Mise en place du SIG 3D des éléments aériens des lignes des Transports Publics Genevois

Joris 26/12/1991 Vidalenche

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

ÉCOLE SUPÉRIEURE DES GÉOMÈTRES ET TOPOGRAPHES

MÉMOIRE

présenté en vue d'obtenir

le DIPLÔME D'INGÉNIEUR CNAM

Spécialité : Géomètre et Topographe

par

Joris VIDALENCHE

Mise en place du SIG 3D des éléments aériens des lignes des Transports Publics Genevois

Soutenu le 08 juillet 2014

JURY

PRESIDENT : **Monsieur Stéphane DURAND**

MEMBRES : **Monsieur Jérôme HENRY, maître de stage**
Monsieur Vincent HABCHI, professeur référent
Monsieur Jean-Michel FOLLIN

Remerciements

Je souhaiterais remercier tout d'abord l'ensemble de l'équipe de HKD Géomatique SA, et plus particulièrement Messieurs Jérôme HENRY et Samuel DUNANT pour m'avoir offert l'opportunité de réaliser mon travail de fin d'études dans leur entreprise, pour la confiance qu'ils m'ont accordée et pour l'excellent suivi de mon activité dont ils ont fait preuve.

Je suis également très reconnaissant envers Monsieur Gaëtan GIVORD et l'équipe de saisie des lignes, Messieurs Camille DIZIAIN et Romain GERY, pour leur aide et leur patience qui m'ont été indispensables pour mener à bien ce projet.

D'une manière plus générale, merci à l'ensemble du bureau pour sa convivialité qui m'a permis d'effectuer mon travail dans d'excellentes conditions.

Je remercie également mon professeur référent, Monsieur Vincent HABCHI pour son suivi et sa disponibilité.

Enfin, merci à toutes les personnes qui m'ont soutenu durant ces quelques mois et qui auront aidé, de près comme de loin, à la réussite de ce TFE.

Liste des abréviations

CAO : Conception assistée par ordinateur
DAO : Dessin assisté par ordinateur
Esri® : Environmental systems research institute
LOD : Level of details
MLS : Mobile laser scanning
SIG : Système d'information géographique
SDK : Software development kit
SITG : Système d'information du territoire à Genève
TB : Trolleybus
TPG : Transports publics genevois
TW : Tramway

Glossaire

Framework : ensemble cohérent de composants logiciels servant à créer les fondations et les grandes lignes d'un logiciel (trad. litt. : « cadre de travail »).

Geoprocessor (ou Geoprocessing tool) **:** outil de manipulation de données spatiales.

Software development kit : ensemble d'outils d'aide à la programmation permettant de créer des applications de type défini (Android™ SDK pour les applications Android™, ArcObjects™ SDK pour les extensions ArcGIS®, etc.).

Standalone : une application logicielle est dite standalone quand il est possible de l'utiliser seule, indépendamment d'autres applications.

Symbologie : définition de la représentation des différentes entités d'un SIG.

Topologie : branche des mathématiques concernant l'étude des relations de position.

Trolleybus : véhicule électrique utilisant les voies publiques, roulant sur pneumatiques, alimenté par une ligne aérienne bifilaire.

Table des matières

Introduction

La Suisse (ou officiellement, la Confédération suisse) est organisée en 26 cantons, possédant chacun ses propres lois. Chaque canton est ensuite subdivisé en communes. Situé à l'extrémité ouest du lac Léman, le canton de Genève regroupe 45 communes. Il est l'un des plus peuplés (460 534 habitants en 2011 selon l'Office Fédéral Suisse de la Statistique), malgré le fait qu'il soit également l'un des plus petits avec ses 280 kilomètres carrés environ. La ville de Genève et ses alentours attirent avec force les travailleurs du canton de Vaud, au nord, mais également les travailleurs français de l'Ain et de la Haute-Savoie.

Fondée en 1957, HKD Géomatique SA est aujourd'hui basée à Onex, dans le canton de Genève. Depuis 2012, elle possède également le bureau Peitrequin Olivier SA situé à Nyon, sur le canton de Vaud. Le bureau d'Onex, au sein duquel j'ai travaillé durant ces 5 mois, emploie actuellement 18 personnes, ce qui lui permet d'être présent sur divers projets de grande ampleur, tel que la modélisation 3D des lignes des TPG (transports publics genevois).

Les TPG souhaitant disposer d'une base de données afin de gérer les éléments aériens des lignes de tramways et de trolleybus de Genève, HKD Géomatique a été mandaté pour réaliser le lever des voies du canton traversées par ces lignes et mettre en place une base de données sous forme de SIG (système d'information géographique) 3D. Ainsi, le bureau a réalisé en janvier 2013 le relevé 3D de soixante-dix kilomètres de corps de rues par MLS (mobile laser scanning), puis a modélisé les lignes aériennes des TPG sous forme d'entités ponctuelles (aiguillages[1], antennes, boitiers, contrepoids, croisements, feux d'aiguillages, isolateurs, isolateurs de haubans, panneaux, parafoudres, scellements et sectionneurs) et linéaires (câbles électriques, consoles, feeders, haubans, lignes de contact, pinces de courbe, protections, suspensions et tubes).

À mon arrivée chez HKD Géomatique, la base de données arrivait à la fin de cette phase de saisie. Celle-ci consiste en la modélisation des éléments aériens en 3D au moyen du logiciel Trimble® Trident-3D Analyst®, en utilisant comme support le nuage de points et les photographies acquis par la mission de MLS. Cette étape va ainsi poser une première problématique que l'on tentera de résoudre durant ce TFE : l'accrochage des éléments se faisant sur le nuage de points et non sur les entités elles-mêmes, les erreurs de topologie sont inévitables.

Par ailleurs, la symbologie utilisée lors de cette modélisation reste basique et n'est pas adaptée à l'exploitation de la base de données en tant que SIG, ce qui constitue notre deuxième problématique : concevoir une symbologie 3D en adéquation avec l'utilisation que l'on fera de cette géodatabase.

Par la suite, ce SIG sera finalement livré aux TPG, ce qui pose de nouvelles problématiques. Premièrement, ceux-ci ayant l'habitude d'utiliser des logiciels de CAO (conception assistée par ordinateur), ils ont besoin que le passage du monde du SIG vers celui du DAO (dessin assisté par ordinateur) soit supporté, sous la forme d'un outil d'export. Toujours dans les outils d'export, il leur serait également utile d'avoir à disposition la possibilité d'exporter des vues en PDF. Il conviendra donc d'intégrer ces deux outils dans notre SIG.

[1]Dans le langage courant, le terme « aiguillage » est utilisé à la place de l'expression officielle « appareil de voie ». Dans notre cas, on s'intéresse au dispositif situé sur les lignes de contact et non au sol, et qui porte bien le nom d'« aiguillage ».

Enfin, la base de données livrée devra être régulièrement mise à jour, au fur et à mesure des opérations d'entretien du réseau. Cependant, cette tâche reviendra aux techniciens des TPG, peu rompus à cette pratique. Il est donc indispensable de leur proposer un outil simple pour la réaliser, ce qui représente le quatrième et dernier objectif de ce TFE.

Ce mémoire présentera ainsi les recherches effectuées sur ces différents travaux. Étant bien distincts les uns des autres, ils constitueront chacun une des quatre parties de cet écrit.

I Création d'un outil de contrôles topologiques en 3D

En SIG, le respect de la topologie est un critère de qualité majeur. En effet, un SIG a pour vocation de renseigner l'utilisateur sur les caractéristiques intrinsèques de l'objet étudié (attributs, position absolue), mais aussi sur les relations qui existent avec son voisinage. Il est donc important de contrôler, et de corriger, les relations topologiques entre les diverses entités du produit. Dans notre cas, il faudra s'assurer, par exemple, que chaque panneau soit bien suspendu à un hauban et ne « lévite » pas. Cependant, il faut bien comprendre que détecter et corriger ces erreurs manuellement sur les soixante-dix kilomètres de lignes relevées représenterait un travail colossal, d'autant plus qu'il est à effectuer selon les 3 dimensions. Il convient donc d'automatiser le processus. Pour donner un aperçu de la quantité d'information présente dans notre base de données, les tableaux 1 et 2 présentent quelques statistiques issues de celle-ci.

Éléments ponctuels	***Nombre***	***Pourcentage***
Isolateurs de haubans	20 824	54,06
Isolateurs	13 384	34,75
Scellements	2 550	6,62
Panneaux	810	2,10
Croisements	311	0,81
Sectionneurs	210	0,55
Aiguillages	145	0,38
Contrepoids	93	0,24
Feux	74	0,19
Boitiers	55	0,14
Parafoudres	33	0,09
Antennes	28	0,07
Total	**38 517**	**100,00**

Tableau 1 : Statistiques - éléments ponctuels

Éléments linéaires	***Longueur (km)***	***Pourcentage***
Lignes de contact	207,318	48,42
Haubans	152,095	35,52
Feeders	35,366	8,26
Câbles électriques	10,628	2,48
Tubes	9,157	2,14
Suspensions	5,382	1,26
Pinces de courbes	5,121	1,20
Consoles	2,751	0,64
Protections	0,363	0,08
Total câbles	**410,790**	**95,94**
Total autres	**17,392**	**4,06**
Total	**428,182**	**100,00**

Tableau 2 : Statistiques - éléments linéaires

I.1 Etat de l'art

Avant de se lancer dans cette automatisation, il faut déjà se pencher sur les outils existants pour étudier les relations topologiques entre deux entités.

La base de données sera traitée sous ArcGIS®. On peut donc se renseigner sur les solutions déjà proposées par Esri®. Comme l'a déjà remarqué Bernard LACHANCE (2005), ArcGIS® dispose d'outils de topologie uniquement en 2D. Selon lui, bien que ces outils restent fonctionnels dans un environnement 3D, ils travaillent toujours en 2D, comme s'ils plaquaient les entités sur le plan (O,x,y) avant de les traiter.

Ainsi, Gaëtan GIVORD, ingénieur chez HKD Géomatique, a déjà proposé une piste de recherche. Cette dernière consiste à utiliser ces outils 2D sur la base de données telle-quelle, puis de lui faire subir une rotation autour des axes (O,x) ou (O,y) de manière à réutiliser les outils de contrôles topologiques 2D dans un plan vertical (et enfin réaliser la rotation inverse pour revenir dans le système d'origine). Cette solution est certes efficace, mais oblige à transformer l'ensemble des données, même celles qui ne devront subir aucune correction. Cette méthode est donc lourde à mettre en œuvre, et peut présenter des risques pour le positionnement des objets suite aux transformations effectuées (chaque rotation constitue un nouveau calcul et peut entrainer une erreur d'arrondi). Il met ainsi en évidence le fait qu'il est préférable de réaliser les contrôles directement dans un espace en trois dimensions.

Passé cette première approche, il apparait a priori que de tels outils n'existent pas (y compris avec d'autres logiciels), les SIG étant traditionnellement en deux dimensions. Ils devront donc être entièrement développés. A noter qu'ils seront relativement simples : la base ne contient que des entités ponctuelles et linéaires, et les relations topologiques seront uniquement du type « se touchent » / « ne se touchent pas ».

Etant arrivé à des conclusions similaires, Florian GANDOR, lors d'un précédent stage chez HKD Géomatique, a cherché à résoudre le problème via le langage Python™ et le module pyshp, permettant de traiter des shapefiles sous ce langage de programmation. Pour ce faire, il a développé un script qui procède par comparaison de coordonnées : pour deux entités données, l'une est définie comme fixe, tandis que l'autre est considérée comme mobile. La distance qui les sépare est alors calculée à partir de leurs coordonnées. En dessous d'un seuil de tolérance défini par l'utilisateur, l'entité mobile est corrigée. Le script procède ainsi pour chaque couple d'entités devant être topologiquement cohérent.

Sur la figure ci-après, l'entité à corriger est la polyligne rouge. Pour le premier nœud de cette dernière, le script cherche un point fixe à proximité (en bleu) et corrige la polyligne. Il procède ensuite de même avec le point suivant.

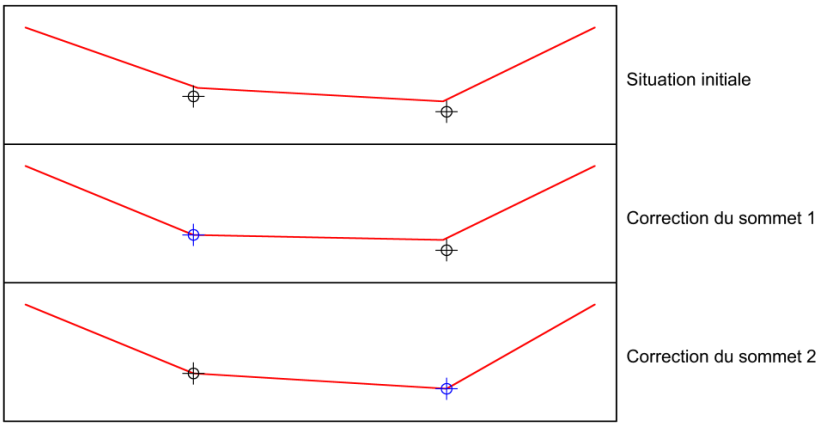


Figure 1 : Fonctionnement du script de Florian GANDOR

I.2 Réalisations

I.2.1 Reprise du script initial et premiers ajouts

Ce premier script propose un fonctionnement à la fois simple et efficace. Cependant, il possède quelques défauts qui doivent être corrigés pour un traitement de qualité. Ces premiers ajouts constituent des modifications mineures facilement implémentées dans le code d'origine.

I.2.1.1 Ordre des contrôles

Pour commencer, la solution initiale traite les différents couples dans un ordre arbitraire. Or, nous avons dit que lors de chaque correction, une entité est considérée comme fixe tandis que l'autre est définie comme mobile. Il faut donc que l'entité fixe, qui sert de référence, n'ait pas à être corrigée par la suite, d'où l'importance de déterminer un ordre précis pour effectuer les corrections.

Voici la méthodologie suivie pour déterminer cet ordre :

- La liste exhaustive des couples à contrôler est consignée dans un tableau à trois colonnes : rang (vide pour l'instant), entité fixe et entité mobile. Les entités sont colorées suivant si elles interviennent comme toujours fixes (en vert), toujours mobiles (en bleu), ou les deux suivant le contrôle considéré (en rouge).
- Le tableau est ensuite trié pour regrouper les contrôles ayant la même entité mobile. Si pour un de ces groupes les éléments de la colonne « entité fixe » sont tous verts, le contrôle peut être effectué. On place le groupe en tête du tableau et on y associe le rang 1. Les entités mobiles correspondantes passent alors en vert dans les deux colonnes
- L'étape précédente est alors répétée pour le rang 2, puis 3, etc. jusqu'à ce que toutes les entités soient colorées en vert.

Une illustration de cette méthode est disponible en annexe 1.

I.2.1.2 Outil standalone

L'outil proposé par Florian GANDOR n'était pas utilisable directement, mais devait être exécuté depuis la suite logicielle Anaconda® qui intègre un environnement de développement Python™. De plus, l'exécution s'effectuait en ligne de commande. L'amélioration consiste donc en la création d'une interface, ainsi que la « compilation » de l'application.

L'interface permet tout d'abord une saisie plus rapide et plus intuitive des paramètres de traitement. De plus, elle présente également l'avantage de regrouper plusieurs outils réalisés par la suite. Elle a été entièrement codée avec le module TkInter présent dans la distribution de Python™ 2.7.

Concernant la « compilation », elle procure 2 avantages. De prime abord, elle permet à l'utilisateur de travailler sans accéder au code source, et sécurise ainsi ce dernier contre toute modification involontaire. De plus, elle permet d'utiliser l'application depuis n'importe quel poste informatique, et de manière autonome. À noter que le terme « compilation » est à mettre ici entre guillemets. En effet, Python™ est un langage interprété et non compilé (l'utilisation d'un script Python™ nécessite un interpréteur qui « lit » les lignes de code et les exécute à la volée). De ce fait, le compilateur utilisé (py2exe dans notre cas) encapsule dans un exécutable le code source et un interpréteur Python™, sans réaliser une véritable compilation (à aucun moment le code source n'est traduit en langage machine). De plus, l'ensemble des bibliothèques nécessaires à l'exécution du script sont rassemblées dans le même dossier que

cet exécutable. Ainsi, on obtient une application qui n'est plus dépendante d'une installation particulière. Pour pouvoir la faire fonctionner, il suffit de copier l'ensemble du dossier contenant l'exécutable sur le poste de l'opérateur.

On remarquera qu'un fichier .pyc est généré au moment de l'exécution d'un script Python™. Il correspond à une version semi-compilée de celui-ci, et qui permet une exécution plus rapide. Il est dit semi-compilé car c'est un code intermédiaire entre code source et langage machine. Sa distribution aurait permis de résoudre la contrainte de protection du code (celui-ci n'est alors plus compréhensible par l'utilisateur et le rebute à effectuer des modifications), mais nécessite toujours une installation de Python™ sur le poste utilisé. Cette solution n'aurait donc pas entièrement rempli l'objectif fixé.

I.2.2 Modifications majeures du script

Les modifications qui suivent représentent des améliorations plus conséquentes, portant sur le fonctionnement lui-même du script initial. Pour plus de clarté, un nouveau script a été réalisé, s'inspirant du premier mais implémentant ces améliorations dès sa conception.

I.2.2.1 Notion de « plus proche »

La première correction majeure du script s'est portée sur l'implémentation de ce que nous appellerons la notion de « plus proche ». C'est-à-dire que si plusieurs corrections sont possibles, le script devra choisir d'effectuer celle par rapport à l'entité la plus proche.

En effet, le script d'origine ne prend pas en compte cette notion, et corrige l'entité mobile par rapport à chaque objet fixe intervenant dans le contrôle (à condition bien sûr que l'écart entre les deux soit sous le seuil de tolérance). Ainsi, à la fin du contrôle, c'est la dernière correction qui fait foi. Cette manière de fonctionner entraîne de ce fait des erreurs de correction.

La figure ci-dessous présente les conséquences de cette modification : les images supérieures représentent le fonctionnement du script d'origine, tandis que celles du bas correspondent au nouveau. À l'itération 1, la correction va être calculée par rapport au premier point codé dans le shapefile. À la suivante, ce sera le deuxième point.

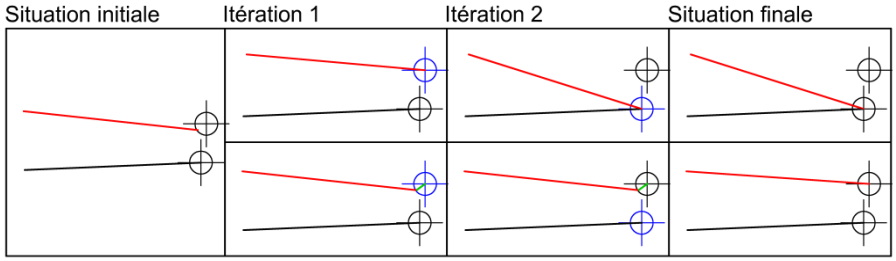


Figure 2 : Notion de « plus proche »

L'ajout de cette notion représente également une modification importante sur le fonctionnement lui-même du script. En effet, plutôt que de corriger l'entité chaque fois que le seuil de tolérance est respecté, il est nécessaire de stocker la correction sans l'appliquer ainsi que l'écart entre les deux entités. Ainsi, chaque fois qu'un contrôle est positif (i.e. entité à corriger), les écarts sont comparés et les coordonnées corrigées les plus proches de celles initiales sont conservées. L'entité mobile est ensuite corrigée une fois que les contrôles avec toutes les entités fixes concernées ont été effectués.

I.2.2.2 Ajout de nouveaux types de contrôles

Après concertation avec l'équipe de saisie, il était apparu que les objets ponctuels étaient plus facilement modélisables que les entités linéaires et, de ce fait, qu'ils étaient saisis avec plus de fiabilité. Florian GANDOR avait donc pensé son script pour considérer tous les ponctuels comme fixes, et de porter les corrections sur les linéaires. Ainsi, il avait défini deux types de contrôles : le test d'un sommet (extrémité ou sommet intermédiaire) de linéaire par rapport à un point, et le test d'un sommet de linéaire par rapport au sommet d'un autre linéaire.

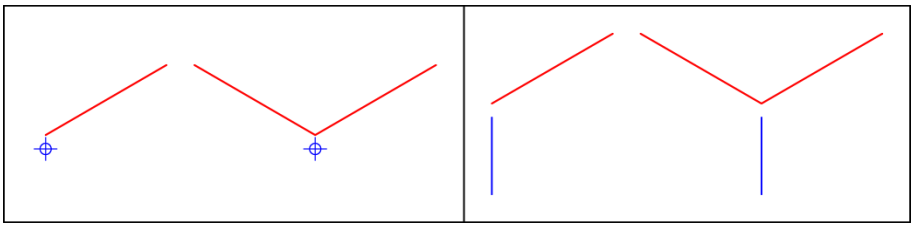


Figure 3 : Les deux types de contrôles implémentés par Florian GANDOR

Cependant, ces deux types de contrôles ne permettent pas de corriger les cas où un sommet de linéaire n'a pas été inséré à proximité de l'entité fixe (dans ce cas, il s'agit d'une faute de modélisation). Il a donc fallu rajouter ces nouveaux types au script.

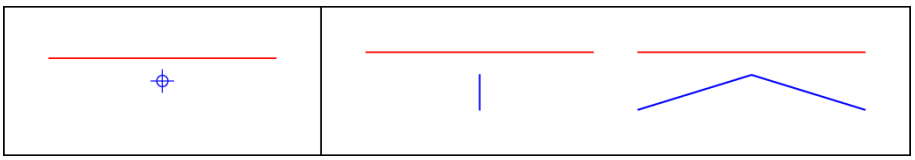


Figure 4 : Nouveaux types de contrôles implémentés

L'algorithme fonctionne comme suit :

- la distance entre le point ou le sommet fixe et le linéaire mobile est calculée par projection orthogonale ;
- cette distance est contrôlée pour respecter le seuil de tolérance et la notion de plus proche (cf. § I.2.2.1) ;
- lorsque les deux étapes précédentes ont été réalisées pour tous les points ou sommets candidats pour la correction, un sommet est inséré dans la portion du linéaire concerné, à l'emplacement de l'objet fixe retenu.

A l'utilisation, il s'est avéré que cet ajout réalisait bien la fonction attendue, mais générait également un nombre important de « fausses corrections ». En effet, il est courant que certains éléments linéaires passent à proximité d'entités ponctuelles sans y être attachés dans la réalité. L'application, qui ne peut faire la différence, va réaliser une modification qui n'a pas lieu d'être. L'illustration suivante est extraite d'ArcScene™. Le carré noir signale une potentielle erreur : d'après notre outil, le hauban (ligne rouge) devrait être en contact avec l'isolateur (point bleu). C'est une fausse détection : en réalité, cet isolateur attache le feeder (ligne jaune) à la console (ligne verte).

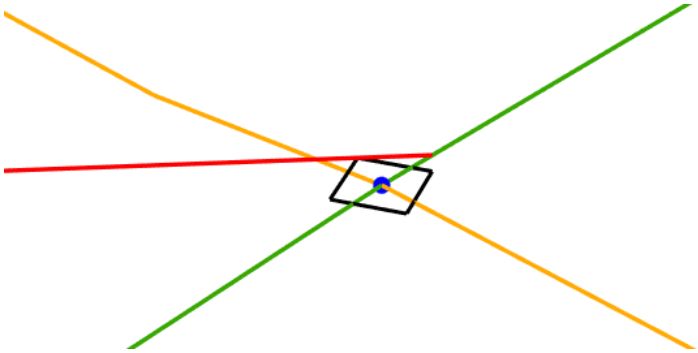


Figure 5 : Fausse détection visualisée dans ArcScene™

Face à ce nouveau problème, la solution trouvée a été de réduire cette fonction à une simple détection, sans correction automatique. Les zones positives à cette détection sont consignées dans un shapefile qui permet ensuite à l'opérateur de visualiser rapidement l'emplacement d'éventuelles erreurs et de les corriger manuellement.

I.2.2.3 Corrections par rapport au bâti et aux mâts TPG

Outre les corrections topologiques internes à la base de données, il faut savoir que le produit doit être cohérent avec l'ensemble des mâts TPG et bâti 3D présents dans le cadastre 3D du canton de Genève.

L'implémentation de base de cette fonction reprend le même principe que les contrôles précédents, mais seules les composantes X et Y sont corrigées. D'une part, cela est suffisant car les façades des bâtiments sont toutes strictement verticales, et d'autre part, cette simplification permet de ne traiter que des intersections de lignes plutôt que des intersections de lignes et de faces (ce qui se traduit par un script plus léger et plus performant). Cependant, deux problèmes apparaissent :

- certains mâts ne semblent pas être aux bons emplacements ;
- des scellements sont situés très loin des façades du bâti et nécessitent un seuil de tolérance élevé pour être corrigés.

Concernant les mâts, les données nous avaient déjà été fournies par les TPG par le passé. Après contact de la personne responsable de la gestion de ces données, il apparaît que des changements de mâts ont eu lieu depuis, d'où les incohérences avec le lever effectué par HKD Géomatique. La dernière mise à jour de ces données date en effet de 2011, et correspond en réalité à une reprojection suite au changement de cadre de référence en Suisse (MN03 vers MN95). Le lever des mâts, quant à lui, a été réalisé en 2008. Malheureusement, aucune base de données plus récente n'est disponible à ce jour. Le lever et la représentation des mâts TPG ne faisant pas partie du mandat, la seule solution applicable consiste à modifier l'application pour que ce contrôle ne soit pas obligatoire. Ainsi, il ne s'effectuera pas si l'utilisateur ne spécifie pas l'emplacement du fichier de la base de données des mâts.

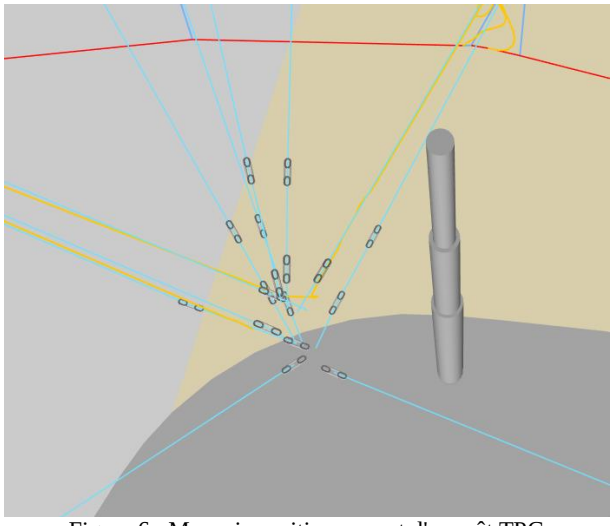


Figure 6 : Mauvais positionnement d'un mât TPG

Pour ce qui est des bâtiments 3D, ils sont directement extraits de la base de données du SITG (système d'information du territoire à Genève). Ils correspondent donc bien à la version la plus à jour. Le problème des écarts importants entre scellements et façades vient du niveau de détails de la modélisation du bâti. En effet, on classe habituellement les bâtiments en 5 groupes de niveaux de détails :

- le LOD 0 qui correspond à une modélisation au niveau d'une région (orthophotographie plaquée sur un MNT) ;
- le LOD 1 qui représente les bâtiments sous forme de blocs avec des toits plats;
- le LOD 2 qui rajoute la modélisation simplifiée de la toiture, et parfois une texture ;
- le LOD 3 qui intègre les détails architecturaux comme les portes ou les fenêtres ;
- le LOD 4 qui représente également l'intérieur du bâtiment.

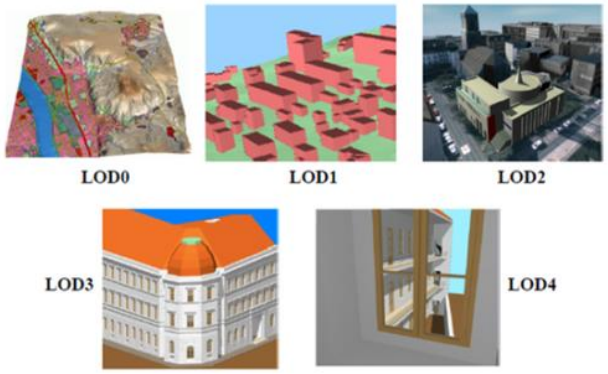


Figure 7 : Les cinq LOD définis par CityGML

Dans notre cas, on se situe au niveau du LOD 2. Comme Laurène MENU (2011) l'avait remarqué, les bâtiments ont été obtenus par extrusion depuis leur empreinte cadastrale. De ce fait, certains éléments sur lesquels reposent les scellements ne sont pas modélisés (les balcons par exemple), d'où des écarts parfois importants. Voici une illustration du phénomène (le mur réel est en rouge tandis que la modélisation est en bleu ; le point noir représente le scellement) :

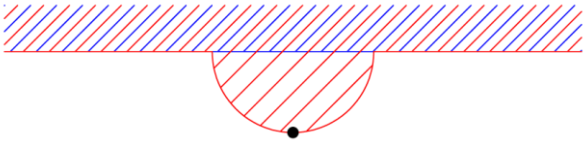


Figure 8 : Erreur topologique entre un scellement et une façade

Il convient donc d'introduire un deuxième seuil de tolérance dans le script, permettant d'avoir une tolérance assez large sur le contrôle des scellements, et plus fine sur le reste de la base.

Cependant, il apparaît que certains écarts sont toujours trop importants : avec un seuil de tolérance fixé à 50 cm, plusieurs scellements restent non corrigés. La question que l'on se pose est alors « faut-il corriger ces scellements à tout prix pour respecter la continuité avec le bâti 3D, ou conserver volontairement ces erreurs de topologie pour garder la précision de notre modélisation ? » Actuellement notre décision s'est portée sur le deuxième choix, mais pour pallier à un possible changement d'avis, la même mesure que pour les mâts a été adoptée (à savoir, laisser à l'utilisateur le choix de corriger ou non la base selon le bâti).

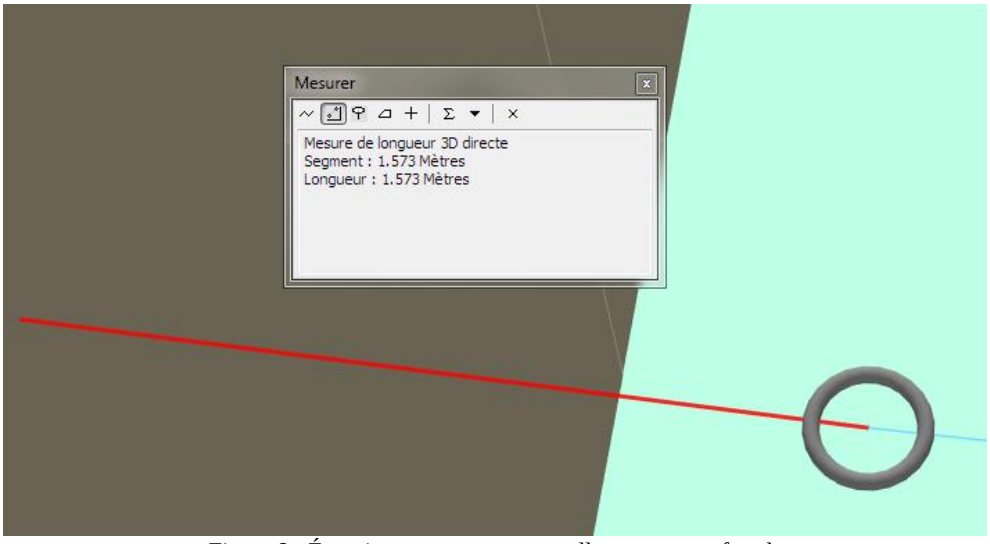


Figure 9 : Écart important entre un scellement et une façade

I.3 Conclusion

Pour ce travail, les recherches que Florian GANDOR a réalisées avant moi ont constitué un solide point de départ. Son script, bien que parfaitement fonctionnel, nécessitait cependant un certain nombre d'ajouts pour répondre aux besoins fixés. Tandis que certains se sont insérés naturellement, d'autres ont demandé un travail de recherche plus poussé et ont nécessité une restructuration complète de l'outil, qui garde néanmoins le principe de fonctionnement global de son premier développeur.

De plus, l'implémentation de nouveaux critères de correction ont parfois provoqué des résultats indésirables, entrainant un vrai travail de réflexion pour trouver le meilleur compromis.

Enfin, il faut concéder le fait que le script n'est pas des plus optimisés. En effet, il a nécessité environ huit heures pour traiter l'ensemble des données, sans effectuer les corrections selon les mâts et le bâti, et sur un ordinateur haut de gamme (processeur Intel Core i7, 32 Go de mémoire vive, disque SSD,…). La faute notamment au langage Python™ qui n'est pas le plus rapide à exécuter (mais un des plus intuitifs à utiliser). Cependant, il faut considérer que cette application reste un outil interne à l'entreprise, et n'a d'utilité qu'en amont pour constituer la base de données. Il n'est donc pas anormal de privilégier la rapidité de développement plutôt que la vitesse d'exécution.

II Conception de la symbologie

Un SIG n'est rien sans une symbologie adaptée. Ceci est d'autant plus vrai lorsque celui-ci est en trois dimensions, où des objets situés sur plusieurs plans peuvent se superposer. Ainsi, il est important de mettre au point une symbologie qui sera parlante pour l'utilisateur final, et représentative de la réalité.

II.1 Etat de l'art

De manière à respecter les contraintes énoncées précédemment, les TPG souhaiteraient que cette symbologie soit inspirée de la réalité. Autrement dit, il faudrait qu'un panneau soit modélisé par un symbole en forme de panneau plutôt que représenté par un point coloré par exemple.

Par ailleurs, la société Kummler+Matter SA, qui fournit les différents éléments aériens des lignes, a accepté que les symboles de leur catalogue soient réutilisés par HKD Géomatique pour ce mandat. De plus, ce catalogue est facilement accessible sur leur site internet et en PDF.

Ainsi, un essai avait été fait, consistant à insérer directement ces symboles (donc sous forme d'images 2D) dans la modélisation. Cependant, le résultat avait été peu probant et a démontré la nécessité d'utiliser de véritables symboles en 3D. Ceci est possible avec ArcGIS®, qui permet d'utiliser des modèles 3D dans divers formats (COLLADA, Skp, 3DS, etc.).

Concernant la modélisation des symboles elle-même, Trimble® SketchUp® parait parfaitement adapté car gratuit, simple d'utilisation, et dont le format de fichier natif (Skp) est pris en charge par ArcGIS®.

II.2 Réalisations

II.2.1 Modélisation des symboles

II.2.1.1 Dessin des symboles

Comme présenté précédemment, le logiciel Trimble® SketchUp® a été choisi pour modéliser les symboles en 3D. Ce travail ne sera effectué que pour les entités ponctuelles. En effet, la symbologie des linéaires sera définie directement dans ArcGIS® avec une simple attribution d'épaisseurs et de couleurs aux polylignes.

Le catalogue de Kummler+Matter SA propose pour chaque objet un(e) à quelques schéma(s) et/ou photographie(s) ainsi que certaines côtes (un exemple est disponible en annexe 2). A partir de ces données, il est donc possible de reproduire chaque élément sous Trimble® SketchUp®, moyennant quelques simplifications. L'objet n'étant pas toujours dimensionné en totalité, il est souvent nécessaire d'estimer les côtes manquantes par des mesures à l'écran puis par un produit en croix par rapport aux mesures connues. De ce fait, il en résulte des modèles qui ne peuvent reproduire parfaitement la réalité, mais qui restent très suffisants pour s'en faire une bonne représentation (cf. annexe 3).

Les symboles sont alors prêts à être utilisés par ArcGIS®.

II.2.1.2 Import et attribution des symboles

Pour pouvoir attribuer correctement les symboles, il convient au préalable de catégoriser chaque classe d'entités selon un ou plusieurs champs (par exemple, les panneaux seront

catégorisés selon le type de véhicule, le type de panneau et la vitesse (qui n'a de réelle utilité que pour les panneaux de type « vitesse »)). Il suffit ensuite d'importer le bon symbole pour chaque catégorie de chaque classe d'entités. Pour ce faire, il faut, lors de la définition du symbole, choisir le type « symbole ponctuel 3D » et sélectionner le fichier approprié. La procédure est donc très simple.

Initialement, l'idée était de garder les symboles au format natif de Trimble® SketchUp®, qui est censé être reconnu par ArcGIS®. Cependant, ce dernier ne prend pas en charge toutes les versions du format Skp, ce qui rend la manipulation impossible dans notre cas. Heureusement, Trimble® SketchUp® permet l'export au format COLLADA qui est aussi reconnu par ArcGIS®. La manipulation peu alors s'effectuer sans problème. De plus, ce format de fichier est également reconnu par un grand nombre de logiciels de modélisation et de CAO, ce qui en fait finalement un choix judicieux si ces modèles devaient être réutilisés ou retravaillés par la suite.

II.2.2 Développements complémentaires

II.2.2.1 Gestion de l'orientation des symboles

À ce stade, la symbologie est bien affectée à chaque entité de la base. En revanche, l'orientation de nos symboles reste celle par défaut et ne correspond pas à la réalité.

De même que pour les contrôles topologiques, réaliser l'orientation de chaque symbole manuellement représenterait un travail titanesque. De ce fait, automatiser la procédure apparait à nouveau comme une évidence. Après quelques recherches parmi les options de symbologie d'ArcGIS®, il apparaît la possibilité d'orienter automatiquement les symboles selon un champ. Il suffit donc de développer une application qui se chargera de calculer l'orientation de chaque symbole et de l'enregistrer dans un champ approprié. Le langage Python™ ayant fait ses preuves pour la manipulation des shapefiles lors des contrôles topologiques, il est de nouveau utilisé pour résoudre ce problème.

Le fonctionnement du script est relativement simple : pour chaque ponctuel ayant besoin d'être orienté (isolateurs, isolateurs de haubans, sectionneurs, aiguillages, croisements, antennes, panneaux, feux d'aiguillages, boucles isolantes et guides-perches), un nouvel attribut est créé : « rot_z », qui correspond à l'orientation du symbole selon l'axe (O, z) (c'est donc un gisement). Chacune d'elles est ensuite calculée de la manière suivante : le sommet de linéaire sur lequel l'entité est fixée est identifié, puis le gisement de ce sommet vers le suivant est calculé (ou vers le précédent lorsque l'entité à orienter est située sur le sommet final du linéaire).

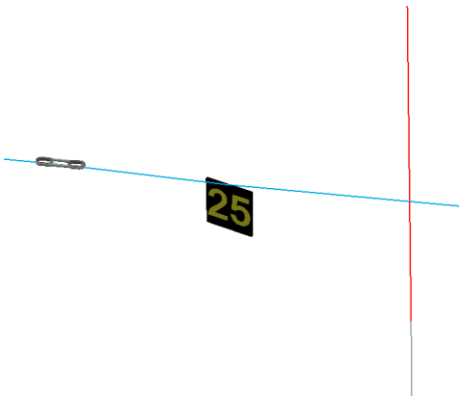


Figure 10 : Orientation par défaut d'un panneau

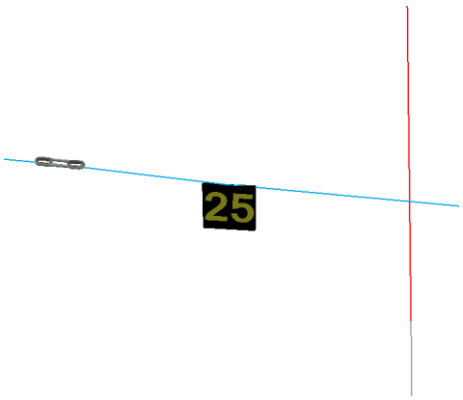


Figure 11 : Orientation d'un panneau après calcul

Les isolateurs de haubans représentent un cas particulier. En effet, ceux-ci doivent également être orientés selon leur axe (O, x) (équivaut à une distance zénithale). Un attribut « rot_x » est donc créé pour cette classe d'entités et est complété d'une manière similaire au calcul des attributs « rot_z » (la différence réside dans le calcul de distances zénithales plutôt que de gisements).

Pour finir, cette étape faisant partie du travail de conception du SIG au même titre que les contrôles topologiques, et étant tout deux développés dans le même langage de programmation, les deux outils se voient regroupés dans une interface commune.

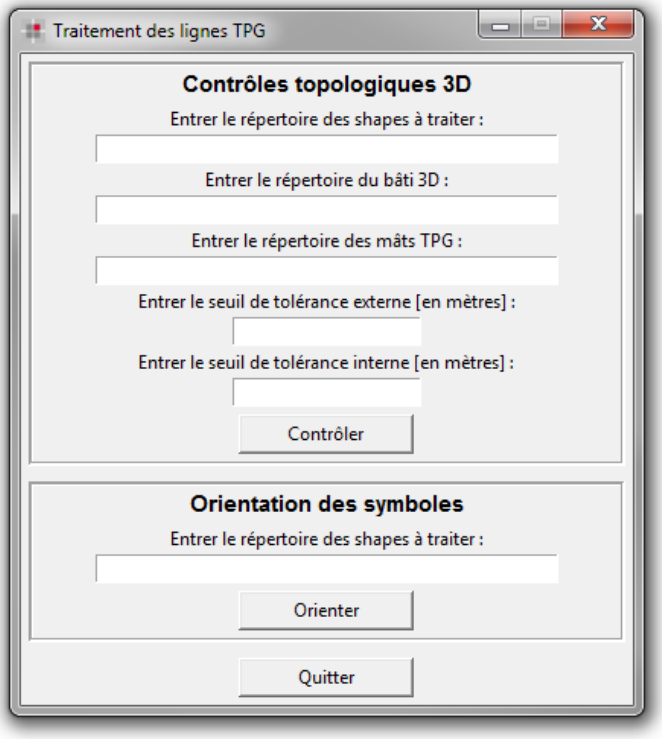


Figure 12 : Interface du programme "Traitement des lignes TPG"

II.2.2.2 Gestion de l'affichage 2D/3D

A l'usage, il apparaît une grande faiblesse à notre modélisation. En effet, il faut savoir qu'en plus de la visualisation en trois dimensions, ArcScene™ propose également une vue orthographique (donc en 2D, vue du dessus). Or, si notre symbologie est parfaitement adaptée à la vue 3D, il en est tout autrement lors du passage en vue orthographique : les symboles apparaissent trop petits, et parfois mal orientés (les informations des panneaux ne sont plus visibles par exemple).

Figure 13 : Symbologie 3D et vue orthographique

La solution consiste en la création d'un outil qui permet de jongler entre les deux types de visualisation et d'adapter la symbologie à la vue choisie. Il faut également savoir que cet outil doit être disponible directement dans ArcScene™, intégré dans une barre d'outils. Il est donc nécessaire de créer une extension qui regroupera cet outil ainsi que d'autres fonctions que nous développerons plus tard.

Pour réaliser une telle extension, deux solutions sont possibles :

- la programmation avec le framework Microsoft®.NET (C# ou VB.NET) ou en Java™ depuis la version 10.0 ;
- la programmation en langage Python™ à partir de la version 10.1.

HKD Géomatique disposant de licences pour ces deux versions, il est décidé d'utiliser le langage C#, qui permettra le développement d'une extension compatible avec toute version d'ArcGIS® à partir de la 10.0. De plus, C# permet de créer des interfaces relativement élaborées (avec la gestion de divers évènements survenant dans les boîtes de dialogues notamment, chose qui n'est pas possible avec TkInter), ce qui nous sera très utile par la suite. Pour ce faire, il est nécessaire d'installer ArcObjects™ SDK qui permet d'utiliser les fonctionnalités d'ArcGIS® dans le script C#, mais aussi de compiler la solution au format .esriaddin (format des extensions ArcGIS®).

Pour commencer, la symbologie 3D initiale est enregistrée dans un ensemble de fichiers .lyr (un pour chaque classe d'entités) afin de la réutiliser plus tard sans avoir besoin de la redéfinir. On adapte ensuite la symbologie à l'affichage en 2D : la taille des symboles est multipliée par 2 et l'orientation des panneaux et des feux d'aiguillage est modifiée de manière à ce qu'ils soient vus de face par l'utilisateur. Cette nouvelle symbologie est alors sauvegardée dans un deuxième jeu de fichiers .lyr.

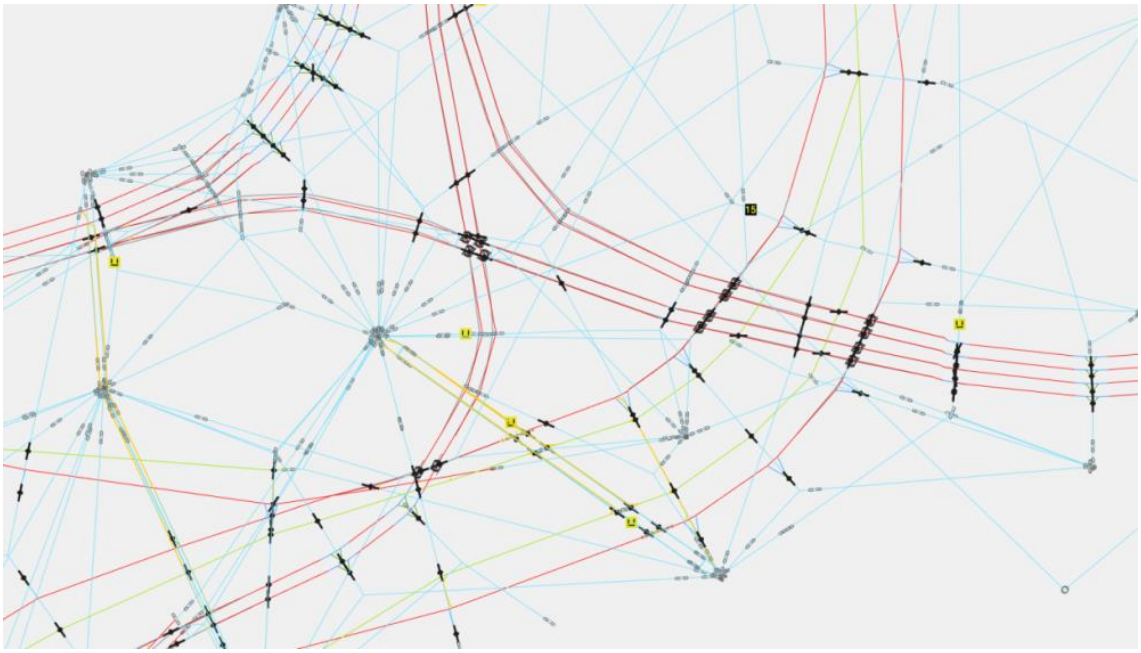


Figure 14 : Symbologie adaptée à la vue orthographique

Deux outils sont alors codés : l'un permet de gérer le passage à la vue en perspective, et l'autre gère l'affichage en 2D. Chacun de ces outils devant utiliser respectivement les deux jeux de symboles définis précédemment (les fichiers .lyr), il est nécessaire de les définir comme ressources incorporées plutôt que de les utiliser tels-quels. Ainsi, l'extension compilée intègrera elle-même ces fichiers et constituera un fichier unique, comme c'est le cas habituellement pour une extension ArcGIS®. Le code en lui-même reste assez simple puisqu'il suffit d'utiliser les outils d'ArcGIS® disponibles via ArcObjects™ SDK pour appliquer la symbologie et changer la vue. La difficulté résulte dans le fait que l'extension ne peut utiliser directement ces ressources (hormis certains types de fichier comme les images). Il faut donc les extraire auparavantauto. Pour cela, l'extension crée un répertoire temporaire, y extrait la symbologie à utiliser, l'applique à la base de données, et supprime finalement le répertoire d'extraction (l'annexe 5 illustre ce procédé).

II.3 Conclusion

Mon travail sur la symbologie a clairement été séparé en deux parties. La première, concernant toute l'étape de création des symboles peut paraître essentiellement une étape de production. Elle a toutefois demandé un travail de réflexion, puisqu'il a fallu choisir une représentation des éléments ponctuels suffisamment proche de la réalité pour les rendre facilement identifiables par l'utilisateur final, sans toutefois rentrer dans l'excès afin de ne pas surcharger la visualisation et de ne pas y consacrer plus de temps que nécessaire.

La deuxième étape a demandé un travail de recherche tout autre, puisqu'il s'agissait de résoudre les différents problèmes techniques rencontrés pour exploiter les symboles. Cela a poussé au développement de nouveaux outils pour automatiser les étapes de traitements à la fois pour la conception de la base, mais également pour son exploitation.

Pour finir, il faut préciser que ce travail a été initialement réalisé pour être exploité sous ArcScene™, pour sa prise en charge de la 3D. Cependant, si l'utilisateur souhaite travailler dans un environnement 2D, il n'est pas exclu qu'il se dirige vers le module ArcMap™. Il apparaît alors un nouveau problème qui demanderait à être résolu. En effet, bien que les symboles ponctuels 3D soient bien reconnus, l'affichage n'est pas convaincant (symboles trop petits, mauvaise orientation). Par manque de temps, ce problème n'a malheureusement pas encore pu être traité, et reste donc à résoudre.

III Mise en place d'outils d'export

L'outil que constituera ce SIG pour les TPG pourra leur servir pour diverses applications. Les TPG souhaitent ainsi disposer de fonctions permettant l'export de cette base de données, et plus particulièrement vers le monde du DAO. L'export en PDF est également intéressant pour extraire diverses vues de la modélisation. Ce sont ces points particuliers que nous développerons dans cette partie.

III.1 Etat de l'art

Au premier abord, cette question semble trouver une réponse assez simple, ArcGIS® intégrant nativement un outil d'export vers les formats DWG et DXF. Cependant, il s'avère que celui-ci n'est pas suffisant. En effet, bien que la conversion des linéaires soit fidèle, il n'en est pas de même avec les ponctuels : aucune symbologie n'est appliquée pendant le traitement. Les points sont finalement représentés par des points, et non par des symboles. Ceci nuit à la lisibilité du plan ainsi obtenu et entraine une importante perte d'informations. Nous réaliserons donc un outil qui permettra de pallier à ces imperfections.

Les données étant toujours au format shapefile, il est naturel de se tourner une fois de plus vers le langage Python™ et le module pyshp. En revanche, le format DWG étant un format propriétaire d'Autodesk, il est très difficile (voire impossible) de trouver un tel module pour ce type de fichier. En revanche, le DXF rencontre plus de succès. En effet, étant un format ouvert, sa structure est diffusée publiquement, et il existe la bibliothèque SDXF qui permet de générer un fichier DXF directement depuis un script Python™. De plus, les plans de pose du réseau aérien que les TPG ont en leur possession sont également au format DXF. Le choix de ce format pour notre export permet donc d'uniformiser les deux sources de données.

Par ailleurs, la vocation des plans obtenus par extraction et celle des plans de pose seront similaires. Ces derniers étant en 2D, notre export le sera également, toujours dans ce souci d'uniformité. Il faudra aussi réaliser une nouvelle symbologie, inspirée de celle présente sur les plans de pose. Pour cela, les TPG ont mis à notre disposition un document qui consigne ces symboles. Certains pourront être repris tel-quels, mais pour quelques éléments non représentés sur ces plans, il faudra les créer de toutes pièces.

Il serait également intéressant pour les TPG de disposer d'un outil permettant l'export en PDF, de manière à réaliser des présentations. ArcGIS® intègre également ce type d'export et le réalise très bien. Une classe « ExportPDF » est d'ailleurs présente dans ArcObjects™ SDK et laisse présumer une possible intégration dans notre barre d'outils. Une autre possibilité : il existe des modules pour Python™ qui permettent la lecture et/ou l'écriture de fichiers PDF, de la même manière que SDXF avec les DXF. Le PDF est en effet un format ouvert, lui aussi, et donc relativement facile à reproduire, en théorie. Enfin, reste la possibilité d'utiliser l'export DXF puis un outil de conversion DXF vers PDF, mais l'outil ne serait plus fidèle à la représentation sous ArcGIS®.

III.2 Réalisations

III.2.1 Export vers le monde de la DAO

III.2.1.1 Définition d'une symbologie DXF

Comme on vient de le voir, il est nécessaire de réaliser une nouvelle symbologie spécifique pour le fichier DXF issu de l'export. L'idée de départ était de réaliser manuellement

des blocs pour ces symboles et de les regrouper dans un même répertoire (en quelque sorte, un travail similaire à celui décrit au paragraphe II.2.1.1), puis de les insérer dans le DXF lors de l'export. Cependant, il se trouve que le module SDXF ne permet pas d'insérer des éléments extérieurs dans le dessin. En revanche, il est tout à fait possible de créer nos blocs directement par programmation dans le script Python™. Cette solution nécessite de réaliser des tests après l'implémentation de chaque bloc, afin de vérifier si celui-ci est bien tracé comme espéré. Pour éviter une perte de temps importante à lancer la totalité du script (et donc traiter toute la base de données) une méthodologie a été mise en place :

- on crée un script annexe, dans lequel on code uniquement la définition du bloc en cours de création et son insertion à l'origine du dessin. Ces blocs correspondent à un ensemble de polylignes et de solides ;
- lorsque le résultat escompté est atteint, cette portion de code est insérée dans le script principal.

```
b_Panneau_danger = dxf.Block("Panneau_danger_bloc")
b_Panneau_danger.append(dxf.PolyLine(points = [(-1.6,0.4),(1.6,0.4),(1.6,-0.4),(-1.6,-0.4),(-1.6,0.4)]))
b_Panneau_danger.append(dxf.PolyLine(points = [(-1.2,0.35),(-1.55,-0.35),(-0.85,-0.35),(-1.2,0.35)]))
b_Panneau_danger.append(dxf.Solid(points = [(-1.2314,0.2271),(-1.1560,0.2271),(-1.3247,-0.1302),(-1.2678,-0.0526)]))
b_Panneau_danger.append(dxf.Solid(points = [(-1.2678,-0.0526),(-1.1214,0.0373),(-1.3247,-0.1302),(-1.2117,-0.0689)]))
b_Panneau_danger.append(dxf.Solid(points = [(-1.2117,-0.0689),(-1.1214,0.0373),(-1.2155,-0.2098),(-1.1856,-0.2153)]))
b_Panneau_danger.append(dxf.Solid(points = [(-1.2464,-0.1951),(-1.1513,-0.2125),(-1.2172,-0.3204),(-1.2172,-0.3204)]))
b_Panneau_danger.append(dxf.Text(text = "600 Volts",point = (-0.72,-0.12), height = 0.25))
b_Panneau_danger.append(dxf.PolyLine(points = [(1.2,0.35),(1.55,-0.35),(0.85,-0.35),(1.2,0.35)]))
b_Panneau_danger.append(dxf.Solid(points = [(1.1686,0.2271),(1.244,0.2271),(1.0753,-0.1302),(1.1322,-0.0526)]))
b_Panneau_danger.append(dxf.Solid(points = [(1.1322,-0.0526),(1.2786,0.0373),(1.0753,-0.1302),(1.1883,-0.0689)]))
b_Panneau_danger.append(dxf.Solid(points = [(1.1883,-0.0689),(1.2786,0.0373),(1.1845,-0.2098),(1.2144,-0.2153)]))
b_Panneau_danger.append(dxf.Solid(points = [(1.1536,-0.1951),(1.2487,-0.2125),(1.1828,-0.3204),(1.1828,-0.3204)]))
Dessin.blocks.append(b_Panneau_danger)
```

Figure 15 : Un bloc DXF codé en Python™ avec le module SDXF

Figure 16 : Visualisation du bloc codé en figure 15

III.2.1.2 Développement de l'outil

Grâce aux deux modules précédemment cités, le principe de fonctionnement apparait globalement : l'idée est de lire la géométrie et les attributs des shapefiles en entrée, puis de traiter ces données pour écrire un fichier DXF en sortie.

Pour rentrer plus en détails dans le fonctionnement du script, celui-ci est structuré en trois grandes parties. La première « initialise » le traitement : elle s'occupe d'importer les différents modules utilisés dans le script, de spécifier les répertoires de travail (celui contenant les données ainsi que celui où sera enregistré le fichier exporté), et enfin de lire les données contenues dans les shapefiles. Plus tard, lors de la définition des couleurs des calques, un dernier élément a été rajouté : le DXF utilisant un système numérique pour représenter les couleurs (par exemple le chiffre 1 code le rouge, le 2 code le jaune, etc.), les couleurs utilisées n'apparaissaient pas clairement dans le script, ce qui nuisait à sa lisibilité. De ce fait, la correspondance de quelques couleurs de base avec leur code couleur a été implémentée. Ainsi, plutôt que d'appeler une couleur par son code, elle peut l'être par son nom.

La deuxième partie est une étape de préparation du dessin. Elle comprend la création des différents calques et la définition des symboles sous forme de blocs. Il est intéressant de regrouper dans une même partie de script toutes les définitions de couches et de blocs. Structuré de cette manière, celui-ci gagne en visibilité et facilitera le travail d'un futur développeur si le script devait être repris par la suite.

Enfin, la troisième étape traite réellement les données et les trace dans le DXF. D'une manière général, le procédé est le même pour chaque classe d'entités :

- pour un ponctuel, le script récupère les coordonnées du premier élément, insère le bloc correspondant à ces coordonnées, puis passe au suivant ;
- pour un linéaire, le script récupère les coordonnées de chaque point du premier élément et les insère dans une table. Lorsque chaque point a été traité, le linéaire est tracé suivant cette table. L'opération est ensuite réitérée pour les éléments suivants.

Cependant, certains symboles à insérer entraînent des particularités :

- pour la plupart des ponctuels, une orientation est à prendre en compte. Il se trouve qu'elle est identique à celle enregistrée dans l'attribut « rot_z » calculée précédemment (cf. § II.2.2.1), et est donc récupérée directement dans la table attributaire ;
- certains symboles n'ont pas d'équivalent dans le modèle d'origine : les retenues de courbes (cas particuliers de pinces de courbes), les pinces de passage, les croisements de lignes de contact à hauteurs différentes, et les amarrages pour fil de contact ou feeder. Pour ceux-ci, des tests particuliers sont donc à effectuer pour les détecter :
    - Les retenues de courbes correspondent aux pinces de courbes pour trolleybus, qui sont ici représentées différemment de celles pour tramways. Initialement, une pince de courbe est modélisée par une polyligne passant par 3 points. Ainsi, si la pince de courbe a bien la valeur « TB » pour l'attribut « Type_vehic », le script devra localiser le point du milieu de cette pince et y insérer le symbole adéquat.
    - Les pinces de passage servent de transition entre les lignes de contact et les tubes. De ce fait, pour les détecter, le script devra tester à chaque extrémité de tube si ladite extrémité touche une ligne de contact ou non, et insérer ou non la pince en fonction du résultat.
    - Pour les croisements de lignes de contact à hauteurs différentes, le script cherche pour chaque segment si un autre l'intersecte en deux dimensions. Si c'est le cas, les coordonnées X et Y du point d'intersection permettent d'interpoler deux nouveaux points d'altitudes différentes sur les deux segments en 3D. Le symbole est alors inséré sur le plus haut.
    - Enfin, les amarrages pour fil de contact ou feeder correspondent à des haubans supportant directement un fil de contact ou un feeder, sans passer par une suspension. Pour les détecter, le script détecte les extrémités des haubans qui correspondent aussi à des extrémités de ligne de contact ou de feeder. Le symbole est ensuite inséré au milieu du segment de hauban menant à cette extrémité.

  Pour ces 4 éléments, aucune orientation n'avait été calculée pour la base de données, puisqu'ils n'étaient pas représentés. Il faut donc penser à le déterminer ici, de la même manière que décrite au paragraphe II.2.2.1.

III.2.1.3 Intégration dans ArcGIS®

Chronologiquement, cet outil a été développé avant celui concernant la gestion de l'affichage sous ArcScene™ (cf. § II.2.2.2). De ce fait, l'intégration dans le logiciel n'avait pas encore été abordée, et le développement avec des outils maîtrisés avait été privilégié pour assurer une bonne avancée des recherches. Cependant, comme expliqué précédemment, la création d'une extension ArcGIS® a nécessité un développement en C#, ce qui impliquerait de recoder entièrement l'outil dans ce langage. En plus d'entrainer une perte de temps non négligeable, ceci nécessite l'utilisation d'un équivalent en C# de SDXF. Un seul a été trouvé et, selon son développeur, celui-ci génère parfois des erreurs. Cette solution n'est donc pas viable pour réaliser un produit destiné à être livré à un client.

En revanche, mes recherches m'ont montré qu'il est possible d'exécuter une application extérieure au script C# depuis celui-ci. Ainsi, la partie interface de l'outil est programmée dans ce langage, tandis que la partie traitement reste en Python™. Cependant, cette dissociation pose certaines contraintes. En effet, il faut déjà que la partie interface « connaisse » l'emplacement du module de traitement en toute circonstance. De plus, l'extension doit être fonctionnelle sans avoir besoin d'application extérieure (autre qu'une installation d'ArcGIS® bien entendu). Et finalement, il faut protéger la partie codée en Python™, où le code source n'est pas protégé contre une modification involontaire de la part de l'utilisateur.

Un procédé permet de répondre à ces trois problématiques. Celui-ci consiste à réunir la « compilation » du script Python™ comme évoqué au paragraphe I.2.1.2, et l'incorporation de celui-ci en ressource de l'application C# comme présenté au paragraphe II.2.2.2. Outre l'exécutable, il faut également incorporer dans ces ressources l'ensemble des fichiers générés par py2exe.

La partie interface de l'outil réalise ainsi un certain nombre d'actions en amont du véritable traitement des données. Premièrement, elle propose à l'utilisateur de choisir le répertoire où s'effectuera l'export. Le chemin d'accès à ce répertoire est alors noté dans un fichier .txt, lui-même placé dans un répertoire temporaire. Nous verrons l'utilité de cette action par la suite.

Ensuite, les différentes couches présentes dans la scène (pour ArcScene™) ou dans la vue (pour ArcMap™) sont exportées au format shapefile dans le répertoire temporaire. La nécessité de cette étape est due au fait que les données seront livrées au format géodatabase (.gdb), qui est illisible pour pyshp.

Le script de traitement (et tous ses fichiers annexes) est alors extrait à son tour dans le répertoire temporaire, et est exécuté. Il va commencer par récupérer le chemin d'accès inscrit dans le fichier .txt mentionné précédemment afin de savoir où écrire les données en sortie, et réalise enfin l'export. Pour finir, le dossier temporaire et son contenu sont effacés.

III.2.2 Export en PDF

III.2.2.1 Export via ArcObjects™ SDK

Comme on l'a vu précédemment, ArcGIS® propose déjà un outil d'export PDF. Pour commencer, on va donc tenter de le réutiliser et de l'implémenter directement dans notre barre d'outils, de manière à centraliser toutes les fonctions susceptibles d'intéresser les techniciens des TPG. Quelques tests de l'outil montrent qu'il est parfaitement fonctionnel, et permet d'exporter directement la vue courante.

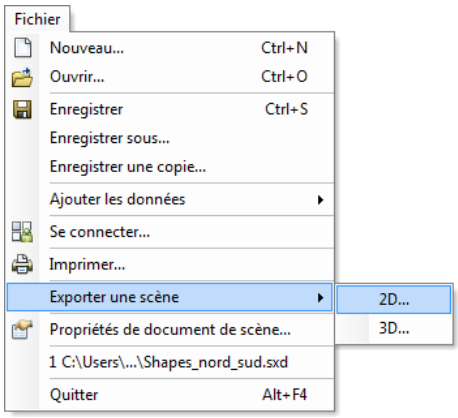


Figure 17 : Outils d'export 2D d'ArcScene™

Dans ArcObjects™ SDK, il est nécessaire de passer par la création d'une instance de la classe « ExportPDF », qui correspond à ce même outil. Cependant il apparait que celui-ci est surtout utilisé dans ArcMap™ et peu (voire pas du tout) sous ArcScene™. Les différents exemples d'utilisations trouvés (y compris sur l'aide officielle d'ArcGIS®) ne l'utilisent ainsi qu'avec un environnement ArcMap™, et font donc appel à des types d'objets qui ne sont pas

présents dans ArcScene™. Bien que la manipulation soit en théorie possible, puisque l'outil intégré est fonctionnel, le manque d'informations sur le sujet n'a pas permis de concrétiser cette piste.

III.2.2.2 Export via Python™

Du fait que la piste précédente n'ait pas permis d'aboutir à un résultat, une deuxième a été explorée : réaliser un travail similaire à l'export DXF (cf. § III.2.1), mais avec une bibliothèque PDF. Il existe en effet des bibliothèques de ce type disponibles sur internet, tel que pyPDF ou pdfrw. Cependant, ceux-ci sont conçus pour traiter principalement du texte, voire des images, mais pas des éléments vectoriels comme il serait nécessaire pour notre projet. De ce fait, pour pouvoir avancer sur cette piste, il convient de réaliser nous-même une bibliothèque adaptée à nos besoins.

Pour ce faire, il faut auparavant étudier la structure d'un fichier PDF. Il existe une norme (ISO 32000-1) qui la décrit. Ce format ayant été créé par la société Adobe Systems, cette dernière met à disposition une version non-officielle et gratuite de cette norme (contrairement à l'originale qui est payante). Ainsi, il est possible de créer nous-même des fichiers PDF respectant cette norme. La structure du PDF est relativement simple, et organisée en plusieurs blocs. Nous ne rentrerons pas ici dans les détails de l'encodage du PDF, mais nous parlerons des difficultés rencontrées. En effet, si les données textuelles sont encodées de manière relativement compréhensible, ce n'est pas le cas pour d'autres types d'objets, tels que les éléments vectoriels (qui sont ceux qui nous intéressent pour notre export).

Un rapide coup d'œil sur un PDF généré depuis AutoCAD® nous montre que ces éléments sont présents dans un bloc « stream », totalement indescriptible pour nous. En réalité, l'ensemble des données présentes dans ce bloc ont été compressées afin de minimiser le poids du fichier. La figure suivante montre un aperçu de ce bloc « stream » lorsque le fichier est ouvert avec un éditeur de texte :

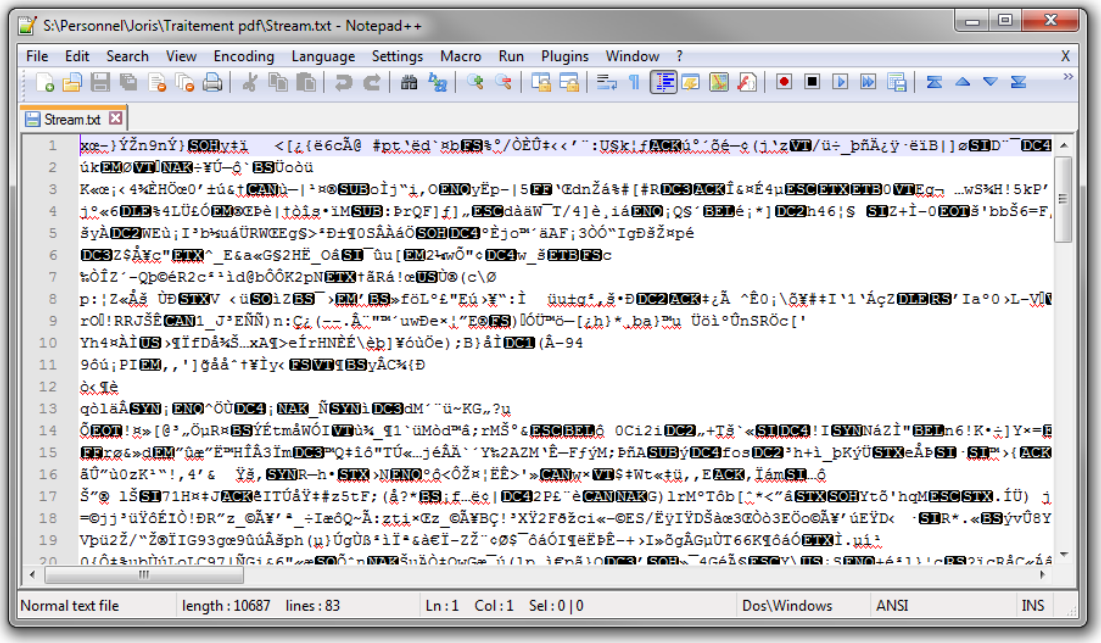


Figure 18 : Extrait d'un stream PDF non décompressé

Alors que les logiciels capables de lire des PDF possèdent les algorithmes nécessaires à la décompression de ces données lors de leur lecture, rares sont les utilitaires qui permettent d'extraire le code initial qui correspond à celles-ci. Il faut également savoir que c'est l'équivalent hexadécimal du code qui subit cette étape de compression/décompression. Après

plusieurs tests, une bibliothèque Python™ a permis d'y parvenir. Une fois le code hexadécimal décompressé, il est converti en son équivalent ASCII qui, après quelques traitements de mise en forme, s'apparente bien à un fragment de fichier PDF :

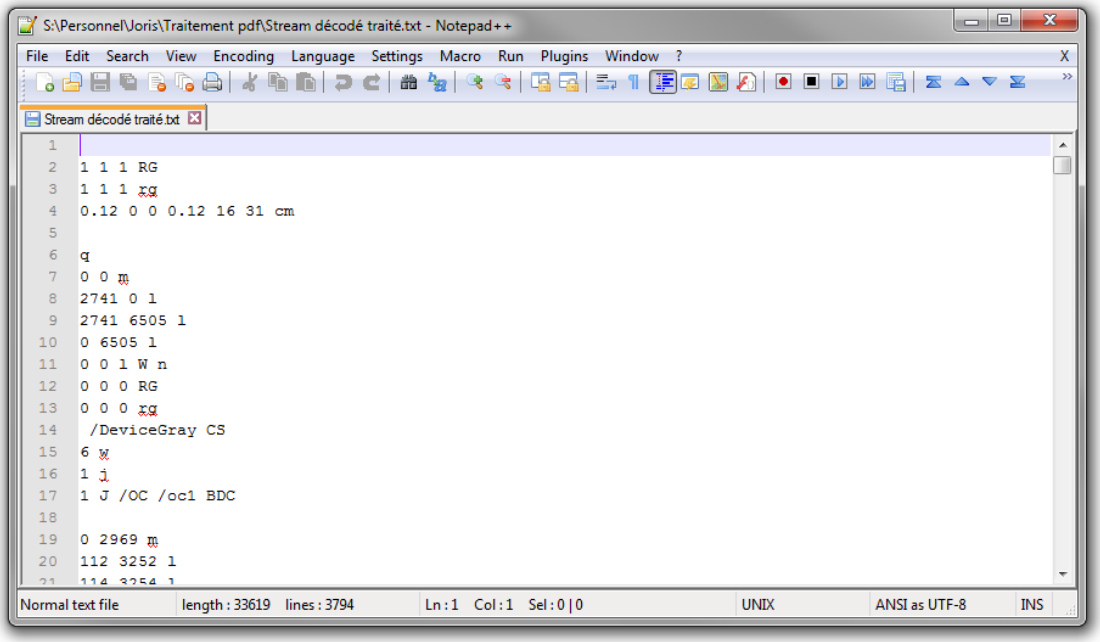


Figure 19 : Extrait d'un stream PDF après décompression

Cependant, il faut reconnaître que la structure reste encore relativement compliquée à comprendre, et que réaliser entièrement une bibliothèque qui permettrait de générer un tel fichier n'est pas possible compte tenu du temps alloué pour le TFE.

III.3 Conclusion

Du côté de l'export en DXF, l'objectif a été pleinement atteint puisque l'outil est parfaitement fonctionnel. Cependant, il subsiste un défaut dans la méthode employée : le fait de devoir extraire les données en shapefile ainsi que le programme d'export et de l'exécuter en dehors d'ArcGIS® plutôt que directement sur les données de la géodatabase demande un temps de traitement relativement important, pendant lequel ArcGIS® se fige. Ce n'est pas un réel problème dans le sens où le traitement est bien effectué et que le logiciel redevient actif à la fin de l'export, mais il serait toujours profitable de rechercher s'il est possible de réaliser l'export en « arrière-plan », de manière à ce que l'utilisateur puisse continuer à travailler pendant ce temps. Enfin, l'outil actuel permet d'exporter la totalité de la base de données. Une amélioration judicieuse serait de permettre à l'utilisateur de limiter l'export à une zone sélectionnée auparavant.

Pour l'export en PDF en revanche, il n'a pas été possible de l'intégrer à notre extension, malgré les différentes pistes explorées. L'utilisateur d'ArcGIS® ayant toujours la possibilité de réaliser cet export simplement avec les fonctionnalités proposées d'origine par le logiciel, il est préférable de ne pas trop s'attarder sur ce point, et de développer les autres outils qui doivent être implémentés dans l'extension.

IV Développement d'outils de mise à jour

Un bon SIG se doit d'être régulièrement mis à jour. ArcGIS® propose déjà des outils de mise à jour, que ce soit en 2D sous ArcMap™ ou en 3D dans ArcScene™. Cependant, rappelons que les personnes susceptibles de devoir les utiliser sont les techniciens des TPG, peu habitués à l'utilisation de logiciels de SIG. Or, ces outils s'avèrent peu intuitifs, et relativement complexes pour une personne non rompue à cette pratique. Ainsi, il sera nécessaire de réaliser un outil tout aussi fonctionnel, mais simple d'utilisation.

IV.1 Etat de l'art

Il est intéressant de noter que le logiciel Trimble® Trident-3D Analyst® utilisé par l'équipe de modélisation propose une méthode efficace pour la saisie des attributs en utilisant un système de listes déroulantes pour choisir la valeur de chaque attribut, ce qui permet de les renseigner conformément au modèle de données et d'éviter les variations orthographiques (majuscules/minuscules, singuliers/pluriels, etc.). De plus, ce système entraine un gain de temps conséquent, puisqu'il suffit de sélectionner une valeur d'attribut plutôt que de la saisir manuellement. Notre outil s'inspirera donc de ce procédé, et reprendra le système de listes déroulantes qui a prouvé son efficacité.

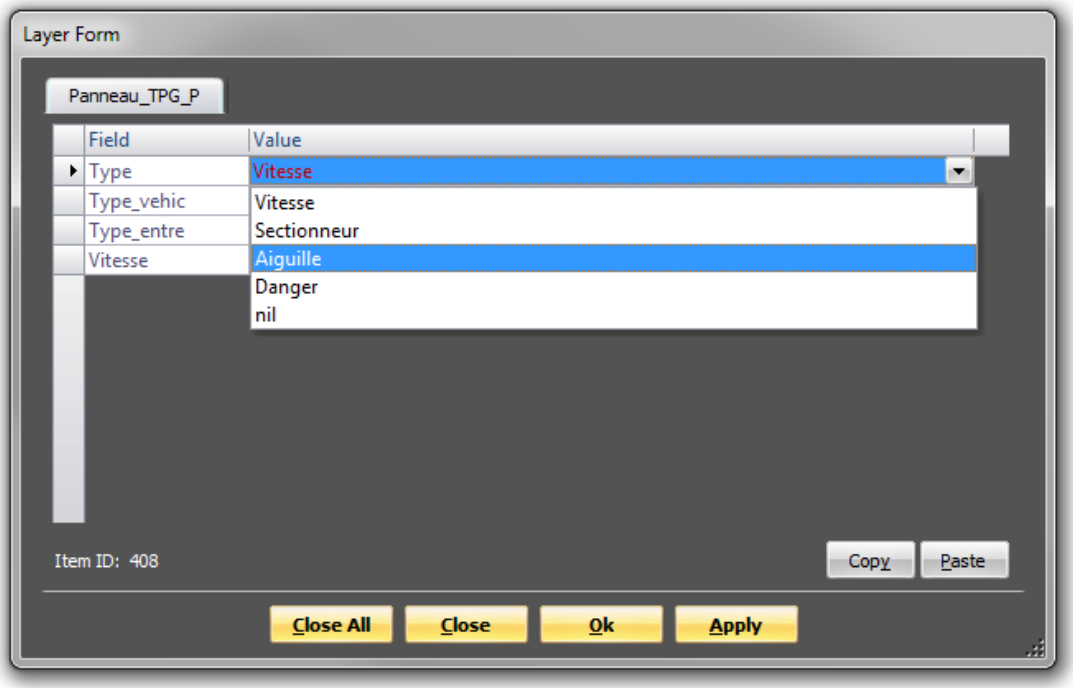


Figure 20 : Système de saisie des attributs dans Trimble® Trident-3D Analyst®

Ne réalisant que l'interface de l'outil, le développement peut se faire directement, et intégralement en C#. En effet, le framework .NET permet de réaliser des interfaces au moyen de boites de dialogues appelées WinForms (contraction de Windows Forms). Chaque WinForm peut ensuite contenir divers Widgets, qui correspondent aux divers éléments graphiques présents dans la boite de dialogue (boites de texte, listes déroulantes, cases à cocher, etc…). Il est ainsi possible de créer une interface totalement adaptée à l'utilisation que l'on veut en faire.

IV.2 Réalisations

IV.2.1 Ajout d'une nouvelle entité

IV.2.1.1 Conception graphique des boites de saisie

L'idée est d'utiliser des WinForms pour guider entièrement l'utilisateur lors de la saisie en lui demandant de remplir les informations géographiques et attributaires concernant l'objet à ajouter. Lorsque les éléments sont à choisir parmi une liste exhaustive de possibilités (la plupart des attributs), la saisie se fait au moyen de listes déroulantes. Pour tous les autres cas (les coordonnées par exemple), elle est réalisée par remplissage de champs. Initialement, il était prévu de créer un seul WinForm dont le contenu aurait été généré dynamiquement selon l'entité à créer. Cependant, le contenu des boites de saisie varient de manière importante d'un élément à l'autre, ce qui se traduisait par un code complexe pour gérer tous les cas. De ce fait, il a finalement été décidé de créer une boite pour chaque classe d'entités, ce qui oblige à créer plus d'objets dans notre application, mais qui se révèle être beaucoup plus compréhensible et simple à coder.

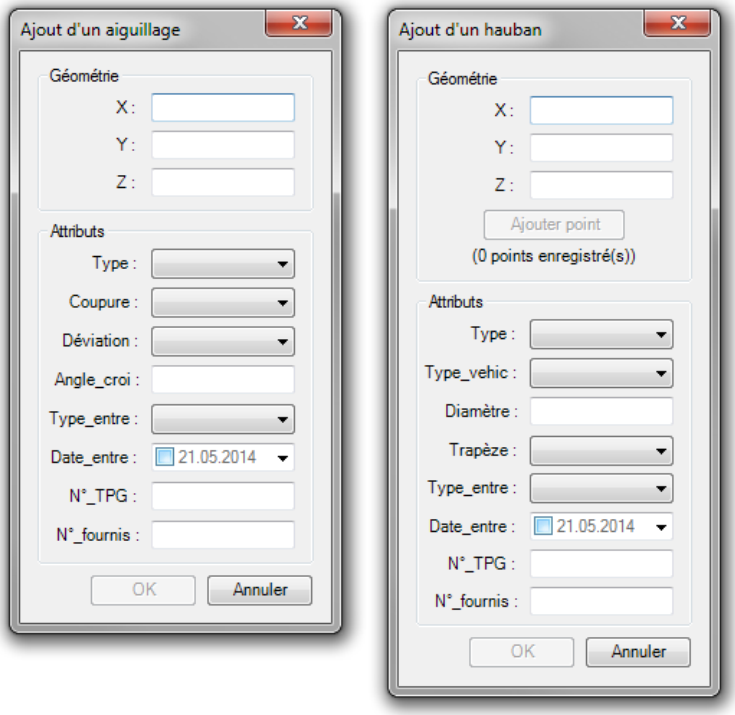


Figure 21 : Exemple de boites de saisie de nouveaux éléments

On remarque que les boites de saisie sont toutes deux divisées en deux parties : « Géométrie » et « Attributs ». Le bloc « Géométrie » diffère suivant le type d'entité : pour un ponctuel, il contient uniquement les champs « X », « Y » et « Z » à remplir, tandis que pour un linéaire, il comporte également un bouton « Ajouter point » et un compteur. En effet, un linéaire étant défini par plusieurs points, il faut que l'utilisateur puisse rentrer l'ensemble des points qui le compose avant de valider la création de l'objet. Par ailleurs, notons que ce bouton n'est pas cliquable tant que les trois champs n'ont pas été remplis (nous en reparlerons un peu plus tard). Les différents points ainsi enregistrés sont consignés dans une liste qui s'actualise au fur et à mesure de la saisie.

Concernant le cadre « Attribut », il varie bien évidement en fonction de l'entité à insérer. Néanmoins, certains attributs sont communs à toutes : « Type_entre », « Date_entre », « N°_TPG » et « N°_fournis ». L'attribut « Date_entre » constitue un cas particulier car il est

rempli à l'aide d'un DateTimePicker. Ce widget permet de sélectionner directement une date dans un calendrier. De plus, le remplissage de cet attribut étant facultatif, il contient une case à cocher pour décider de l'activer ou non :

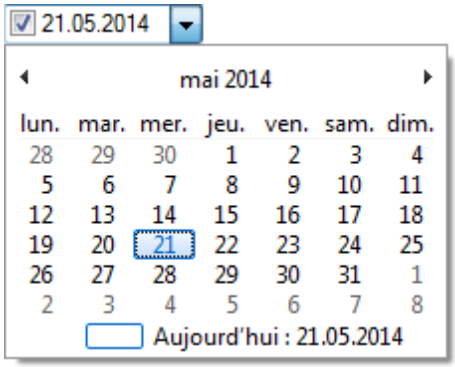


Figure 22 : Saisie d'une date à l'aide d'un DateTimePicker

IV.2.1.2 Articulation de l'interface

Une interface étant composée de plusieurs boites de dialogues, il convient de faire le lien entre chacune d'elles. De plus, pour éviter bon nombre d'erreurs lors du traitement des données renseignées dans celles-ci, il est important de permettre la validation d'une boite uniquement après que les informations obligatoires aient été correctement remplies.

Lors du lancement de l'outil, il est demandé de choisir quel type d'entité est à créer. Cela permet à l'application non seulement de savoir dans quelle couche elle va devoir tracer l'élément, mais également de présenter à l'utilisateur la boite de dialogue de saisie adéquate :

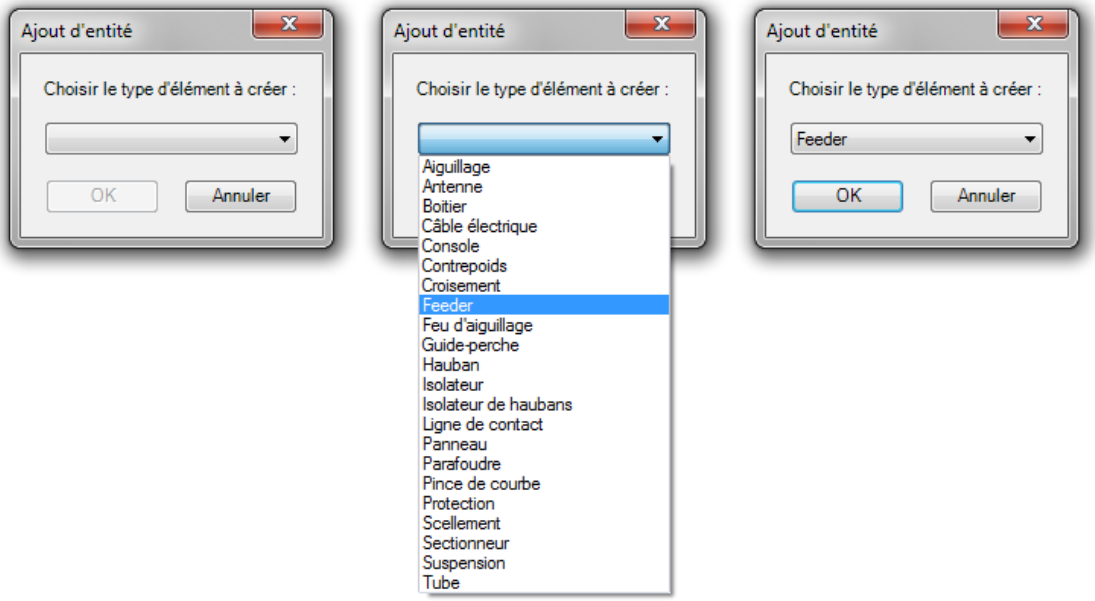


Figure 23 : Sélection de l'entité à ajouter

La figure précédente montre également l'activation du bouton « OK » une fois que le choix a été fait dans la liste déroulante. Ceci est géré à l'aide d'une fonction qui surveille le changement du texte affiché dans la liste, et qui rend le bouton actif uniquement lorsque ce texte n'est pas vide (ce qui correspond à l'évènement « un choix a été fait »).

La fenêtre de saisie adaptée est alors affichée à l'écran :

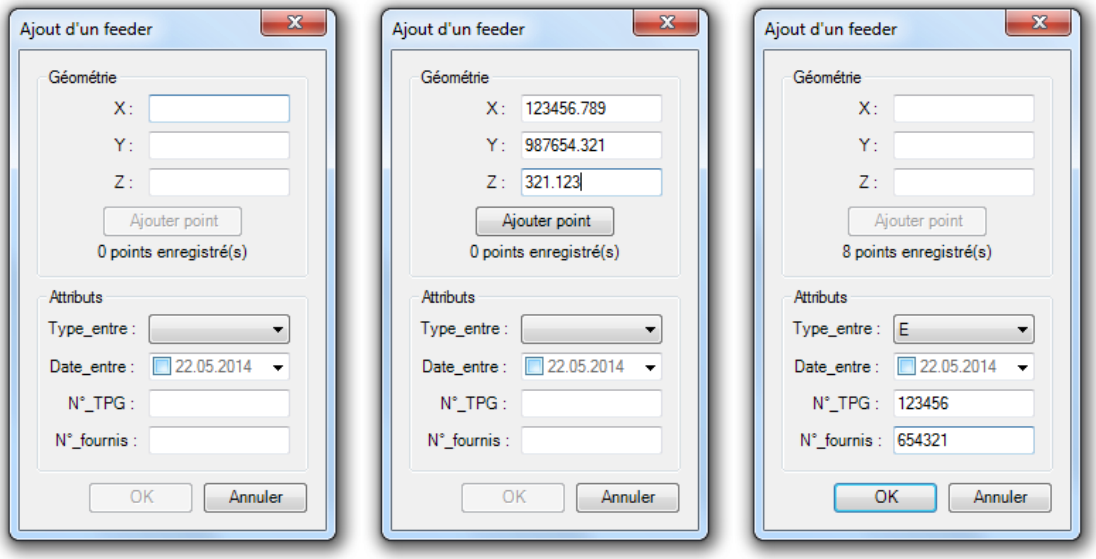


Figure 24 : Saisie d'un nouveau feeder

On remarque ici le même procédé avec le bouton « Ajouter point ». La méthode est identique à celle utilisée dans la boite de dialogue précédente et porte sur les trois champs du cadre « Géométrie ». De plus, il est vérifié que ceux-ci sont bien remplis avec des caractères numériques puisqu'il s'agit de coordonnées. Pour pouvoir valider la boite de dialogue, au moins deux points doivent être enregistrés (ou les coordonnées doivent être renseignées pour un ponctuel) et les attributs doivent être complétés (hormis « Date_entre » qui est facultatif).

Après avoir validé une fenêtre de saisie, un nouveau WinForm permet à l'utilisateur de choisir s'il veut créer une nouvelle entité de même type que celle qu'il vient de saisir ou non. Ainsi, il n'a pas à relancer la procédure depuis le début de l'outil, ce qui constitue un gain en termes de temps et de praticité.

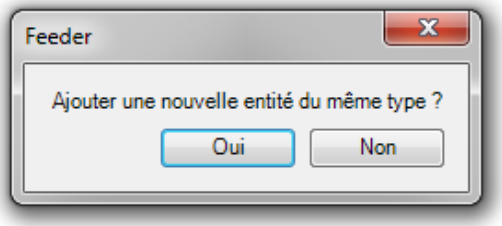


Figure 25 : Saisie d'un élément de même type

IV.2.2 Édition d'entités existantes

IV.2.2.1 Modification

Au sujet de la modification d'entités, il est plus compliqué de séparer le code du script en deux parties. En effet, l'interface a besoin d'informations sur l'entité à modifier pour être fonctionnelle (elle doit par exemple récupérer les valeurs d'attributs de l'objet et les présenter dans une boite de dialogue).

Pour commencer, l'application vérifie qu'une entité (et une seule) est sélectionnée. Les informations de position et d'attributs sur cet objet sont alors récupérées dans la base de données. La programmation de cette étape représente la difficulté majeure dans le développement de l'outil. En effet, pour accéder à ces informations, il est nécessaire de

parcourir l'ensemble de la hiérarchie des classes d'ArcObjects™, depuis les couches présentes dans ArcGIS® jusqu'à l'entité sélectionnée, (diverses classes intermédiaires rendent cette hiérarchie relativement complexe). De plus, l'aide en ligne d'ArcObjects™ pour C#.NET est relativement peu documenté sur ce sujet. En revanche, son équivalent pour VBA propose un script répondant au problème. Une « traduction » de ce dernier a donc été effectuée pour l'implémenter en C# dans l'extension.

Les boites de dialogue de modification d'entité sont alors identiques à celles de saisie, hormis le fait que les divers champs sont déjà remplis avec leurs valeurs actuelles. De plus, une modification est apportée pour les entités linéaires afin de visualiser et modifier les informations géométriques facilement : les coordonnées des sommets sont affichées dans un tableau. Pour modifier un sommet, il suffit donc de changer les valeurs des cellules correspondantes. À noter que seulement des valeurs numériques peuvent être saisies, pour éviter de générer une erreur par la suite. Enfin, un clic droit sur ce tableau affiche un menu permettant d'accéder à des fonctions d'ajout et de suppression de sommet(s).

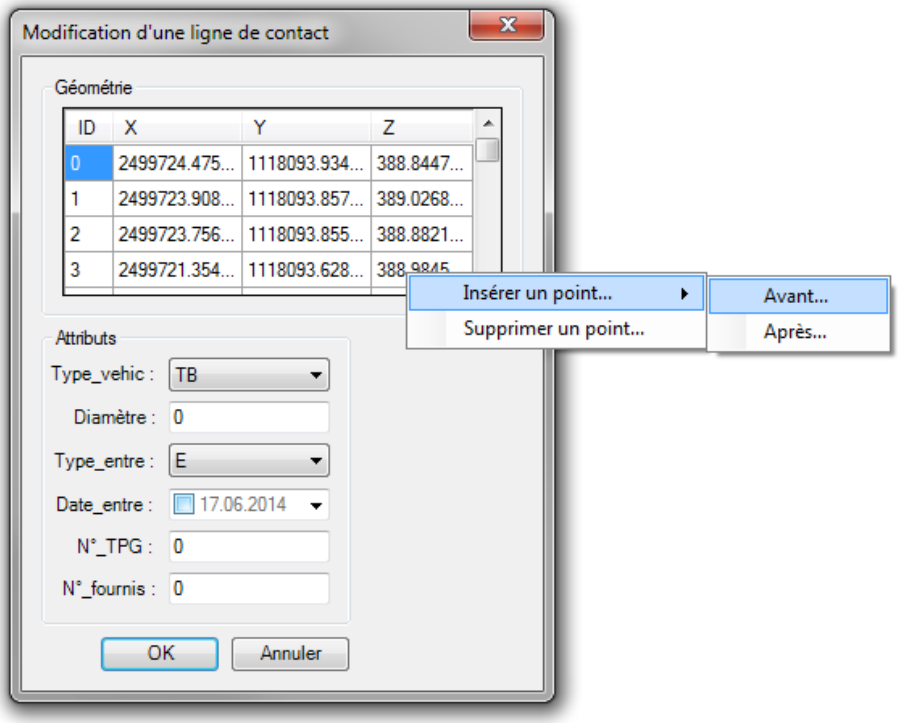


Figure 26 : Modification d'une ligne de contact

IV.2.2.2 Suppression

Pour l'outil de suppression d'entités, il suffit d'effacer de la géodatabase les éléments que l'utilisateur aura préalablement sélectionnés. Concernant l'interface, elle est de ce fait relativement simple puisqu'elle consiste simplement en un message d'avertissement et de demande de confirmation (la suppression étant définitive).

Ici, la solution est de passer par le Geoprocessor « DeleteFeatures » d'ArcObjects™. Celui-ci prend une couche en entrée. Si une sélection est présente dans cette couche, seules ces entités sont supprimées. Dans le cas contraire, tous les éléments sont effacés, ce qui nous pose problème. Il faut donc détecter en amont les couches comportant une sélection, et n'utiliser le Geoprocessor que sur celles-ci. Pour ce faire, on réutilise le même procédé qu'au paragraphe IV.2.2.1 afin d'identifier les couches à traiter.

IV.3 Conclusion

Cet outil demandant un travail important, son développement est toujours en cours. A cette date, la suppression d'entités est entièrement fonctionnelle et les interfaces de saisie et de modification sont terminées.

La difficulté principale pour le développement de cet outil réside dans le fait que la structure des éléments de base de données dans ArcObjects™ est plus complexe que celle présentée habituellement en SIG (y compris ArcGIS®). Ainsi, on a l'habitude de dire qu'une entité est contenue dans une couche. Pour ArcObjects™, l'entité fait partie d'une classe d'entité, elle-même représentée par un objet « FeatureLayer » (litt. « Couche d'entités), qui est affiché dans une couche de notre SIG. Autre exemple avec les sélections : dans un logiciel de SIG, on assimile naturellement une sélection à un ensemble d'entités. Avec ArcObjects™, une sélection est elle-même un objet qui va pointer vers les entités sélectionnées. Ceci est dû non seulement au langage de programmation C# qui est orienté objet, mais également à la volonté d'Esri® de réaliser un outil performant et permettant d'accéder aux fonctionnalités les plus avancées d'ArcGIS®.

Conclusion générale

Mettre en place un SIG ne se résume pas à rassembler de la donnée géographique dans une même application. Pour livrer un produit de qualité au client, il est nécessaire d'assurer la qualité de ces données et de leur représentation, ainsi que d'adapter l'outil aux besoins spécifiques du client. À cela, il faut ajouter les difficultés apportées par l'utilisation de la 3D dans ce projet. En effet, les SIG 3D sont encore peu courants de nos jours, et les réalisations dans ce domaine restent rares. Ainsi, la mise en place du SIG 3D des éléments aériens des lignes des TPG constitue un travail novateur.

Compte tenu de la méthode utilisée pour relever et modéliser ces éléments, des contrôles topologiques étaient indispensables pour respecter le critère de qualité des données évoqué plus haut. Face à la quantité d'entités à traiter, l'automatisation de la détection et de la correction des erreurs de topologie était également obligatoire. Basé sur les travaux de Florian GANDOR, l'élaboration d'une application en Python™ a permis de répondre à ce besoin. Certains compromis ont cependant été nécessaires durant le développement de l'outil. En effet, l'application restant interne à HKD Géomatique et ne devant servir qu'occasionnellement, le choix du langage de programmation a privilégié la rapidité de développement plutôt que la vitesse de traitement. De plus, certaines erreurs ne peuvent être que détectées, et nécessitent ensuite l'intervention d'un opérateur pour les corriger. Ce dernier point pourrait ainsi faire l'objet d'une étude spécifique pour réduire au possible le nombre de corrections à effectuer manuellement.

L'étape suivante de ce travail a été la définition de la symbologie 3D et son application dans ArcGIS®. L'élaboration des différents symboles a nécessité un véritable travail de réflexion, afin d'obtenir une représentation des divers éléments suffisamment proche de la réalité pour les identifier facilement, sans toutefois pousser la modélisation au point d'y allouer plus de temps que nécessaire et de surcharger la visualisation de la base de données. Pour finir, des applications annexes ont été réalisées afin d'exploiter au mieux cette symbologie dans ArcScene™, privilégié pour son travail en trois dimensions. En revanche, l'affichage des symboles n'est pas convaincant dans le cas d'une éventuelle visualisation sous ArcMap™ (en 2D), ce qui nécessiterait une nouvelle phase de développement pour résoudre ce problème.

Les TPG ayant besoin que la liaison entre ce SIG et le monde du DAO soit assurée, il a ensuite été nécessaire de se pencher sur les possibilités d'export en DXF. Un outil spécifique a donc été réalisé, et permet d'obtenir un résultat harmonisé avec les plans de pose en possession des TPG. Bien que parfaitement fonctionnel, l'outil pourrait faire l'objet d'améliorations : ajouter la possibilité d'exporter uniquement une partie de la base (actuellement, la totalité de la base est traitée d'un bloc), et le faire fonctionner en arrière-plan pour éviter de figer ArcGIS® pendant l'export. La possibilité d'exporter en PDF a également été abordée, mais les pistes explorées n'ont pas permis d'obtenir un meilleur résultat que celui déjà présent dans ArcGIS®.

Enfin, les opérations de mises à jour revenant aux techniciens des TPG, peu habitués à utiliser un tel outil, des fonctionnalités simples et intuitives ont dû être développées pour leur permettre de réaliser simplement cette tâche. L'idée est de limiter au maximum la possibilité d'erreurs en contrôlant les informations renseignées par l'opérateur au fur-et-à-mesure de leur saisie et par un système de listes déroulantes lorsqu'un choix doit être effectué parmi plusieurs possibilités. Ce travail étant relativement conséquent, il est toujours en développement au moment de la rédaction de cet écrit. Ainsi, la totalité de l'interface de l'outil a été réalisée, mais le traitement effectif des données reste à programmer.

Pour finir, participer à ce projet aura été pour moi une expérience très enrichissante. En effet, je devais apporter des éléments de réponses à quatre objectifs différents, ce qui a demandé une gestion efficace de mon activité pour traiter au mieux chaque partie du projet. Au vue de l'ampleur de ce projet et du temps alloué par un TFE, le but était avant tout de proposer des solutions fonctionnelles répondant aux différentes problématiques, sans être forcément les plus optimisées. Le résultat constitue donc une réponse de base aux objectifs, tout en permettant une future évolution de ces solutions.

Bibliographie

Adobe Systems Inc. *Document management – Portable document format – Part 1: PDF 1.7*, [en ligne]. 2008, 756 p. Disponible sur : http://wwwimages.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/PDF32000_2008.pdf (consulté le 20/05/2014).

Esri®. ArcGIS® Resource Center – ArcObjects™ SDK 10 Microsoft .NET Framework, [en ligne]. Disponible sur : http://help.arcgis.com/en/sdk/10.0/arcobjects_net/ao_home.html (consulté le 05/05/2014).

Esri®. ArcGIS® Resource Center – ArcObjects™ SDK 10 Visual Basic for Applications, [en ligne]. Disponible sur : http://help.arcgis.com/en/sdk/10.0/vba_desktop/ao_home.html (consulté le 02/06/2014).

GeoRezo – Le Portail Géomatique. GeoRezo, [en ligne]. Disponible sur : http://georezo.net (consulté le 05/05/2014).

SDXF – Python™ Library for DXF. In : Kellbot. Kellbot!, [en ligne]. Disponible sur http://www.kellbot.com/sdxf-python-library-for-dxf (consulté le 15/04/2014).

Kummler+Matter SA. Catalogue lignes contact, [en ligne]. Disponible sur : http://catalog.kuma.ch/fr (consulté le 09/05/2014).

LACHANCE Bernard. *Développement d'une structure topologique de données 3D pour l'analyse de modèles Géologiques*, [en ligne]. Mémoire de maîtrise en sciences géomatiques, Laval : Faculté des études supérieures de l'Université Laval, 2005, 116 p. Disponible sur : http://archimede.bibl.ulaval.ca/archimede/fichiers/22501/22501.pdf (consulté le 05/02/2014).

LAWHEAD Joel. pyshp – Python™ Shapefile Library, [en ligne]. Disponible sur : http://code.google.com/p/pyshp (consulté le 13/05/2014).

MENU Laurène. *Réalisation d'un cadastre aérien en 3D sur le canton de Genève : Mise en application au cas des lignes de Transports Publics Genevois*. Mémoire d'ingénieur CNAM spécialité Géomètre et Topographe, Le Mans : École Supérieure des Géomètres et Topographes, 2011, 96 p.

MINERY Cédric. *Des données 3D pour les architectes, urbanistes et paysagistes*, [en ligne]. Mémoire d'ingénieur INSA spécialité Topographie, Strasbourg : Institut National des Sciences Appliquées, 2011, 69 p. Disponible sur : http://eprints2.insa-strasbourg.fr/928/1/Mémoire_Cedric_MINERY.pdf (consulté le 05/05/2014).

OpenClassrooms. OpenClassrooms, [en ligne]. Disponible sur : http://fr.openclassrooms.com (consulté le 09/05/2014).

RETZLAFF Jimmy et al. Python™ : py2exe, [en ligne]. Disponible sur : http://www.py2exe.org (consulté le 05/03/2014).

Stack Overflow Inc. Stack Overflow, [en ligne]. Disponible sur : http://stackoverflow.com (consulté le 02/06/2014)

Liste des figures

Liste des tableaux

Table des annexes

Annexe 1 :
Détermination de l'ordre des contrôles topologiques

État initial

Rang	Contrôles	
	Fixe	Mobile
	Parafoudre	Câble électrique
	Boîtiers	Câble électrique
	Antenne	Câble électrique
	Ligne contact	Câble électrique
	Feeder	Câble électrique
	Console	Câble électrique
	Scellement	Console
	Isolateur de haubans	Console
	Isolateur	Console
	Feu	Console
	Parafoudre	Console
	Hauban	Console
	Hauban	Feeder
	Isolateur de haubans	Feeder
	Isolateur	Feeder
	Sectionneur	Feeder
	Scellement	Hauban
	Isolateur de haubans	Hauban
	Feu	Hauban
	Parafoudre	Hauban
	Isolateur	Hauban
	Panneau	Hauban
	Hauban	Ligne contact
	Isolateur de haubans	Ligne contact
	Isolateur	Ligne contact
	Sectionneur	Ligne contact
	Aiguillage	Ligne contact
	Antenne	Ligne contact
	Ligne contact	Pince de courbe
	Hauban	Protection
	Ligne contact	Protection
	Isolateur	Suspension
	Sectionneur	Suspension
	Aiguillage	Suspension
	Isolateur de haubans	Suspension
	Hauban	Suspension
	Ligne contact	Suspension
	Pince de courbe	Suspension
	Câble électrique	Suspension
	Croisement	Suspension
	Croisement	Tube
	Ligne contact	Tube
	Sectionneur	Tube
	Aiguillage	Tube

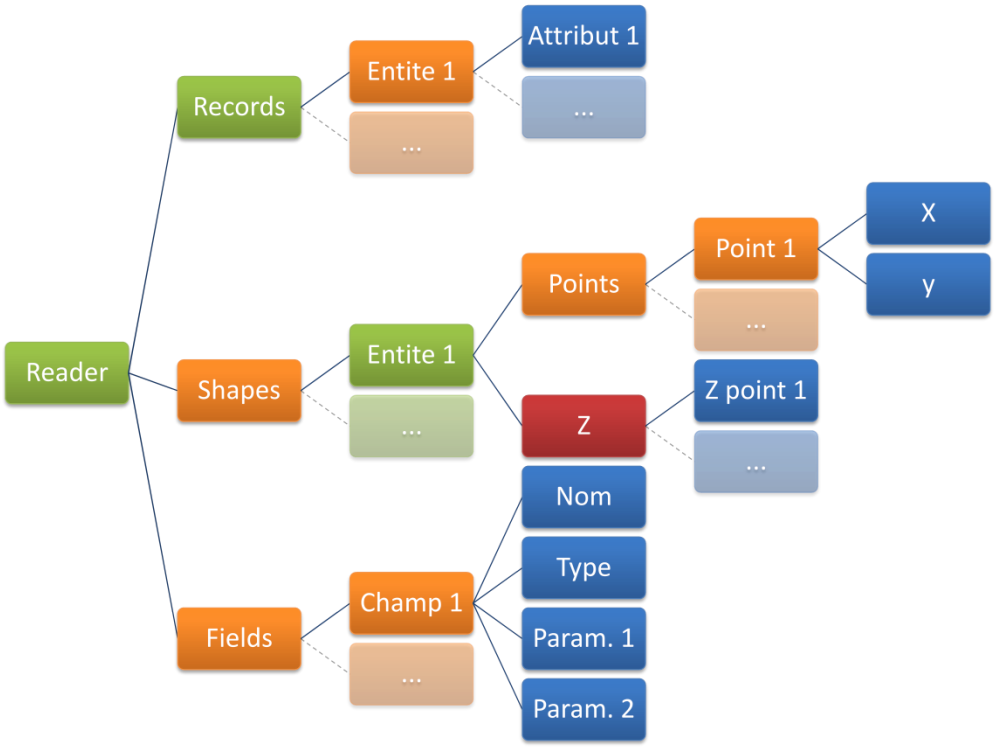
État en cours de traitement

Rang	Contrôles	
	Fixe	Mobile
1	Feu	Hauban
1	Isolateur	Hauban
1	Isolateur de haubans	Hauban
1	Panneau	Hauban
1	Parafoudre	Hauban
1	Scellement	Hauban
2	Feu	Console
2	Hauban	Console
2	Isolateur	Console
2	Isolateur de haubans	Console
2	Parafoudre	Console
2	Scellement	Console
2	Hauban	Feeder
2	Isolateur	Feeder
2	Isolateur de haubans	Feeder
2	Sectionneur	Feeder
2	Aiguillage	Ligne contact
2	Antenne	Ligne contact
2	Hauban	Ligne contact
2	Isolateur	Ligne contact
2	Isolateur de haubans	Ligne contact
2	Sectionneur	Ligne contact
	Parafoudre	Câble électrique
	Boîtiers	Câble électrique
	Antenne	Câble électrique
	Ligne contact	Câble électrique
	Feeder	Câble électrique
	Console	Câble électrique
	Ligne contact	Pince de courbe
	Hauban	Protection
	Ligne contact	Protection
	Isolateur	Suspension
	Sectionneur	Suspension
	Aiguillage	Suspension
	Isolateur de haubans	Suspension
	Hauban	Suspension
	Ligne contact	Suspension
	Pince de courbe	Suspension
	Câble électrique	Suspension
	Croisement	Suspension
	Croisement	Tube
	Ligne contact	Tube
	Sectionneur	Tube
	Aiguillage	Tube

État final

Rang	Contrôles	
	Fixe	Mobile
1	Feu	Hauban
1	Isolateur	Hauban
1	Isolateur de haubans	Hauban
1	Panneau	Hauban
1	Parafoudre	Hauban
1	Scellement	Hauban
2	Feu	Console
2	Hauban	Console
2	Isolateur	Console
2	Isolateur de haubans	Console
2	Parafoudre	Console
2	Scellement	Console
2	Hauban	Feeder
2	Isolateur	Feeder
2	Isolateur de haubans	Feeder
2	Sectionneur	Feeder
2	Aiguillage	Ligne contact
2	Antenne	Ligne contact
2	Hauban	Ligne contact
2	Isolateur	Ligne contact
2	Isolateur de haubans	Ligne contact
2	Sectionneur	Ligne contact
3	Antenne	Câble électrique
3	Boîtiers	Câble électrique
3	Console	Câble électrique
3	Feeder	Câble électrique
3	Ligne contact	Câble électrique
3	Parafoudre	Câble électrique
3	Ligne contact	Pince de courbe
3	Hauban	Protection
3	Ligne contact	Protection
3	Aiguillage	Tube
3	Croisement	Tube
3	Ligne contact	Tube
3	Sectionneur	Tube
4	Aiguillage	Suspension
4	Câble électrique	Suspension
4	Croisement	Suspension
4	Hauban	Suspension
4	Isolateur	Suspension
4	Isolateur de haubans	Suspension
4	Ligne contact	Suspension
4	Pince de courbe	Suspension
4	Sectionneur	Suspension

Annexe 2 :
Structure des données importées avec pyshp



En vert sont représentées les instances de classes ; les cadres orange correspondent aux listes ; en rouge, on distingue les tuples ; le bleu représente les chaînes de caractères et les types numériques.

Les données sont importées par une classe Reader, subdivisée en trois entités : Records (qui contient les attributs), Shapes (qui contient la géométrie) et Fields (qui contient les informations sur les attributs : nom, type, longueur, etc.).

Annexe 3 :
Exemple de fiche du catalogue de Kummler+Matter SA

Corps isolé type 1
Référence C1595-01

Données techniques

Dimensions

B: 72 mm
A: 170 mm

Matériel et poids

Poids: 0.335 kg

Charges

Charge de rupture vertical: 20 kN
Charge de rupture: 20 kN

Normes appliquées

-

Fonction

-

Spécifications

Tension de service: max. 1.5 kV

Accessoires

-

Remarques

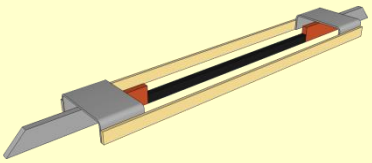
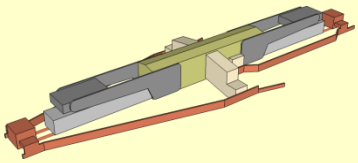
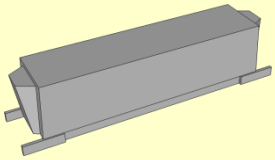
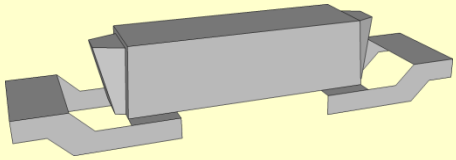
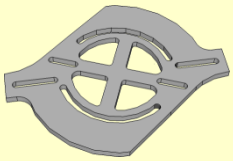
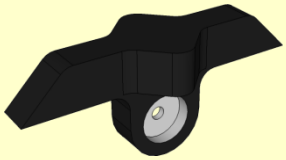
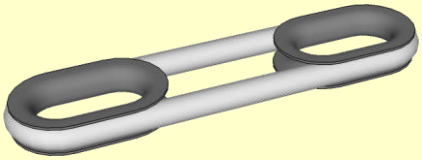
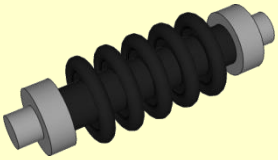
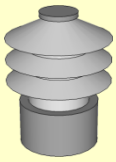
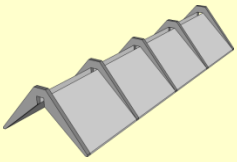
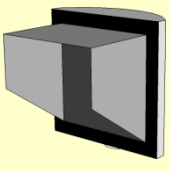
- Isolateur synthétique noir
- Armatures de raccord en acier inox

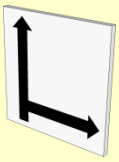
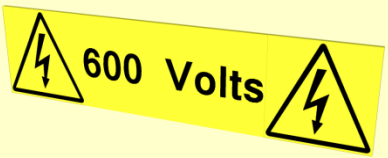
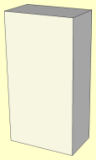
Kummler+Matter
UNE ENTREPRISE DU GROUPE ALPIQ

Kummler+Matter SA
Hohlstrasse 176
Case postale
CH-8026 Zurich

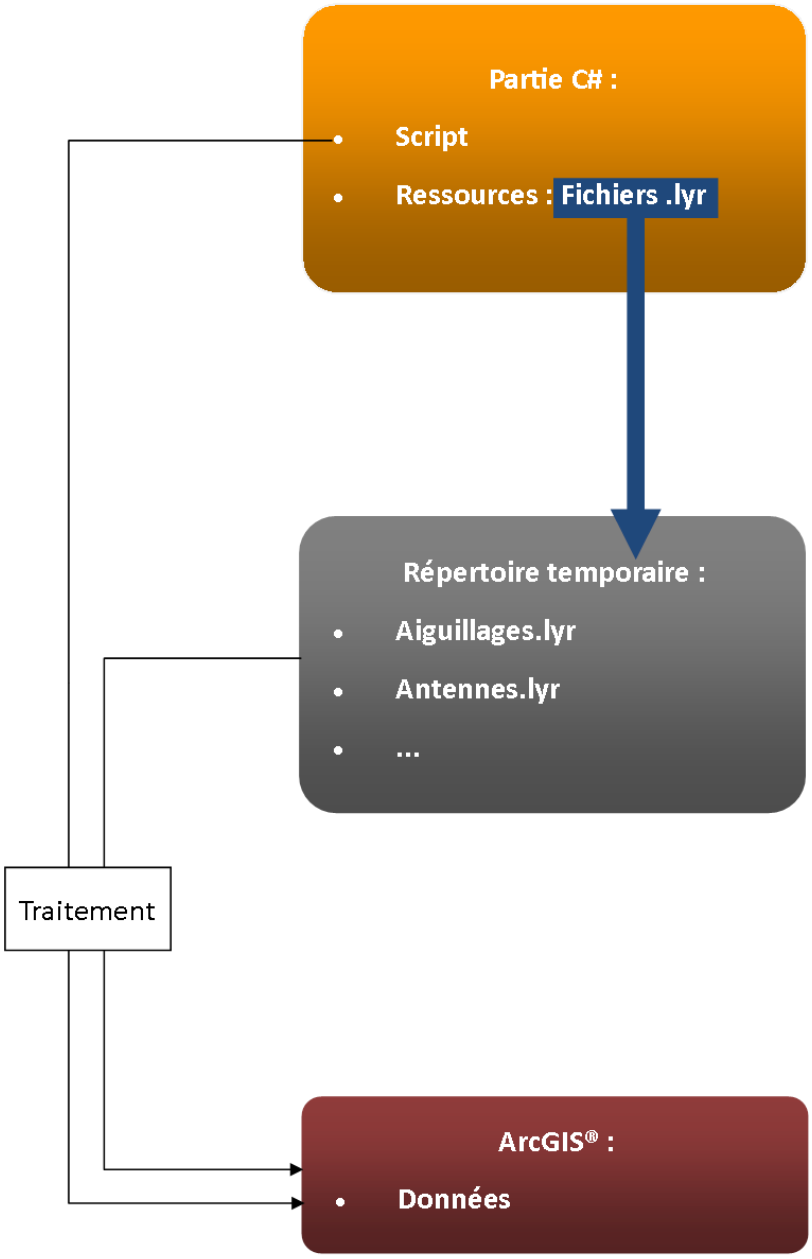
Téléphone +41 44 247 47 47
Téléfax +41 44 247 47 77
kuma@kuma.ch
www.kuma.ch

Annexe 4 :
Table des symboles 3D

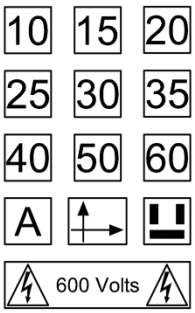
 Sectionneur sans coupure	 Sectionneur avec coupure
 Aiguillage avec alimentation 24 V	 Aiguillage mécanique
 Croisement	 Antenne
 Isolateur composite	 Isolateur porcelaine
 Boucle isolante	 Isolateur CFF
 Parafoudre	 Guide-perche
 Scellement	 Feu d'aiguillage

 Panneau d’aiguillage (TB)	 Panneau d’aiguillage(TW)
 Panneau de calage	 Panneau de sectionneur
 Panneau de limite de vitesse (10 km/h)	 Panneau de limite de vitesse (15 km/h)
 Panneau de limite de vitesse (20 km/h)	 Panneau de limite de vitesse (25 km/h)
 Panneau de limite de vitesse (30 km/h)	 Panneau de limite de vitesse (35 km/h)
 Panneau de limite de vitesse (40 km/h)	 Panneau de limite de vitesse (50 km/h)
 Panneau de limite de vitesse (60 km/h)	 Panneau de signalement de danger
 Contrepoids	 Boitier

Annexe 5 :
Organisation de l'outil de gestion de l'affichage



Annexe 6 :
Table des symboles DXF



Panneaux

Antenne

Retenue de courbe

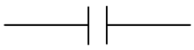
Croisement

Croisement réglable

Aiguillage

Sectionnement pour ligne tramway

Sectionnement pour ligne trolleybus



Sectionnement feeder

Pince de passage

Croisement de lignes de contact à hauteurs différentes

Isolateur

Isolateur de haubans

Amarrage pour fil de contact ou feeder

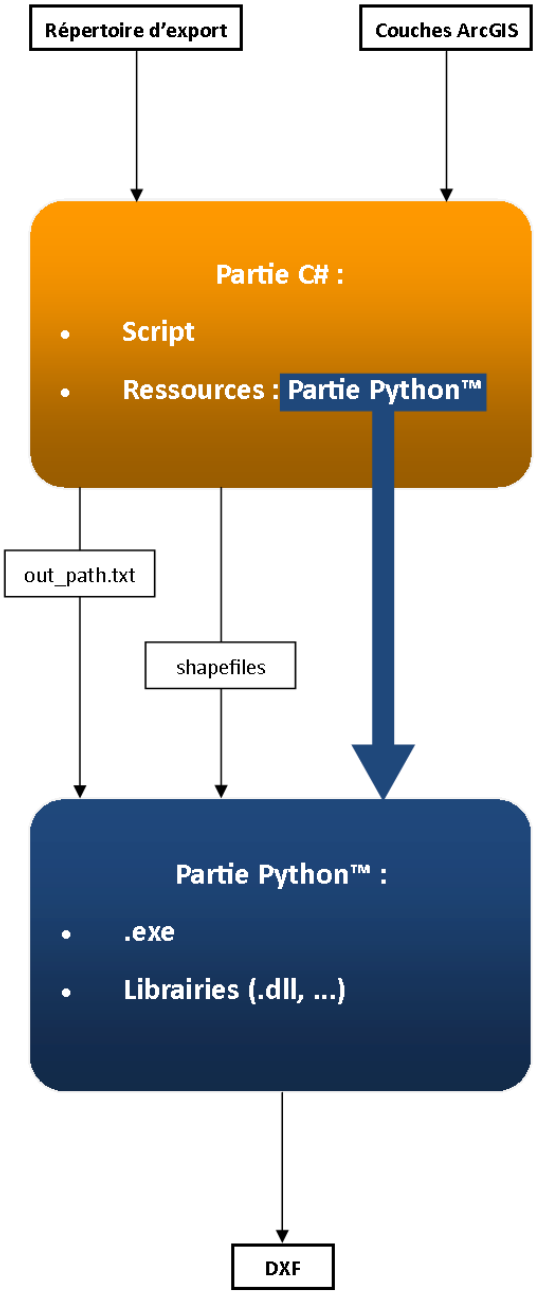
Scellement

Parafoudre

Contrepoids

Boitier

Annexe 7 :
Organisation de l'outil d'export DXF



Annexe 8 :
Poster scientifique

Annexe en page suivante.

Mise en place du SIG 3D des éléments aériens des lignes des Transports Publics Genevois

Introduction

Les transports publiques genevois (TPG) souhaitant disposer d'une base de données des éléments aériens de leurs lignes de tramways et de trolleybus, HKD Géomatique SA a été mandatée pour les relever et les consigner sous la forme d'un SIG 3D.

L'acquisition des données a été effectuée par mobile laser scanning (MLS), sous forme de nuages de points et d'images géoréférencées. Les différentes entités on ensuite été modélisées sous forme de points et de polylignes via le logiciel Trimble® Trident-3D Analyst®.


Véhicule d'acquisition

La mise en place du SIG 3D final se traduit alors par quatre objectifs :
- réaliser des outils de contrôles topologiques en 3D ;
- concevoir la symbologie de la base ;
- mettre en place des outils d'export, principalement vers le monde du dessin assisté par ordinateur (DAO) ;
- proposer des outils simples permettant la mise à jour de la base par les équipes des TPG.

Contrôles topologiques en 3D

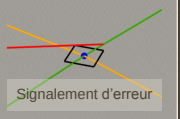
Face à la quantité d'objets à contrôler (environ 38 500 ponctuels et 428 kilomètres de linéaires), on crée un script Python™ pour automatiser la tâche.

Pour chaque couple d'objets à contrôler, une entité est définie comme fixe et l'autre comme mobile. Si la distance entre les deux est inférieure au seuil de tolérance choisi, l'élément mobile est corrigé par rapport à l'entité fixe.

L'ordre dans lequel les entités sont contrôlées a son importance puisque pour être défini comme entité fixe, un objet doit d'abord avoir été contrôlé.

Implémenter la notion de «plus proche» permet de corriger une entité erronée par rapport à l'entité fixe la plus proche lorsque plusieurs corrections sont possibles.

Lorsqu'il faut rajouter un sommet à un linéaire, le programme réalise souvent des fausses corrections. Dans ce cas, ces dernières ne sont pas appliquées mais seulement signalées.


Signalement d'erreur

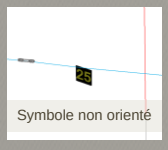
La correction par rapport aux éléments extérieurs à la base de données est facultative car leur modélisation n'est pas forcement à jour ou suffisamment précise.


Mât mal positionné

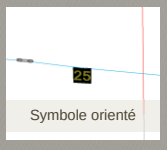
Symbologie

Seul les symboles ponctuels sont à créer. Pour les linéaire, un simple code couleur est utilisé. Leur création est réalisée sous Trimble® SketchUp®, à partir des fiches du catalogue du fournisseur.

Un script Python™ est réalisé pour gérer l'orientation des symboles. Celui-ci calcul leur paramètres d'orientation et les inscrits dans des nouveaux champs de la base de données. ArcGIS® permet ensuite d'orienter les symboles selon ces champs.

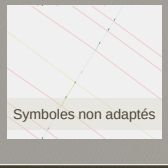


Symbole non orienté



Symbole orienté

Pour une visualisation en 2D, il convient d'adapter la symbologie. Un outil est donc développé et intégré à ArcGIS® afin de gérer cette tâche.

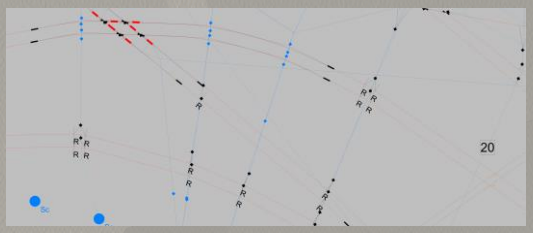

Symboles non adaptés

Symboles adaptés

Outils d'export

Le DXF a été choisi comme format d'export vers le monde du DAO. ArcGIS® permet un export dans ce format, mais ne conserve pas la symbolique. Un outil a donc été développé. Le traitement s'effectue en Python™, mais l'outil est intégré dans ArcGIS® par une application en C# .NET qui exécute le script de traitement.

Une symbologie spécifique est appliquée, sous forme de blocs insérés à l'emplacement des ponctuels. Ces blocs sont directement définis dans le code Python™. Pour rester cohérent avec les plans de montage des TPG, notre symbologie s'inspire de ces derniers.



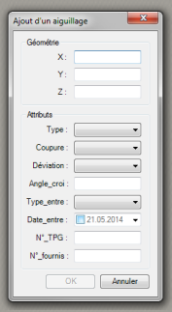
En parallèle, des recherches ont été menées pour exporter des vues en PDF. Cependant, les pistes explorées n'ont pas abouti à un meilleur résultat que l'outil déjà présent dans ArcGIS®.

Outils de mise à jour

Les futures mises à jour de la base seront réalisées par les techniciens des TPG, peu habitués à manipuler ce type d'outils. L'objectif est donc de remplacer ceux déjà présents dans ArcGIS® par une application plus intuitive et limitant au maximum les possibilités d'erreurs. Le travail effectué jusqu'à présent se focalise uniquement sur l'interface de l'outil.

Celui-ci doit permettre l'ajout de nouvelles entités, la modification et la suppression d'entités existantes.

Rappel de la loi de Murphy : «S'il y a plus d'une façon de faire quelque chose, et que l'une d'elles conduit à un désastre, alors il y aura quelqu'un pour le faire de cette façon». De ce fait, on contrôle au maximum la saisie des informations par l'utilisateur :



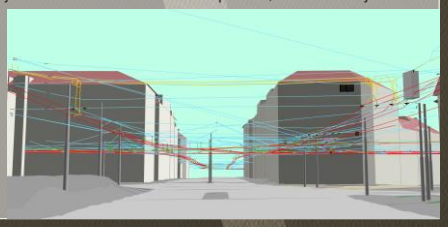
Les champs devant contenir des valeurs prédéfinies se remplissent à l'aide de listes déroulantes pour forcer le choix parmi ces valeurs uniquement.

Les boites de dialogues ne peuvent être validées que lorsque toutes les informations on été correctement saisies (nombres pour les coordonnées, champs obligatoires remplis,…).

Conclusion

Pour mettre en place un SIG, il est nécessaire d'assurer la qualité des données et de leur représentation, et adapter l'outil aux besoins spécifiques du client. De plus, l'utilisation de la 3D dans ce projet ajoute des difficultés supplémentaires, car encore peu courante dans le domaine du SIG.

Les trois premiers objectifs ont étés entièrement remplis. Bien que totalement fonctionnels, les outils réalisés restent toutefois modifiables si des améliorations venaient à être requises. Les outils de mise à jour étant relativement complexes, ils sont toujours en phase de développement. Dans le cadre du TFE, il avait été décidé de se focaliser sur l'interface uniquement. Étant entièrement réalisée à ce jour, il ne reste plus qu'à implémenter le traitement des données lui-même.



hkdgéomatique

Poster réalisé en juin 2014 par Joris VIDALENCHE
suite au travail de fin d'études réalisé chez HKD Géomatique SA de février à juin 2014.
Contact : j.vidalenche@orange.fr

le cnam esgt
École supérieure des géomètres et topographes

Mise en place du SIG 3D des éléments aériens des lignes des Transports Publics Genevois.

Mémoire d'Ingénieur C.N.A.M., Le Mans 2014

RESUMÉ

La mise en place du SIG 3D des éléments aériens des Transports Publics Genevois s'est déroulée en quatre étapes : le développement d'un outil de contrôles topologiques en 3D de la base de données préalablement établie, la définition d'une symbologie 3D et la gestion de son affichage dans ArcGIS®, la conception d'un outil d'export de la base en DXF, et la réalisation d'outils de mise à jour.

Une application indépendante a été développée pour regrouper les outils de contrôles topologiques et de préparation des symboles, qui sont des outils de conception du SIG. Les fonctionnalités d'export et de mise à jour (qui servent à l'exploitation des données) ont été intégrées à ArcGIS® via une extension.

Mots clés : TPG, SIG, 3D, topologie, symbologie, export, mises à jour.

SUMMARY

The four steps to set up the 3D GIS of the overhead elements of the Transports Publics Genevois were as follows: the development of a topological checks tool in 3D of the database previously established, the definition of a 3D symbology and the display management of these symbols in ArcGIS®, the design of an export tools from the database to DXF, and the creation of updates tools.

A standalone application has been developed in order to regroup topological checks and symbols preparation tools, which are used for the design of this GIS. Export and updates functionality (used for data exploitation) has been integrated with ArcGIS® via an add-in.

Key words: TPG, GIS, 3D, topology, symbology, export, updates.