygons.shp
cd ../simplified-land-polygons-complete-3857/
shapeindex simplified_land_polygons.shp
cd ../
```

B.2.2 Mise en place de OSM Bright

Il est nécessaire de spécifier l'emplacement des fichiers de données dans le fichier .mml contenant le projet cartoCSS. En effet, la génération de tuiles nécessite les géométries des lignes de côté et des différents pays.

Edition du fichier mml

```
cd osm-bright-master/osm-bright/
nano osm-bright.osm2pgsql.mml
```

Remplacement des lignes concernant les shp précédemment chargés (tels que les bordures naturelles) et spécification de leur emplacement. Ces lignes sont identifiables car elles pointent vers des fichiers .zip. Il est nécessaire d'ajouter le type de fichier ("shape").

land polygons

```
"file": "/usr/local/share/maps/style/osm-bright-master/shp/
    land-polygons-split-3857/land_polygons.shp",
"type": "shape"
```

simplified land polygons

```
"file": "/usr/local/share/maps/style/osm-bright-master/shp/
        simplified-land-polygons-complete-3857/simplified_land_polygons.shp",
"type": "shape",
```

ne 10m populated places simple

```
"file": "/usr/local/share/maps/style/osm-bright-master/shp/
        ne_10m_populated_places_simple/ne_10m_populated_places_simple.shp",
"type": "shape"
```

Dans la section "ne_places", il est également nécessaire de remplacer les lignes "srs" et "srs-name" par :

```
"srs": "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"
```

ATTENTION à vérifier le nom des fichiers : selon les versions, les fichiers de type "shapefile" n'ont pas exactement le même nom, par exemple le fichier anciennement nommé "ne 10 populated places" est devenu "ne 10 populated places **simple**". De plus, une vérification des shapefiles dans le fichier XML permet de s'assurer que mapnik ira rechercher les shp au bon emplacement.

B.2.3 Modification des paramètres de style de Mapnik

Il peut être intéressant d'éditer les paramètres par défaut de Mapnik, afin de lui permettre d'accéder correctement à la base de données ainsi qu'aux lignes de côtes.
Pour ce faire, il faut se baser sur les fichiers "templates" originaux que l'on copie avant de les éditer :

```
cd ~/src/mapnik-style/inc
cp fontset-settings.xml.inc.template fontset-settings.xml.inc
cp datasource-settings.xml.inc.template datasource-settings.xml.inc
cp settings.xml.inc.template settings.xml.inc
```

settings.xml.inc

Ce fichier contient les paramètres associés au style :

- **Symboles** qui seront associés aux différentes entités géographiques. Précise le répertoire qui les contient ;
- **Projection** qui sera utilisée dans la base de données postgres et qui est associée à osm2pgsql. Le code srs900913 étant équivalent au Web-Mercator (3857) et signifiant "Google" en écriture "leet" (1337) ;
- **Opérations spatiales d'inclusion (ST_Dwithin)** qui permet de définir le pas qui sépare deux entités géographiques distinctes. L'unité de ce pas dépend du système de coordonnées. Deux valeurs sont prévues selon qu'on travaille en Web-Mercator (900913 / 3857) ou en WGS84 ;
- **Répertoire des lignes de côtes** qui sont utilisées ;
- **Préfixe des tables OSM** qui sera utilisé afin de distinguer les tables OSM des autres tables au sein de la base. Par défaut, ce préfixe est "planet_osm".

```
<!--
Settings for symbols, the spatial reference of your postgis tables,
coastline shapefiles directory, and their prefix names.
-->

<!-- use 'symbols' unless you have moved the symbols directory -->
<!ENTITY symbols "symbols">

<!-- use the '&srs900913;' entity if you have called osm2pgsql without
special flags (or with -m); use '&srs4326;' if you have used -l -->
```

```
<!ENTITY osm2pgsql_projection "&srs900913;">

<!-- used for 'node_in_way' ST_DWithin spatial operations -->
<!-- Use 0.1 (meters) when your database is in 900913     -->
<!-- Use 0.000001 (degrees) when your database is in 4326 -->
<!ENTITY dwithin_900913 "0.1">
<!ENTITY dwithin_4326 "0.00001">
<!ENTITY dwithin_node_way "&dwithin_900913;">

<!-- use 'world_boundaries', which is the usual naming for the local
folder the coastline shapefiles are unzipped into -->
<!ENTITY world_boundaries "/usr/local/share/world_boundaries">

<!-- use 'planet_osm' unless you have customized your database table
prefix using the osm2pgsql 'prefix' flag -->
<!ENTITY prefix "planet_osm">
```

datasource-settings.xml.inc

Ce fichier de configuration contient les paramètres de la base de données : type (postgis), nom, host (adresse IP), port, utilisateur, mot de passe et extension spatiale (dans le système de coordonnées projetées : 3857).

```
<!--
Settings for your postgres setup.

Note: feel free to leave password, host, port, or use blank
-->

<Parameter name="type">postgis</Parameter>
<Parameter name="password">postgres</Parameter>
<Parameter name="host">localhost</Parameter>
<Parameter name="port">5432</Parameter>
<Parameter name="user">postgres</Parameter>
<Parameter name="dbname">gis</Parameter>
<!-- this should be 'false' if you are manually providing the 'extent' -->
<Parameter name="estimate_extent">false</Parameter>
<!-- manually provided extent in epsg 900913 for whole globe -->
<!-- providing this speeds up Mapnik database queries -->
<Parameter name="extent">-20037508,-19929239,20037508,19929239</Parameter>
```

Il est possible de commenter (selon la syntaxe des commentaires de type HTML) certains paramètres, afin de permettre Mapnik d'utiliser l'utilisateur unix local (ex : benjamin). Ce qui sera par contre évité dans le cas présent, car il n'est pas identique à celui de la base de données.

fontset-settings.xml.inc

Contient les définitions de polices de caractères ainsi que l'information pour les modifier. Il n'est, pour l'instant, pas modifié.

```
<!--Settings for Mapnik Fonts
To see the fonts you have installed with Mapnik do:
ls `python -c "import mapnik;print mapnik.fontscollectionpath"`-->

<FontSet name="book-fonts">
  <Font face-name="DejaVu Sans Book" />
```

```
  <Font face-name="unifont Medium" />
</FontSet>
<FontSet name="bold-fonts">
  <Font face-name="DejaVu Sans Bold" />
  <Font face-name="unifont Medium" />
</FontSet>
<FontSet name="oblique-fonts">
  <Font face-name="DejaVu Sans Oblique" />
  <Font face-name="unifont Medium" />
</FontSet>
```

B.2.4 Compilation de la feuille de style

Avant de pouvoir exploiter le style avec Mapnik, il est nécessaire de le convertir en un fichier xml conforme aux attentes de Mapnik. Pour ce faire le compilateur "carto" est utilisé en ligne de commande après avoir exécuté le script Python de configuration préalable propre à OSM Bright.

configure.py

Ce script a pour objectif de *récolter les variables propres à l'environnement de travail* tels que : le nom de la base de données, les ID, les répertoires contenant les shapefiles, etc.

Copie du fichier configure.py original (configure.py.sample)

```
cp configure.py.sample configure.py
```

Edition dans un éditeur de texte

```
nano configure.py
```

Que l'on édite en modifiant la ligne pointant vers le répertoire du projet (fichier .mml) Mapbox /Documents/Mapbox/project en :

```
/usr/local/share/maps/style
```

Il en va de même pour les paramètres de la base de données :

```
config["postgis"]["host"]     = "localhost"
config["postgis"]["port"]     = "5432"
config["postgis"]["dbname"]   = "gis"
config["postgis"]["user"]     = "postgres"
config["postgis"]["password"] = "postgres"
```

make.py

Cet autre script python a pour rôle de *créer un répertoire (OSMBright) qui contiendra le nécessaire au rendu* (style en .mml ou .xml, palette de couleur, symboles...).
Édition du fichier make.py :

```
defaultconfig["postgis"]["host"]     = "localhost"
defaultconfig["postgis"]["port"]     = "5432"
defaultconfig["postgis"]["dbname"]   = "gis"
defaultconfig["postgis"]["user"]     = "postgres"
defaultconfig["postgis"]["password"] = "postgres"
```

On peut maintenant l'exécuter :

```
./make.py
```

utilitaire carto

```
cd ../OSMBright/
carto project.mml > OSMBright.xml
```

Nous obtenons donc le fichier de style XML suivant :

```
/usr/local/share/maps/style/OSMBright/OSMBright.xml
```

ATTENTION que cet utilitaire en ligne de commande ne peut s'exécuter directement dans le répertoire /usr/local/share/maps/style/OSMBright/ , même avec les droits de super-utilisateur. Il est donc nécessaire d'effectuer la manipulation dans un répertoire s'y prêtant pour ensuite déplacer le fichier .xml dans /usr/local/share/maps/style/OSMBright/ :

```
sudo mv repertoire-de-travail-provisoir/OSMBright.xml
        /usr/local/share/maps/style/OSMBright/OSMBright.xml
```

B.3 Configuration du serveur web

B.3.1 Configuration de Renderd

Il est nécessaire de modifier les paramètres (en supprimant le; qui relègue la ligne au titre de commentaire) du daemon Renderd, afin de pourvoir fournir des tuiles au serveur Apache. Le fichier de configuration (renderd.conf) local et pour cet "host" (usr) se trouve dans le répertoire /usr/local/etc qui est spécifiquement créé à cet effet.

Fichier renderd.conf

Édition du fichier de configuration renderd.conf

```
cd /usr/local/etc
sudo gedit renderd.conf
```

Ainsi, la section [renderd] devient :

```
[renderd]
socketname=/var/run/renderd/renderd.sock
num_threads=4
tile_dir=/var/lib/mod_tile
stats_file=/var/run/renderd/renderd.stats
```

De même, la section [mapnik] devient :

```
[mapnik]
plugins_dir=/usr/local/lib/mapnik/input
font_dir=/usr/share/fonts/truetype/ttf-dejavu
font_dir_recurse=1
```

Enfin, la section [default] qui spécifie le répertoire des tuiles (/osm_tiles/), l'hôte, la taille des tuiles et le fichier de style XML :

```
[default]
URI=/osm_tiles/
TILEDIR=/var/lib/mod_tile
XML=/usr/local/share/maps/style/OSMBright/OSMBright.xml
HOST=localhost
TILESIZE=256
ATTRIBUTION=&copy;<a href=\"http://www.openstreetmap.org/\">OpenStreetMap</a>
```

Module Apache Mod_Tile

Création des répertoires accessibles à l'utilisateur "postgres" de la base de données :

```
sudo mkdir /var/run/renderd
sudo chown username /var/run/renderd
sudo mkdir /var/lib/mod_tile
sudo chown username /var/lib/mod_tile
```

B.3.2 Configuration de Mod_Tile

Fichier mod_tile.load

Création du fichier

```
cd /etc/apache2/mods-available
sudo touch mod_tile.load
```

Ensuite on édite ce même fichier pour y ajouter la ligne

```
LoadModule tile_module /usr/lib/apache2/modules/mod_tile.so
```

Ensuite, on indique à Apache d'utiliser ce module :

```
sudo a2enmod tile
```

fichier 000-default.conf

Ce fichier de configuration est à modifier au sein du serveur Apache, afin de charger Mod_Tile. Il faut donc y spécifier l'emplacement du fichier de config de Renderd ou encore les différents timeout.

```
cd /etc/apache2/sites-enabled/
sudo gedit 000-default.conf
```

Il est nécessaire d'ajouter les lignes suivantes après <?/VirtualHost> :

```
LoadTileConfigFile /usr/local/etc/renderd.conf
ModTileRenderdSocketName /var/run/renderd/renderd.sock
# Timeout before giving up for a tile to be rendered
ModTileRequestTimeout 0
# Timeout before giving up for a tile to be rendered that is otherwise missing
ModTileMissingRequestTimeout 300
```

Ce dernier timeout est à définir selon les capacités du serveur. Dans le cas présent, avec seulement 4 G de RAM, il a été fixé à 300 secondes. Ceci peut paraître excessif pour un chargement de tuiles mais la question des performances n'est pas encore abordée dans le cadre de ce prototype.
Ces changements ne prendront effet qu'après un redémarrage :

```
service apache2 restart
```

B.4 Chargement des données OSM dans le serveur

B.4.1 Obtention des données OSM

Commande wget et protocole HTTP

Chargement des données européennes :

```
mkdir -p ~/osm-EU
cd ~/osm-EU
wget http://download.geofabrik.de/europe-latest.osm.pbf.md5
wget http://download.geofabrik.de/europe-latest.osm.pbf
md5sum -c europe-latest.osm.pbf.md5
```

Ou, plus modestement comme lors de ce prototype, à l'échelle de la Belgique :

```
mkdir -p ~/osm-BE
cd ~/osm-BE
wget http://download.geofabrik.de/europe/belgium-latest.osm.pbf.md5
wget http://download.geofabrik.de/europe/belgium-latest.osm.pbf
md5sum -c belgium-latest.osm.pbf.md5
```

B.4.2 Import des données dans la base

Modification des droits et accès

Afin d'autoriser les accès à la base de données aux autres applications (principalement Mapnik), il faut modifier les accès autorisés au sein des fichiers de configuration de la base de données. Ces fichiers sont "pg_hba.conf" et "postgresql.conf" pour une base de données PostgreSQL.

Fichier pg_hba.conf

Localisation du fichier par le biais de postgresql en lignes de commande :

```
sudo -i -u postgres
```

```
postgres@(...) SHOW hba_file;
```

Edition

```
sudo gedit /etc/postgresql/9.3/main/pg_hba.conf
```

```
# IPv4 local connections:
host    all    all    127.0.0.1/32    md5
host    all    all    192.168.0.4/32    trust
```

Le premier correspondant aux accès depuis la machine en local (127.0.0.1 = localhost) alors que la seconde correspond à un autre ordinateur sur le même réseau (192.168.0.X) qui est utilisé en tant que client extérieur via QGIS lors des manipulations. Le paramètre "md5" spécifie qu'un mot de passe sera demandé alors que "trust" fait confiance et autorise les accès. Il n'est pas idéal/prudent de recourir au paramètres trust, toutefois il a été utilisé dans le cas présent, afin de simplifier la démarche. Chaque machine étant éteinte après l'utilisation. Il est évident que dans le cas d'un serveur professionnel le paramètre "md5" est vivement recommandé.

Fichier postgresql.conf

Localisation à nouveau par le biais de l'utilisateur "postgres" de postgresql en ligne de commande :

```
postgres@(...) SHOW config_file;
```

Edition du fichier texte :

```
sudo gedit /etc/postgresql/9.3/main/postgresql.conf
```

Edition de la ligne "listen_addresses" en supprimant le # (commentaire) et modifiant la valeur 'localhost' en '*' pour **accorder toutes les connections**

```
#------------------------------------------------------------------------------
# CONNECTIONS AND AUTHENTICATION
#------------------------------------------------------------------------------

# - Connection Settings -

listen_addresses = '*'          # what IP address(es) to listen on;
```

Déconnexion puis redémarrage de postgresql :

```
postgres@(...) logout
```

```
benjamin@(...) sudo /etc/init.d/postgresql restart
```

Outils de conversion osm2pgsql

Cet outil en ligne de commande permet l'insertion des données OSM au sein de la base de données "gis" précédemment créée en récupérant les données du fichier ".osm.pbf".

```
osm2pgsql --slim -d gis ~/osm-BE/belgium-latest.osm.pbf
```

-d spécifie le nom de la base de données, –slim active le mode "slim" où les données ne sont pas stockées dans la RAM lors du transfert mais dans des "tracking tables".
L'ensemble des 44 paramètres possibles à cette ligne de commande figurent dans le dépot github du programme[38].

C Script python d'exécution de rendu planimétrique avec Mapnik

En utilisant les données OSM avec le fichier de style XML associé (osm.xml), il est possible de générer directement des images par le biais d'un script python faisant appel à la bibliothèque mapnik.
Ci-après le script "mapnik_tiles_v02.py" :

38. https://github.com/openstreetmap/osm2pgsql/blob/master/docs/usage.md

```
#!/usr/bin/env python
#-*- coding: UTF-8 -*-
import os, ogr, glob, mapnik, psycopg2

"""             REPERTOIRES                 """
print '-Repertoires de travail:'
path_stylesheets =  "/usr/local/share/maps/style/OSMBright"
path_results = os.path.join(os.path.expanduser('~'), 'Mapnik_results_v02')
print '\tPath stylesheets:\t' + path_stylesheets;
print '\tPath results:\t' + path_results;

"""             ENTREE DES PARAMETRES   """
#Image
z = 10
imgx = 500 * z
imgy = 500 * z
map_uri = str(path_results +'/tile_test.png')
print "-Dimensions des tuiles: \n\tx = " + str(imgx) + " pixels \n\ty = "
        + str(imgy) + " pixels\n\turi de l image:" + map_uri

#Projection mercator
merc = mapnik.Projection('+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0
        +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null
        +no_defs +over')
print "-Projection: \n\t mercator spherique"

#Systeme de coord lat/lon par rapport a WGS84 (EPSG: 4326)
longlat = mapnik.Projection('+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs')

#Bounding box
bounds = (5.4, 50.0, 5.8, 50.8) #Liege
if hasattr(mapnik,'Box2d'):                 # creation de la bounding box
        bbox = mapnik.Box2d(*bounds)
else:
        bbox = mapnik.Envelope(*bounds)
transform = mapnik.ProjTransform(longlat,merc)
        #Transformation long-lat --> Web-Mercator
merc_bbox = transform.forward(bbox)      #bbox Belgique en web-mercator

"""             DEFINITION DU STYLE             """
os.chdir(path_stylesheets)      # Placement dans le repertoire stylesheets
retval = os.getcwd()
mapfile = "OSMBright.xml"
print"-Import du fichier de style:\t"+ mapfile

"""             CREATION DE LA MAP              """
m = mapnik.Map(imgx,imgy)       # creation d'une instance m de la classe Map
mapnik.load_map(m,mapfile)
m.srs = merc.params()           # projection en web-mercator
m.zoom_to_box(merc_bbox)        # Ajustement du zoom sur la bbox

"""             GENERATION DE L IMAGE   """
os.chdir(path_results);         # Placement dans le repertoire des resultats
im = mapnik.Image(imgx,imgy)
mapnik.render(m, im)
im.save(map_uri, 'png')         # Rendu possible aussi en svg, pdf...
```

D Déploiement du serveur de tuiles

D.1 Test manuel depuis le terminal

D.1.1 Terminal 1

création du "pid" de Renderd dans /var/run

Cette intervention permet de rendre le process "renderd" accessible/valide aux yeux des autres process. De plus, ces fichiers sont montés dans la RAM par l'OS.

```
sudo mkdir /var/run/renderd
```

Changement de propriétaire pour renderd.pi

On attribue les droits à l'utilisateur associé à la base de données postgresql (benjamin–> postgres).

```
sudo chown postgres /var/run/renderd
sudo chown postgres /var/lib/mod_tile
```

Lancement de renderd en arrière-plan

On lance le daemon Renderd qui permet de réaliser les tuiles OSM :

```
sudo renderd -f -c /usr/local/etc/renderd.conf
```

On ajoute les paramètres -f (foreground) pour que l'opération s'exécute en arrière-plan (tant que le terminal est ouvert) et -c pour spécifier l'emplacement du fichier de configuration (renderd.conf) précédemment créé.

D.1.2 Terminal 2

Parallèlement à Renderd, on (re)démarre le serveur web Apache dans un second terminal :

redémarrage d'apache2

```
sudo /etc/init.d/apache2 restart
```

D.1.3 Test d'affichage

Navigation vers la page

```
http://localhost/osm_tiles/0/0/0.png
```

Cette page affiche les limites des pays avec le zoom minimum.

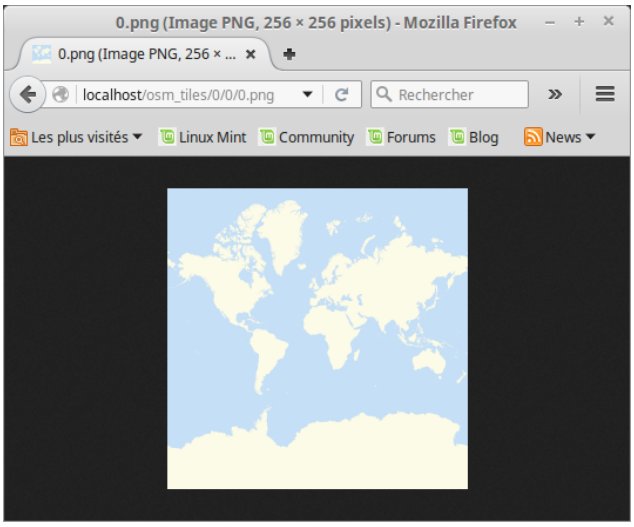

FIGURE 43 – Page http ://localhost/osm_tiles/0/0/0.png

D.2 Configuration du serveur pour un fonctionnement automatique

Il est possible de lancer automatiquement Renderd au démarrage en tant que daemon.

```
sudo cp ~/src/mod_tile/debian/renderd.init /etc/init.d/renderd
sudo chmod u+x /etc/init.d/renderd
```

Et ensuite on édite le fichier "/etc/init.d/renderd" en tant que root :

```
sudo gedit /etc/init.d/renderd
```

Pour y modifier les lignes suivantes :

```
DAEMON=/usr/local/bin/$NAME
DAEMON_ARGS="-c /usr/local/etc/renderd.conf"
(...)
RUNASUSER=postgres
```

Démarrage de Renderd

```
sudo /etc/init.d/renderd start
```

Arrêt de Renderd

```
sudo /etc/init.d/renderd stop
```

Démarrage automatique

Ajout d'un lien vers le répertoire de démarrage interactif :

```
sudo ln -s /etc/init.d/renderd /etc/rc2.d/S20renderd
```

E Classification des tags OSM pour des fonds de cartes aéronautiques

keys	tags	VFR	VFRn	IFR
aerialway	cable_car	1	1	
aerialway	gondola	1	1	
aerialway	chair_lift	1	1	
aerialway	mixed_lift	1	1	
aerialway	drag_lift	1	1	
aerialway	t_bar	1	1	
aerialway	j_bar	1	1	
aerialway	platter	1	1	
aerialway	rope_tow	1		
aerialway	zip_line	1	1	
aerialway	pylon	1	1	
aerialway	station	1	1	
aeroway	terminal	1	1	1
amenity	ferry_terminal	1		
amenity	parking			
amenity	prison	1		
boundary	national_park			
building	cathedral	1		
building	church	1		
building	mosque	1		
building	temple	1		
building	synagogue	1		
building	hospital	1		
building	stadium	1	1	
building	train_station	1		
highway	motorway	1	1	
highway	trunk	1		
highway	primary	1		
highway	secondary	1		
highway	tertiary	1		
highway	motorway_link	1		
highway	trunk_link	1		
highway	primary_link	1		
highway	secondary_link	1		
highway	tertiary_link	1		
highway	raceway	1		
highway	rest_area	1		
historic	monument	1		
landuse	forest	1	1	
landuse	reservoir	1	1	
landuse	quarry	1		
leisure	golf_course	1		
leisure	marina	1		
leisure	stadium	1		
leisure	track	1		

TABLE 10 – Classification des tags OSM pour des fonds de cartes aéronautiques

keys	tags	VFR	VFRn	IFR
man_made	lighthouse	1	1	
man_made	observatory	1		
man_made	offshore_platform	1		
man_made	petroleum_well	1		
man_made	telescope	1		
man_made	water_tower	1		
man_made	windmill	1		
natural	wood	1	1	
natural	tree	1		
natural	tree_row	1		
natural	water	1	1	
natural	volcano	1		
place	country	1	1	1
place	state	1	1	1
place	city	1	1	
place	town	1	1	
power	line	1	1	
power	minor_line	1	1	
power	pole	1	1	
power	portal	1	1	
power	tower	1	1	
railway	monorail	1		
railway	rail	1		
railway	station	1		
landuse	railway	1		
route	ferry	1	1	1
sport	baseball	1	1	
sport	american_football	1	1	
sport	athletics	1	1	
sport	australian_football	1	1	
sport	canadian_football	1	1	
sport	cricket	1	1	
sport	croquet	1		
sport	karting	1		
sport	rugby_league	1	1	
sport	rugby_union	1	1	
sport	running	1		
sport	soccer	1	1	
tourism	theme_park	1		
tourism	zoo	1		
waterway	river	1		
waterway	riverbank	1		
waterway	stream	1		
waterway	canal	1		
waterway	dock	1		
waterway	dam	1		
waterway	weir	1		
waterway	waterfall	1		

TABLE 11 – Classification des tags OSM pour des fonds de cartes aéronautiques (suite)

F Découpage en tuiles selon les niveaux de zoom

Niveau de zoom	Colonnes	Lignes	Tuiles	Degrés/pixel
1	2	1	2	0,3515625000
2	4	2	8	0,1757812500
3	8	4	32	0,0878906250
4	16	8	128	0,0439453125
5	32	16	512	0,0219726563
6	64	32	2048	0,0109863281
7	128	64	8192	0,0054931641
8	256	128	32768	0,0027465820
9	512	256	131072	0,0013732910
10	1024	512	524288	0,0006866455
11	2048	1024	2097152	0,0003433228
12	4096	2048	8388608	0,0001716614
13	8192	4096	33554432	0,0000858307
14	16384	8192	134217728	0,0000429153
15	32768	16384	536870912	0,0000214577
16	65536	32768	2147483648	0,0000107288
17	131072	65536	8589934592	0,0000053644
18	262144	131072	34359738368	0,0000026822
19	524288	262144	137438953472	0,0000013411
20	1048576	524288	549755813888	0,0000006706

TABLE 12 – Découpage en tuiles (512x512 pixels) selon les niveaux de zoom.

G Script python de génération des polygones de Voronoï

```
import ogr , os, glob, sqlite3, re, math
import numpy as np


#connection db
conn = sqlite3.connect("metar.sqlite")
conn.enable_load_extension(True)
conn.execute("SELECT load_extension('mod_spatialite')")
cur = conn.cursor()
conn.text_factory = str
cur.execute("""DROP TABLE IF EXISTS 'm_voronoi'""")
cur.execute("""DROP TABLE IF EXISTS 'route'""")
cur.execute("""DROP TABLE IF EXISTS 'route_buffer'""")
cur.execute("""DROP TABLE IF EXISTS 'voronoi_fr'""")
cur.execute("""DROP TABLE IF EXISTS 'm_circle'""")
cur.execute("""DROP TABLE IF EXISTS 'm_final'""")

cur.execute("DELETE FROM 'metar' WHERE flight_rule IS NULL")
conn.commit()


#creation de la table m_voronoi
cur.execute("""CREATE TABLE m_voronoi (id INTEGER PRIMARY KEY)""")
conn.commit()
cur.execute("SELECT AddGeometryColumn(
        'm_voronoi', 'geometry', 4326, 'POLYGON', 'XY')")


#creation des polygones de voronoi
cur.execute("""SELECT AsText(CastToMultiPolygon(ST_VoronojDiagram(
                                ST_Collect(GEOMETRY))))FROM metar""")
conn.commit()
diag = cur.fetchall()
diag = str(diag).replace("[('MULTIPOLYGON(", "").replace(")',)]", "")
diag = diag.split("), ")


#recuperation des identifiants et regles de vol correspondantes
d_v = {}
result = {}


for i in range(0, len(diag)):
    dia = str(diag[i]).replace("))", ")").replace(")", "))")
    id = str(i+1)
    cur.execute("""INSERT INTO m_voronoi VALUES(
                """+id+""", ST_GeomFromText('POLYGON"""+dia+""" '))""")
    conn.commit()

    cur.execute("""
            SELECT m_voronoi.id
            FROM metar, m_voronoi
            WHERE ST_intersects(metar.GEOMETRY, m_voronoi.geometry)
                                AND m_voronoi.id='"""+id+"""'
             """)
    conn.commit()
    ogc_f = cur.fetchall()
```

```
    d_v[i] = ogc_f

    cur.execute("""
            SELECT ogc_fid
            FROM metar, m_voronoi
            WHERE ST_intersects(metar.GEOMETRY, m_voronoi.geometry)
                                AND m_voronoi.id='"""+id+"""'
            """)
    conn.commit()
    ogc_fid = cur.fetchall()

    ogc_f = str(ogc_fid).split(",)")
    ogc_f = str(ogc_f[0]).replace("[(", "")
    ogc_f = str(ogc_f).replace("", "")


    cur.execute("""
            SELECT flight_rule, longitude, latitude, icao
            FROM metar
            WHERE ogc_fid = '"""+ogc_f+"""'
            """)
    conn.commit()
    data = cur.fetchall()
    result[i] = data


#ajout colonnes a la table m_voronoi
cur.execute("ALTER TABLE m_voronoi ADD ogc_fid FLOAT")
cur.execute("ALTER TABLE m_voronoi ADD fr VARCHAR(254)")
cur.execute("ALTER TABLE m_voronoi ADD latitude FLOAT")
cur.execute("ALTER TABLE m_voronoi ADD longitude FLOAT")
cur.execute("ALTER TABLE m_voronoi ADD ICAO VARCHAR(254)")
conn.commit()


#remplissage de la table m_voronoi
for i in range(0, len(diag)):
    id = str(i+1)

    ogc_f= d_v[i]
    ogc_f = str(ogc_f)
    ogc_f = str(ogc_f).replace("'", "")
    ogc_f1 = str(ogc_f).split(",)")
    ogc_f1 = str(ogc_f1[0]).replace("[(", "")

    data = result[i]
    data_s = str(data).split(",")

    f_rule = data_s[0]
    f_rule = str(f_rule)
    f_rule = str(f_rule).replace("'", "").replace("[(", "")

    latitude = data_s[2]
    latitude = str(latitude)
    latitude = str(latitude).replace("'", "").replace("[(", "")

    longitude = data_s[1]
    longitude = str(longitude)
    longitude = str(longitude).replace("'", "").replace("[(", "")
```

```
    icao = data_s[3]
    icao = str(icao)
    icao = str(icao).replace("'", "").replace("[(", "").replace(")]", "")

    cur.execute("""UPDATE m_voronoi SET ogc_fid = '"""+ogc_fid+"""'
                WHERE id = """+str(id)+""" """)
    cur.execute("""UPDATE m_voronoi SET fr = '"""+f_rule+"""'
                WHERE id = """+str(id)+""" """)
    cur.execute("""UPDATE m_voronoi SET latitude = '"""+latitude+"""'
                WHERE id = """+str(id)+""" """)
    cur.execute("""UPDATE m_voronoi SET longitude = '"""+longitude+"""'
                WHERE id = """+str(id)+""" """)
    cur.execute("""UPDATE m_voronoi SET ICAO = '"""+icao+"""'
                WHERE id = """+str(id)+""" """)
    conn.commit()
```

H Code HTML de la page web contenant la carte de l'interface client

```
<html>
<head>
        <meta charset=utf-8 />
        <title>OSM Tiles</title>
        <meta name='viewport'
        content='initial-scale=1,maximum-scale=1,user-scalable=no' />
        <script src='https://api.mapbox.com/mapbox.js/v3.0.1/mapbox.js'></script>
        <link  href='https://api.mapbox.com/mapbox.js/v3.0.1/mapbox.css'
        rel='stylesheet' />

<!--PLUGIN LEAFLET-ARC FOR GREAT CIRCLES-->
        <script type="text/javascript"
        src="../Leaflet.Arc-master/bin/leaflet-arc.min.js"></script>
<!--PLUGIN LEAFLET-AJAX-->
        <script type="text/javascript"
        src="../leaflet-ajax-master/dist/leaflet.ajax.js"></script>
<!--PLUGIN LEAFLET-CORRIDOR-->
        <script type="text/javascript"
        src="../leaflet-corridor-master/leaflet-corridor.js"></script>
<!--JQUERY-->
        <script type="text/javascript"
        src="jquery-3.2.0.js"></script>
<!--DRAWING TOOLS-->
        <script type="text/javascript"
        src="../Leaflet.draw-master/src/Leaflet.draw.js"></script>

        <script src="../Leaflet.draw-master/src/Leaflet.Draw.Event.js">
        </script>
        <script src="../Leaflet.draw-master/src/edit/handler/Edit.Poly.js">
        </script>
        <script src="../Leaflet.draw-master/src/edit/handler/Edit.SimpleShape.js">
        </script>
        <script src="../Leaflet.draw-master/src/edit/handler/Edit.Circle.js">
        </script>
        <script src="../Leaflet.draw-master/src/edit/handler/Edit.Rectangle.js">
        </script>
        <script src="../Leaflet.draw-master/src/edit/handler/Edit.Marker.js">
        </script>
        <script src="../Leaflet.draw-master/src/draw/handler/Draw.Feature.js">
        </script>
        <script src="../Leaflet.draw-master/src/draw/handler/Draw.Polyline.js">
        </script>
        <script src="../Leaflet.draw-master/src/draw/handler/Draw.Polygon.js">
        </script>
        <script src="../Leaflet.draw-master/src/draw/handler/Draw.SimpleShape.js">
        </script>
        <script src="../Leaflet.draw-master/src/draw/handler/Draw.Rectangle.js">
        </script>
        <script src="../Leaflet.draw-master/src/draw/handler/Draw.Circle.js">
        </script>
        <script src="../Leaflet.draw-master/src/draw/handler/Draw.Marker.js">
        </script>
        <script src="../Leaflet.draw-master/src/ext/TouchEvents.js">
        </script>
        <script src="../Leaflet.draw-master/src/ext/LatLngUtil.js">
```

```
        </script>
        <script src="../Leaflet.draw-master/src/ext/GeometryUtil.js">
        </script>
        <script src="../Leaflet.draw-master/src/ext/LineUtil.Intersect.js">
        </script>
        <script src="../Leaflet.draw-master/src/ext/Polyline.Intersect.js">
        </script>
        <script src="../Leaflet.draw-master/src/ext/Polygon.Intersect.js">
        </script>
        <script src="../Leaflet.draw-master/src/Control.Draw.js">
        </script>
        <script src="../Leaflet.draw-master/src/Tooltip.js">
        </script>
        <script src="../Leaflet.draw-master/src/Toolbar.js">
        </script>
        <script src="../Leaflet.draw-master/src/draw/DrawToolbar.js">
        </script>
        <script src="../Leaflet.draw-master/src/edit/EditToolbar.js">
        </script>
        <script src="../Leaflet.draw-master/src/edit/handler/EditToolbar.Edit.js">
        </script>
        <script src="../Leaflet.draw-master/src/edit/handler/EditToolbar.Delete.js">
        </script>

        <link rel="stylesheet"  href="../Leaflet.draw-master/src/leaflet.draw.css">
        </script>
<!--BUFFER-->
        <script src="../Leaflet.buffer-master/dist/leaflet.buffer.min.js">
        </script>
<!--POINT IN POLYGON -->
        <script src='https://unpkg.com/@mapbox/leaflet-pip@latest/leaflet-pip.js'>
        </script>
<!--EASY PRINT-->
        <script src="../leaflet-easyPrint-gh-pages/dist/leaflet.easyPrint.js">
        </script>

        <style>
                html { height: 100% }
                body { height: 100%; margin: 0; padding: 0;}
                #map{ height: 80% }
                .info
                {
                        padding: 6px 8px;
                        font: 14px/16px Arial, Helvetica, sans-serif;
                        background: white;
                        background: rgba(255, 255, 255, 0.8);
                        box-shadow: 0 0 15px rgba(0, 0, 0, 0.2);
                        border-radius: 5px;
                }
                .legend
                {
                        text-align: left;
                        line-height: 18px;
                        color: #555;
                }
                .legend i
                {
                        width: 18px;
                        height: 18px;
```

```
                        float: left;
                        margin-right: 8px;
                        opacity: 0.7;
                }
                .route-corridor
                {
                        color: #555;
                        stroke: #999;
                        stroke-opacity: 0.3;
                }
</style>
        </style>

</head>
<body>
        <div id="intersectdiv" >
        <div id='map'/>
        </div>

        <a href='#' id='geolocate' class='ui-button'>
        Centrer sur ma position</a>
        <br/>
        <center>

        <a href="javascript:void(getGreatCircle())">
        Draw trajectory (great circle)</a>
        <br/> <br/>
        <a href="javascript:void(getOverflown())">
        Get overflown Flight Rules</a>
        <br/> <br/>
        <a href="javascript:void(getGeometries())">
        Get geometries (GeoJSON)</a>
        <br/> <br/>
        <a href="javascript:void(getOverflownFlightRules())">
        Get overflown flight rules (GeoJSON)</a>
        <br/> <br/>

        <textarea id="pointsTextArea" rows="5" cols="70">
        </textarea>
        </center>
        <hr>

        <script>

        /*****************/
        /*** MAP ENVIR ***/
        /*****************/

        var inetAdr = "192.168.0.10";   // Local IP address
        L.mapbox.accessToken =
        'pk.eyJ1IjoiYmRlc3d5c2VuIiwiYSI6ImNpbXdjcjFqZzAwYjV2bmx5am4xZnEwMTEifQ.
        II4odtnyaceMh3NNKPhzkg';
        var geolocate = document.getElementById('geolocate');
        var map = L.map('map').setView([50.5,4.5], 5);
        L.control.scale({               // scale based on the center of the map
                position: 'bottomleft',
                imperial: false,                // miles
                metric: true                    // km
        }).addTo(map);
```

```
L.easyPrint({                    // Printing options
        title: 'Print map',
        position: 'topleft',
        elementsToHide: 'p, h2'
}).addTo(map);


var meterPerPixel = getMeterPerPixel(map);

/************************/
/*** STYLES DEFINITION ***/
/************************/
function getColor(fr)
{
        switch (fr)
        {
                case 'MVFR':
                        color= "#ff0000";
                        break;
                case 'VFR':
                        color= "#0000ff";
                        break;
                case 'IFR':
                        color= "#ff7800";
                        break;
                case 'LIFR':
                        color= "#458B00";
                        break;
        }
        return color;
}

function voronoiStyle(feature)
{
        return {
                weight: 1,
                opacity: 0.15,
                fillOpacity: 0.15,
                fillColor: getColor(feature.properties.fr),
                color: "#000000"
        };
}

/************************/
/*** LAYERS DEFINITION ***/
/************************/

        /*** MAPBOX TILES ***/
var mapbox_vfr = L.mapbox.styleLayer
('mapbox://styles/bdeswysen/ciyui44ut006s2sudlrgmrpau'),
        mapbox_vfr_nuit = L.mapbox.styleLayer
('mapbox://styles/bdeswysen/cj0nreueu000z2rnyknj3tiv0'),
        mapbox_ifr = L.mapbox.styleLayer
('mapbox://styles/bdeswysen/ciyui44ut006s2sudlrgmrpau');

        /*** LOCAL TILES ***/
// var local_vfr = L.tileLayer(inetAdr+'/osm_tiles/{z}/{x}/{y}.png');
```

```
        /*** METAR AND FLIGHT RULES LAYERS ***/
var startMetar =[{"type":"FeatureCollection","totalFeatures":0,"features":[],
"crs":null}];
var startVoronoi = [{"type":"FeatureCollection","totalFeatures":0,"features":[],
"crs":null}];
var metarLayer = new L.GeoJSON(startMetar, {onEachFeature:popUp});
var voronoiLayer = new L.GeoJSON(startVoronoi, {style: voronoiStyle} );

/*************************************/
/*** DISPLAY FONCTION FOR METAR DATA ***/
/*************************************/

function popUp(f,l)
{
        var out = [];
        if (f.properties)
        {
                for(key in f.properties)
                {
                        out.push("<b>"+key+"</b>: "+f.properties[key]);
                }
                l.bindPopup(out.join("<br />"));
        }
}

/*****************************************************/
/*** DISPLAY OF THE MAP ACCORDING TO THE ZOOM LEVEL ***/
/*****************************************************/

map.on('moveend', function()
{
        if(map.getZoom() >= 5)
        {
        /*** REQUEST TO GEOSERVER ***/
                var geoJsonUrl = 'http://'+inetAdr+':8080/geoserver/OSM/ows';
                var metarParameters =
                {
                        service: 'WFS',
                        version: '1.0.0',
                        request: 'getFeature',
                        typeName: 'OSM:m_metar',
                        // maxFeatures: 50,
                        outputFormat: 'application/json'
                };

                var voronoiParameters =
                {
                        service: 'WFS',
                        version: '1.0.0',
                        request: 'getFeature',
                        typeName: 'OSM:m_voronoi',
                        // maxFeatures: 50,
                        outputFormat: 'application/json'
                };

                var customParams =
                {
                        bbox: map.getBounds().toBBoxString()
                };
```

```
                        metarUrl=geoJsonUrl + L.Util.getParamString(L.Util.extend
                        (metarParameters, customParams));
                        voronoiUrl=geoJsonUrl + L.Util.getParamString(L.Util.extend
                        (voronoiParameters, customParams));

                                // METAR AJAX CALL
                        $.ajax(
                        {
                                url: metarUrl,
                                dataType: "json",
                                success: function(data)
                                {
                                        metarLayer.clearLayers();
                                        metarLayer.addData(data);
                                }
                        });
                                // VORONOI AJAX CALL
                        $.ajax(
                        {
                                url: voronoiUrl,
                                dataType: "json",
                                success: function(data)
                                {
                                        voronoiLayer.clearLayers();
                                        voronoiLayer.addData(data);
                                }
                        });
                }
                /*** REMOVE THE LAYER FOR LOW ZOOM LEVEL ***/
                else
                {
                        map.removeLayer(metarLayer);
                        map.removeLayer(voronoiLayer);
                }
        });


        /*************************/
        /*** LEGEND FLIGHT RULES ***/
        /*************************/

        var legendVoronoi = L.control({position: 'bottomright'});

                /*** CREATION OF THE LEGEND DIV ***/
        legendVoronoi.onAdd = function (map)
        {
                var div = L.DomUtil.create('div', 'info legend'),
                labels = ['<strong>Flight rules</strong>'],
                flightRules = ['VFR', 'MVFR', 'IFR', 'LIFR'];

                for (var i = 0; i < flightRules.length; i++)
                {
                        div.innerHTML +=
                                labels.push(
                                        '<i class="circle" style="background:'
-----------------------------------------+getColor(flightRules[i])+'"></i> '
                                        +(flightRules[i] ? flightRules[i] : '+'));
                }
```

```
        div.innerHTML = labels.join('<br>');
        return div;
};

        /*** LOCK SYSTEM BETWEEN THE 2 LAYERS USING THE SAME LEGEND ***/
var lockFR = false;
var lockFRO = false;

        /*** EVENTS ADDING THE LEGEND ***/
map.on('overlayadd', function (eventLayer)
{
        if (eventLayer.name === 'Flight Rules')
        {
                lockFR = true;
                legendVoronoi.addTo(map);
        }
        if (eventLayer.name === 'Flight Rules overflown')
        {
                lockFRO = true;
                legendVoronoi.addTo(map);
        }
});

        /*** EVENTS REMOVING THE LEGEND ***/
map.on('overlayremove', function (eventLayer)
{
        if (eventLayer.name === 'Flight Rules' )
        {
                if(lockFRO != true)
                {
                        this.removeControl(legendVoronoi);
                }
                lockFR = false;
        }
        if (eventLayer.name === 'Flight Rules overflown')
        {
                if(lockFR != true)
                {
                        this.removeControl(legendVoronoi);
                }
                lockFRO = false;
        }
});

/********************/
/*** DRAWN ITEMS  ***/
/********************/

var drawnItems = new L.FeatureGroup();
map.addLayer(drawnItems);

L.drawLocal.draw.toolbar.buttons.polygon = 'Draw geometries';

/********************/
/*** DRAW CONTROL ***/
/********************/

var drawControl = new L.Control.Draw(
{
```

```
        position: 'topleft',
        draw:
        {
                polyline: true,
                polygon: false,
                circle: false,
                marker: false,
                rectangle: false,
                metric: true
        },
        edit:
        {
                featureGroup: drawnItems,
                remove: true,
                buffer:
                {
                  replacePolylines: false,
                  separateBuffer: true
                }
        }
});
map.addControl(drawControl);

        /*** ITEMS CREATION ***/
map.on(L.Draw.Event.CREATED, function (e)
{
        var type = e.layerType,
                        layer = e.layer;
        if (type === 'marker')
        {
                layer.bindPopup('A popup!');
        }
        drawnItems.addLayer(layer);
});

        /*** ITEMS EDITION ***/
map.on(L.Draw.Event.EDITED, function (e)
{
        var layers = e.layers;
        var countOfEditedLayers = 0;
        layers.eachLayer(function (layer) {
                countOfEditedLayers++;
        });
        console.log("Edited " + countOfEditedLayers + " layers");
});

/************************************/
/*** CREATE GREAT CIRCLE TRAJECTORY ***/
/************************************/

var coordsGreatCircle = new Array;
var trajectoryLine = new L.Polyline(coordsGreatCircle);

var getGreatCircle = function()
{
console.log("GET GREAT CIRCLE");
drawnItems.eachLayer(function(layer)
{
var i = 0;              //point per layer counter
```

```
var pointLatLngs = layer.getLatLngs();
var nbLatLngs = pointLatLngs.length;
pointLatLngs.forEach(function(point)
{
coordsGreatCircle[i]= point;

if (0<i)        // Begin the arcs with min 2 vertices
{
        var dist = coordsGreatCircle[i-1]
        .distanceTo(coordsGreatCircle[i]);
        // get the dist on a great circle using the "Haversine formula"
        var density = 0.001;            // point per m
        var nbVertices = Math.round(density * dist);
        // nb vertices around to an integer number

        /*** DESIGN OF THE ARCS USING VERTICES OF THE DRAWN POLYLINE ***/
        greatCircle= L.Polyline.Arc(coordsGreatCircle[i-1],
        coordsGreatCircle[i],
        {
        vertices: nbVertices    // number of intermediate verticles per arc
        }
        );
                /*** GET THE INTERMEDIATE VERTICES ***/
        greatCircle.getLatLngs().forEach(function(point)
        {
                trajectoryLine.addLatLng(point);
        });
}
i++;
});
});
drawnItems.addLayer(trajectoryLine);
};

/**********************/
/*** OVERFLOWN METAR ***/
/**********************/

var voronoiOverflown = new L.FeatureGroup();

var getOverflown= function()
{

        console.log("GET_OVERFLOWN");
        voronoiOverflown.clearLayers();
                /*** LAYERS ***/
        drawnItems.eachLayer(function(layer)
        {
        var trajectoryLatLngs = layer.getLatLngs();
        i=0;
                /*** LATLNGS ***/
        trajectoryLatLngs.forEach(function(latlngs)
        {
                /*** IF LATLNGS ARE IN AN ARRAY ***/
        if( Object.prototype.toString.call( latlngs ) === '[object Array]' )
        {
        latlngs.forEach(function(point)
        {
                // array of the L.Polygon containing the point
```

```
                var pipResult= leafletPip.pointInLayer( point , voronoiLayer);
                pipResult.forEach(function(polygon)
                {
                        voronoiOverflown.addLayer(polygon).addTo(map);
                });
                i++;
        });
        }
                /*** LATLNG OBJECT ***/
        else
        {
        // array of the L.Polygon containing the point
        var pipResult= leafletPip.pointInLayer( latlngs , voronoiLayer);
        pipResult.forEach(function(polygon)
        {
                voronoiOverflown.addLayer(polygon).addTo(map);
        });

        i++;
        }
        });
        });
};


/*******************/
/*** TEXT EXPORT ***/
/*******************/

var getGeometries = function()
{
        console.log("GET_GEOMETRIES");
        var geomGeoJSON = drawnItems.toGeoJSON();
        var pointsTextArea = document.getElementById('pointsTextArea');
        pointsTextArea.innerHTML = JSON.stringify(geomGeoJSON);
};

var getOverflownFlightRules = function()
{
        console.log("GET_OVERFLOWN_FLIGHT_RULES");
        var ofrGeoJSON = voronoiOverflown.toGeoJSON();
        var pointsTextArea = document.getElementById('pointsTextArea');
        pointsTextArea.innerHTML = JSON.stringify(ofrGeoJSON);
};


/*********************/
/*** LAYER CONTROL ***/
/*********************/
var baseMaps =
{
        'Mapbox vfr': mapbox_vfr.addTo(map),
        'Mapbox vfr-nuit': mapbox_vfr_nuit,
        'Mapbox ifr': mapbox_ifr
        // 'Tuiles vfr': L.tileLayer(inetAdr+'/osm_tiles/{z}/{x}/{y}.png')
};
var overlays=
{
        "Metar": metarLayer,
```

```
                "Flight_Rules": voronoiLayer,
                "Flight_Rules_overflown": voronoiOverflown,
                "Trajectory_(drawn)": drawnItems
        };

        var lc=L.control.layers(baseMaps,overlays);
        lc.addTo(map);

        /**********************/
        /*** GEOLOCALISATION ***/
        /**********************/

        if (!navigator.geolocation)
        {
                geolocate.innerHTML = 'Geolocation is not available';
        }
        else
        {
                geolocate.onclick = function (e) {
                e.preventDefault();
                e.stopPropagation();
                map.locate();
                };
        }

                // Center on the user location using IP
        map.on('locationfound', function(e)
        {
                map.fitBounds(e.bounds);

        });
        map.on('locationerror', function()
        {
                geolocate.innerHTML = 'Position could not be found';
                geolocate.parentNode.removeChild(geolocate);
        });

        </script>

</body>
</html>
```

I Carte aéronautique relatives aux arrivées pour l'aéroport de Liège

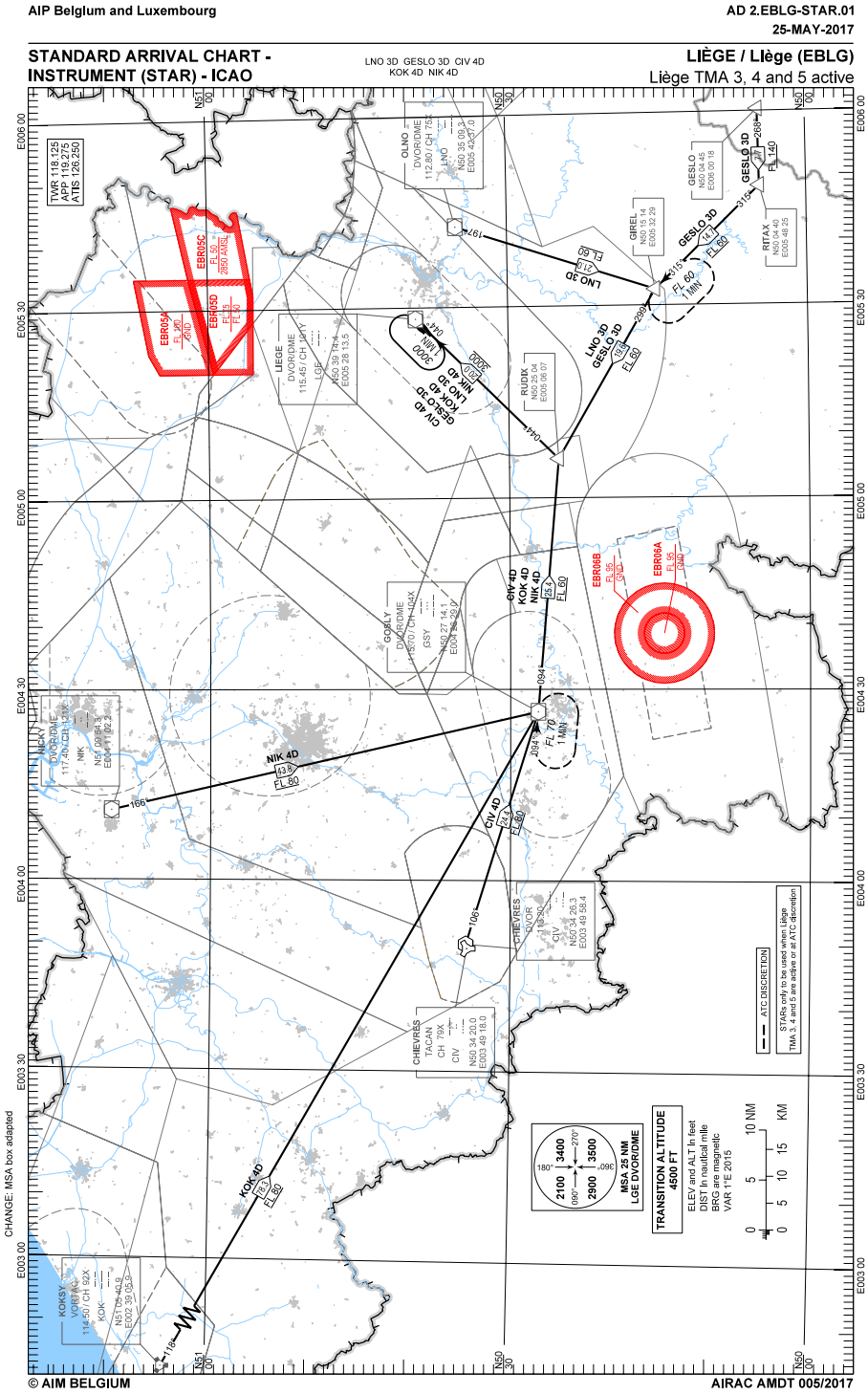

FIGURE 44 – Carte aéronautique relatives aux règles d'arrivée pour l'aéroport de Liège (EBLG). Source : données AIM (Eurocontrol)