UNIVERSITÉ DE LIÈGE
Faculté des sciences
Sciences géographiques

Développement d'une géocodification 3D pour la modélisation du bâti

Mémoire présenté par
Jean-Paul KASPRZYK
Pour l'obtention du titre de
Licencié en sciences géographiques
Option géomatique et géométrologie

Année académique
2007-2008

Bien que ce document soit le rapport définitif d'un projet individuel, il n'aurait pas pu se réaliser sans collaborations ni soutiens actifs.

Je souhaite profiter de ces quelques lignes pour remercier tout d'abord mon promoteur, Monsieur Roland BILLEN, pour son suivi, son soutien, ainsi que ses précieux conseils et remarques lors de la réalisation de ce mémoire.

Je tiens aussi à remercier Monsieur Pierre HALLOT pour le temps qu'il m'a consacré lors de mes journées passées sur le terrain ainsi que Monsieur Benoît SCHUMACKER pour ses explications concernant la programmation en langage PERL.

Un grand merci à ma famille et mes amis pour leur soutien, en particulier Marie LEKANE, Jean-Roger KASPRZYK, Hyun FRERE et Marie PEZZETTI.

Enfin, je remercie également les différents lecteurs de ce mémoire.

Table des matières

1. Introduction

1.1. La géocodification : généralités

La géocodification est une méthode appliquée au levé avec station totale permettant la génération automatique d'un plan à partir de codes alphanumériques associés aux mesures. Autrement dit, après avoir appliqué un code spécifique à chaque point levé (dépendant de la nature du point en question), un algorithme permet le tracé des symboles (ponctuels, linéaires ou zonaux) qui constituent le plan.

La géocodification est donc un puissant outil présentant trois grands avantages au niveau du levé :

- Un gain de temps considérable au niveau du post-traitement
- Un carnet de terrain allégé
- Un guide pour la méthode de levé des différents objets

De nombreux logiciels de CAO (conception assistée par ordinateur) permettent l'utilisation d'une géocodification générant un plan en 2D :

- Covadis
- Topofast
- Strada Polaris
- Mensura
- *Etc…*

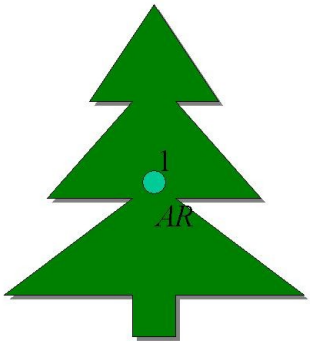
En revanche, les recherches effectuées n'ont révélé aucune trace de documentation concernant une géocodification générant un modèle 3D. Cela peut s'expliquer par le fait que les levés 3D sont généralement effectués avec d'autres outils que la station totale en raison du nombre de points à mesurer beaucoup plus important. Cependant, nous verrons dans le point 1.3 que le développement d'une géocodification 3D présenterait un intérêt potentiel non-négligeable dans certaines circonstances.

1.2. La géocodification 2D : principes de base

La plupart des géocodifications 2D existantes utilisent les mêmes principes de base que nous allons décrire brièvement.

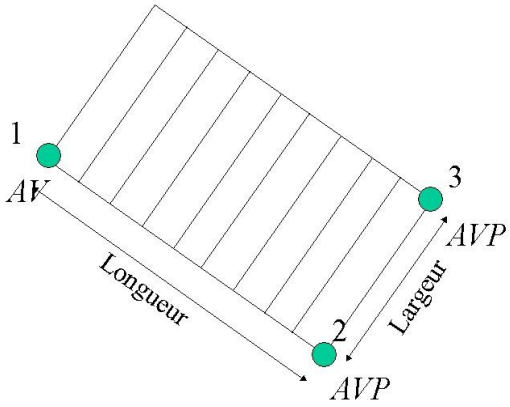
Les symboles non-orientés et non-dimensionnés représentant des entités ponctuelles sont associés à un seul code permettant la génération du dit symbole. Par exemple (figure 1.1), imaginons le code « AR » pour représenter un arbre levé par un point d'insertion : son centre. Remarquons que dans l'exemple de la figure 1.1, le symbole a une orientation mais qui n'est pas définie par l'objet levé (tous les symboles « arbre » sont orientés de la même façon sur le plan).

Figure 1.1. Illustration de la codification 2D d'une entité ponctuelle



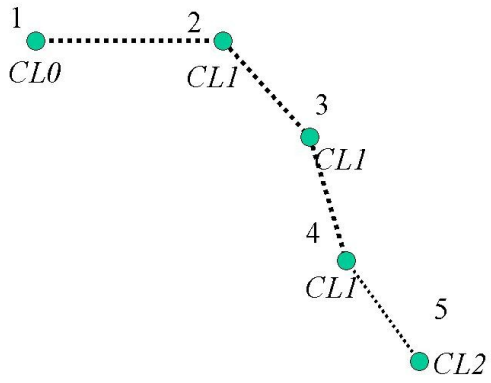
Les symboles orientés et/ou dimensionnés représentant des entités zonales sont associés à un code nécessitant des paramètres. Par exemple (figure 1.2), imaginons un avaloir auquel est associé le code « AV » qui génère un rectangle aux dimensions et à l'orientation dépendant de l'objet levé. Trois points doivent donc être saisis : un point d'insertion (code « AV ») et deux points paramétriques (code « AVP »).

Figure 1.2. Illustration de la codification 2D d'une entité zonale (avaloir)



Enfin, les symboles linéaires peuvent être utilisés pour représenter aussi bien les entités linéaires que zonales (par contour). La codification utilisée est la liaison : il s'agit d'une succession de segments reliés deux par deux partant d'un point de départ et se terminant sur un point d'arrivée (qui peut être le point de départ). Par exemple (figure 1.3), imaginons une clôture à laquelle est associé le code « CL ». Un paramètre d'ouverture (0) est associé au point de départ, un paramètre de liaison rectiligne est associé à un point quelconque de la clôture (1) et un point de fermeture (2) est associé au dernier point de la clôture. Une liaison permet donc de relier plusieurs points saisis de manière consécutive par des segments.

Figure 1.3. Illustration de la codification 2D d'une entité linéaire (clôture)



Les codifications 2D permettent l'utilisation d'un grand nombre de types de codes directement inspirés de ces principes. En pratique, l'utilisateur n'en utilise qu'une dizaine au maximum lors de son levé. En effet, utiliser trop de codes différents augmente fortement le risque d'erreur d'encodage et donc la génération d'un plan erroné. De plus, rappelons que le rôle de la codification est de faciliter le travail effectué lors d'un levé. Il faut donc qu'elle soit à la fois simple et efficace.

Pour finir, notons qu'une géocodification nécessite l'intervention de l'utilisateur dans sa conception. Une table de code permet à ce dernier d'associer un type de code à un symbole via un code pour stocker un objet dans une couche particulière du plan. Par exemple, le type de code « liaison » peut-être associé au symbole linéaire « clôture » via le code « CL », tous les objets levés via ce code étant stockés dans la couche du plan réservée aux clôtures. A la base, une géocodification est donc constituée de méthodes de levé associées à des syntaxes de code.

1.3. Utilité d'une géocodification 3D dans la modélisation 3D

Les deux méthodes d'acquisition de données utilisées actuellement pour la modélisation 3D sont la photogrammétrie et la lasergrammétrie, le levé traditionnel avec station totale et géocodification étant habituellement réservé au travail en planimétrie.

Ce point décrit brièvement les deux méthodes actuelles d'acquisition de données (photogrammétrie et lasergrammétrie) et leurs post-traitements respectifs suivi de la phase de modélisation 3D – commune aux deux méthodes. En fonction de ce qui aura été dit, nous discuterons ensuite des avantages apportés par une éventuelle géocodification 3D dans la modélisation 3D.

1.3.1. La lasergrammétrie

1.3.1.1. Acquisition de données

La lasergrammétrie exploite la technologie du scanner 3D. Cet appareil lève plusieurs millions de points en quelques minutes par balayages horizontaux et verticaux. Contrairement

au levé classique, on ne lève plus quelques points clés permettant de reconstruire la géométrie d'un objet mais l'objet dans sa globalité.

Les « scans » sont réalisés depuis différents points de vue de la scène à numériser. Le nombre de points de vue est proportionnel à la complexité et à la taille de l'environnement à lever. Nous obtenons ainsi un nuage de points pour chaque « scan ». Pour chaque point, des mesures d'angles et de distances permettent le calcul au moyen de formules trigonométriques de coordonnées locales dans un système propre à chaque « scan ». La précision des mesures s'étend de 1 mm à 10 mm.

Figure 1.4. Exemple de scanner : Leica HDS6000

(Source : http://www.leica-geosystems.com)

1.3.1.2. Post-traitement

Le post-traitement s'effectue en deux étapes : la consolidation et le nettoyage.

Connaissant les coordonnées locales des différents points levés depuis les différents points de vue, nous devons maintenant les transférer vers un système de coordonnées unique. Cette phase s'appelle la consolidation. La transformation d'un système de coordonnées tridimensionnel requiert trois paramètres de rotation et trois paramètres de translation. Trois points (ni colinéaires ni coplanaires) de chaque nuage au minimum permettent donc de résoudre le système d'équations. Davantage de points amènent à une résolution du système plus précise par moindre carré. Ainsi, une sélection de trois points homologues minimum par couple de « scans » dans les nuage de points permettent la consolidation. Notons que cette étape peut être réalisée automatiquement par la nouvelle génération de scanners topographiques au moyen de cibles. Ces cibles sont réparties autour de l'objet levé et sont reconnues en tant que points homologues par l'appareil (trois cibles communes doivent donc apparaître sur chaque couple de « scans »).

Ensuite, tous les éléments levés par le scanner ne faisant pas partie du futur modèle 3D doivent être supprimés. Il s'agit de la phase de nettoyage. Elle concerne aussi bien les objets ne faisant pas partie des objectifs du projet (exemple : la végétation) que le bruit de mesure (exemple : gouttes de pluie).

1.3.2 La photogrammétrie

1.3.2.1. Acquisition de données

« *La photogrammétrie est une technique qui consiste à mesurer la surface observée à partir de clichés acquis en configuration stéréoscopique, en utilisant d'une part la vision stéréoscopique pour mettre en correspondance les deux images, et d'autre part une modélisation mathématique de la géométrie de prise de vue.* » (www.wikipedia.org)

En fonction de l'objectif du projet nécessitant le levé, la photogrammétrie peut être aérienne ou terrestre. Dans le premier cas, une série de clichés est effectuée depuis un avion parcourant le terrain par bandes successives (avec un recouvrement entre clichés d'environ 60%). Dans le deuxième cas, des couples stéréoscopiques de clichés sont pris directement sur le terrain. Il va de soi qu'à partir d'un niveau de détail moyen, on privilégie la photogrammétrie terrestre à la photogrammétrie aérienne pour des raisons de précision (liée à la résolution géométrique des clichés) et de visibilité des détails. Notons que pour l'établissement de modèles urbains, la technique la plus efficace est la prise de clichés depuis un véhicule équipé de caméras. Dans les cas du levé de façades, par exemple, deux caméras forment une base stéréoscopique verticale de chaque côté du véhicule. La précision obtenue sur les mesures par photogrammétrie terrestre est de l'ordre de 2 millimètres.

1.3.2.2. Post-traitement

Le post-traitement concernant la photogrammétrie s'effectue en trois étapes : les orientations, l'extraction d'un nuage de points et le nettoyage.

Pour obtenir les coordonnées locales d'un point à partir d'un couple stéréoscopique de clichés, nous devons passer par l'orientation intérieure et l'orientation relative. L'orientation intérieure permet la transformation des mesures effectuées sur les photos dans un système 2D « utilisateur » vers un système calibré (et donc corrigé des différentes distorsions des clichés). L'orientation extérieure implique le placement des gerbes perspectives dans une position relative identique à celles qu'elles occupaient dans l'espace lors de la prise de vue. Concrètement, ces étapes sont réalisées en mesurant les marques de fond de chambre et neuf points homologues sur le couple stéréoscopique. Notons qu'il existe une troisième orientation (l'orientation absolue) en vue d'obtenir les coordonnées des points dans un système de coordonnées globales (exemple : Lambert 72) à appliquer également dans le cas de la lasergrammétrie.

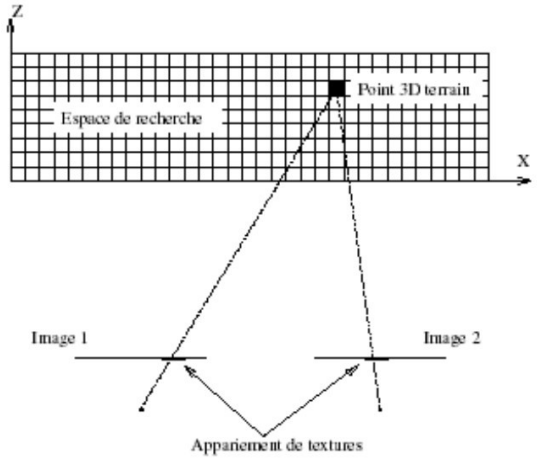
Une fois les paramètres des orientations connus, nous devons extraire un nuage de points. Pour décrire cette étape, nous nous appuierons sur les travaux réalisés sur des façades de bâtiments à l'Institut Géographique National de Saint-Mandé en France (PENARD, PAPARODITIS & PIERROT-DESEILLIGNY 2006).

Pour une scène levée, on obtient généralement une série de clichés où les parties communes apparaissent sur plus de deux photos. Cette redondance d'informations permet un calcul plus précis des coordonnées des différents points extraites des clichés ainsi qu'une meilleure fiabilité dans la sélection de points homologues (réalisée par un algorithme). Dans notre système de coordonnées, nous définissons un volume renfermant le ou les objet(s) à modéliser :

Xmin < X < Xmax
Ymin < Y <Ymax
Zmin < Z < Zmax

Dans ce volume de travail, un algorithme permet de retrouver, pour tous les points d'une image référence (pixels), l'ensemble des points homologues sur les images comprenant le point recherché. Cet algorithme est basé sur le calcul d'un coefficient de corrélation multi-images concernant les différents niveaux de gris de chaque point homologue d'un point de l'image référence. Une fois les points homologues retrouvés pour une image référence, nous pouvons les transformer en un nuage de points de coordonnées X, Y, Z connues et recommencer le travail sur une nouvelle image référence si la première ne comprenait pas tous les points qui nous intéressaient.

Figure 1.5. Illustration de l'algorithme visant l'obtention d'un nuage de points en photogrammétrie



(Source : PENARD, PAPARODITIS & PIERROT-DESEILLIGNY 2006)

Comme pour la lasergrammétrie, cette étape doit être précédée d'un nettoyage permettant à l'algorithme de travailler uniquement sur des points fiables. Par exemple, les vitres des fenêtres de bâtiments sont des points à éviter du fait des réflexions qui diffèrent d'une image à l'autre et de l'absence de texture significative. Le ciel est également à filtrer : les points du ciel sont caractérisés par un coefficient de corrélation (mentionné plus haut) très faible, ce qui permet à l'algorithme de les reconnaître. Les pixels pour lesquels la texture au voisinage est insuffisante doivent également être filtrés (la texture est mesurée par la variance des valeurs de niveau de gris du voisinage). En effet, dans le cas contraire, l'appariement avec des zones similaires serait relativement indéterminé.

Figure 1.6. Exemple de nuage de points obtenu à partir de la photogrammétrie (avec filtrage)

(Source : PENARD, PAPARODITIS & PIERROT-DESEILLIGNY 2006)

Remarquons que l'extraction du nuage de points telle que décrite précédemment s'inscrit dans un niveau de détail optimal. En effet, au final, presque tous les pixels homologues des images sont représentés dans le nuage de points. Le nuage de points ressemble donc à celui que l'on obtiendrait par lasergrammétrie. Dans le cas d'un levé 3D d'un niveau de détail moyen (formes générales des façades), seuls quelques points clés permettant la reconstructions géométriques des objets compris dans le modèle sont sélectionnés manuellement.

1.3.3. La modélisation

1.3.3.1. Généralités

Aussi bien dans le cas de la photogrammétrie que de la lasergrammétrie, le traitement des données renvoie un nuage de points associé au levé effectué. La modélisation permet de transformer ce nuage de points en un ensemble de formes représentant le terrain. En effet, un modèle 3D est constitué d'objets géométriques définissables par des équations mathématiques. En d'autres mots, nous devons passer d'un groupement de points plus ou moins dispersés à une série d'ensembles de points (formes géométriques). Par exemple, un cube peut être défini par six équations de plan borné par des arêtes.

Il existe différents types de modèles 3D dont voici quelques exemples :

- Modèle filaire : les objets sont décrits par des sommets et des arêtes connectées, sans informations sur les faces et les volumes des objets ;
- Modèle surfacique : un objet est représenté par un ensemble de faces elles-même définies par un ensemble de segments ;
- Les courbes et les surfaces paramétriques : ce modèle permet de représenter des objets à faces non-planes. Les objets sont restitués sous forme d'une succession de morceaux de courbes ou de mosaïques de carreaux de surfaces représentés de façon paramétrique ;

- Modèles volumiques : évolution des modèles filaires et surfaciques en prenant compte des propriétés géométriques et topologiques des objets ;
- Modèle CSG : les objets sont décrits à partir de primitives volumiques assemblées par des opérateurs logiques.

1.3.3.2. Segmentation

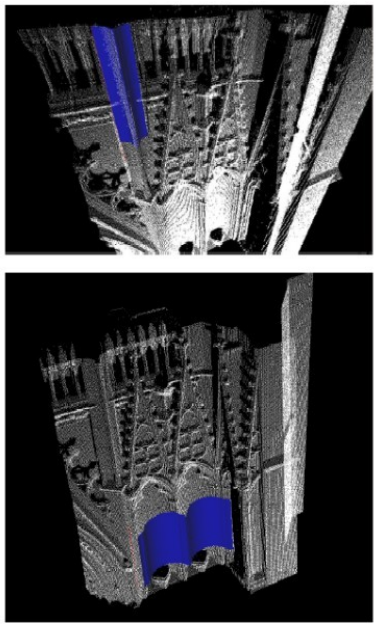
La segmentation est l'étape consistant à sélectionner les différents groupes de points représentant une primitive géométrique définissable dans un modèle 3D. Cette partie du travail peut être effectuée de façon automatique ou semi-automatique par des algorithmes ou manuellement par l'utilisateur.

1.3.3.3. Construction

Deux stratégies concernant la construction du modèle peuvent être envisagées : les primitives et la triangulation. Notons que ces deux techniques peuvent être combinées.

Concernant la construction par primitives, à chaque partie du nuage de points segmentée, nous associons une primitive définissable dans notre modèle 3D. Le choix des primitives est directement influencé par le type d'objet représenté par le modèle. Par exemple, le laboratoire MATIS de l'IGN de Saint-Madré propose trois types de primitive dans un contexte de levé architectural: des surfaces planes, des surfaces de révolution et des cylindres (DEVEAU, PAPARODITIS & PIERROT-DESEILLIGNY 2005). La partie du nuage de point segmentée pour représenter une primitive permet de retrouver les paramètres d'équation de cette dernière (frontières de la surface, axe de révolution, rayons de révolution, *etc*…). Ce type de construction est une approximation : la surface ne passe pas forcément par les points par lesquels elle est définie (ajustement).

Figure 1.7. Exemple de construction par primitive (cylindre)

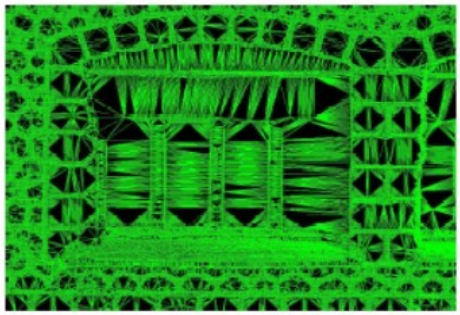


(Source : DEVEAU, PAPARODITIS, PIERROT-DESEILLIGNY 2005)

Concernant la triangulation, le nuage de points est associé à un réseau de triangles reliant les entités ponctuelles deux à deux, en évitant les angles trop aigus et trop obtus. La construction de la triangulation est effectuée de proche en proche en respectant le critère de Delaunay : trois points sont voisins au sens de Delaunay et forment un triangle si le cercle passant par ces trois points ne contient aucun autre point candidat. Le critère de Delaunay n'est valable que dans un plan (intervention de cercles). C'est pourquoi tous les points du nuage sont d'abord projetés sur un plan de référence afin de vérifier la condition de voisinage. Ce type de construction est une interpolation : la surface passe par tous les points par lesquels elle est définie.

Figure 1.8. Exemple de construction par triangulation (photogrammétrie)

(a) Partie d'image



(b) Triangulation correspondante

(Source : PENARD, PAPARODITIS & PIERROT-DESEILLIGNY 2006)

Cette dernière méthode présente le gros avantage d'être assez facilement automatisable et d'être applicable à tous les objets (qu'ils soient à faces planes ou non planes). Cependant, elle comprend également plusieurs défauts :

- Dépendance de la densité des points : les discontinuités dans le nuage de points peuvent être remplies par des triangles.
- La présence de mesures redondantes implique des triangles supplémentaires.
- Il n'y a pas de description orientée objet de la scène.

Malgré son automatisation, cette technique nécessite donc un traitement manuel assez conséquent.

Remarquons que la photographie peut être utilisée pour placer des textures sur les faces construites dans la modélisation : raison pour laquelle les scanners sont souvent équipés d'une caméra.

1.3.4. Intérêt de la géocodification 3D dans la modélisation

Actuellement, la photogrammétrie est utilisée pour des levés de niveau de détail faible (photogrammétrie aérienne), moyens et élevés (photogrammétrie terrestre), et la lasergrammétrie pour des niveaux de détail élevés et très élevés. Sachant qu'un même levé peut combiner les deux techniques (la photogrammétrie pour les formes de base et la lasergrammétrie pour les détails complexes), quel est l'intérêt apporté par une géocodification 3D dans ce contexte ?

1.3.4.1. Domaine d'application : les niveaux de détails

Un levé avec station totale et géocodification 3D s'inscrit dans des niveaux de détail faibles et moyens. En effet, contrairement à la photogrammétrie et à la lasergrammétrie, le nombre de points levés sur le terrain est assez restreint (quelques centaines pour une journée de travail). Imaginons que pour un levé de niveau de détail faible, aucune photographie aérienne ne soit à disposition. Si le terrain à lever est de petite taille, l'établissement d'une mission de vol semble inadapté au projet. Dans ce cas, le levé de quelques points à la station totale est tout à fait approprié.

Un levé moyen est réalisable avec une géocodification, pour autant que les objets à représenter ne soient pas trop complexes. En effet, puisque le processus de description de la géométrie des objets a lieu sur le terrain, une géocodification ne peut pas se permettre d'être trop compliquée, au risque d'être une importante source d'erreurs. En pratique, une dizaine de codes de base différents au maximum doivent être assimilés par l'utilisateur. A cela, nous pouvons ajouter quelques codes plus spécifiques et plus complexes à utiliser en dernier recours (par exemple, pour les objets à faces non-planes). Ainsi, dans le cas d'un levé de niveau moyen avec objets complexes, nous pourrions imaginer la combinaison des techniques de la photogrammétrie (pour les formes complexes et éventuellement les textures) et de la géocodification (pour les formes simples).

Dans le cas de niveaux de détails élevés, effectué habituellement par lasergrammétrie uniquement ou combinée à la photogrammétrie, nous pourrions imaginer une combinaison entre géocodification (formes de base) et lasergrammétrie (détails complexes) ou pourquoi pas une combinaison géocodification-photogrammétrie-lasergrammétrie. Dans ce dernier cas de figure, la photogrammétrie permettrait le levé de formes géométriques complexes sans entrer dans les détails réservés à la lasergrammétrie ainsi que l'apport de textures.

Figure 1.9. Application des différentes méthodes de levé en fonction du niveau de détail

Niveau de détail faible	Géocodification	Photogrammétrie aérienne	Photogrammétrie + géocodification
Niveau de détail moyen	Géocodification	Photogrammétrie terrestre	Photogrammétrie + géocodification
Niveau de détail élevé	Lasergrammétrie	Lasergrammétrie + photogrammétrie	Lasergrammétrie + photogrammétrie + géocodification

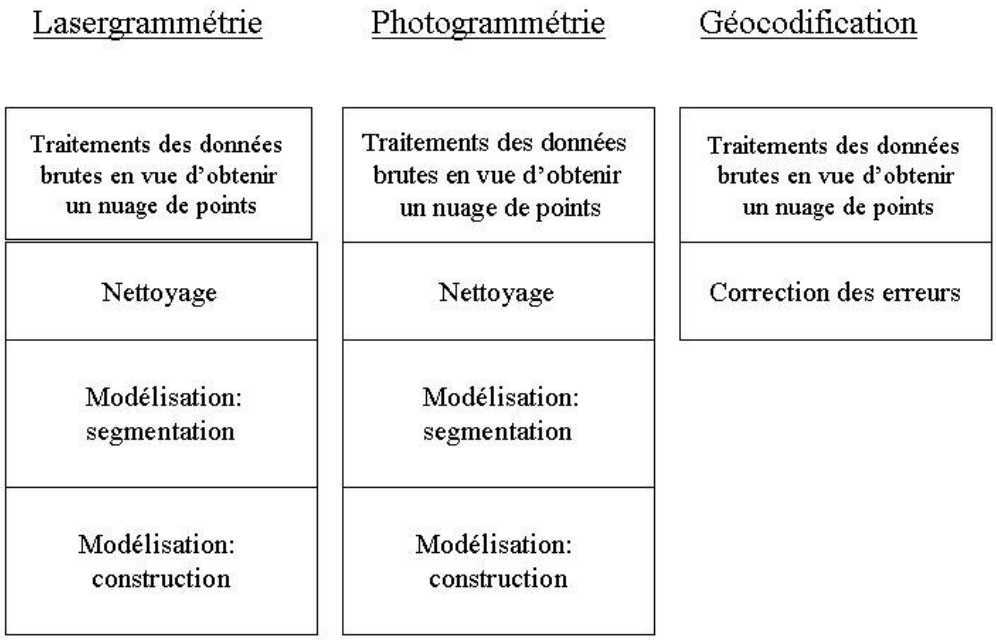
1.3.4.2. Allégement des fichiers de données et du post-traitement

Le gros inconvénient de la géocodification par rapport aux autres techniques est le temps passé sur le terrain. En effet, alors qu'un scanner lève plusieurs millions de points en quelques minutes et qu'un cliché pris en une fraction de seconde regroupe autant de pixels, le nombre de points levés au moyen d'une station totale en une journée de terrain semble dérisoire.

Cela dit, le nombre de points issu d'un levé avec station totale tend vers le minimum nécessaire pour la construction d'un modèle 3D, alors que le nombre de points issu des deux autres techniques est nettement supérieur à ce qu'il faut pour construire un modèle. Cela implique des fichiers « input » pour les algorithmes nettement moins lourds et donc des temps de traitements par ordinateur moins longs.

Nous avons vu qu'aussi bien pour la photogrammétrie que la lasergrammétrie, le post-traitement est assez conséquent : consolidation, nettoyage, modélisation, *etc*.… Concernant un levé avec station totale et géocodification, idéalement, le post-traitement se résume aux différents calculs permettant le passage des données d'angles et de distances vers des coordonnées (compensées) dans un système global (comme pour l'obtention du nuage de points en lasergrammétrie). La simple lecture du fichier de points, avec leurs coordonnées et codes, par un algorithme adéquat renvoie un modèle 3D déjà segmenté et construit. Il ne reste alors plus qu'à corriger les quelques erreurs de géocodification provoquées par l'utilisateur (en pratique, il y en a toujours) en modifiant manuellement les données géométriques du modèle 3D. Ainsi, si la technique de la géocodification est bien adaptée au travail effectué, notamment en ce qui concerne le niveau de détail, nous pouvons espérer un gain de temps considérable par rapport à la seule utilisation de la photogrammétrie et de la lasergrammétrie.

Figure 1.10. Comparaison des différentes étapes de post-traitement pour chaque méthode de levé dans le but d'obtenir un modèle 3D



1.4. Utilité d'une géocodification 3D dans le contexte d'un SIG 3D : la norme CityGML

1.4.1. GML

GML (Geography Markup Language) est une grammaire du langage de programmation de base de données XML (Extensible Markup Language) définie par l'OGC (Open Geospatial Consertium). Elle permet d'exprimer les caractéristiques attributaires et géométriques des entités géographiques dans un format commun à tous dans une base de données géographique (SIG). Ainsi, des échanges d'informations peuvent avoir lieu sur internet.

1.4.2. CityGML

1.4.2.1. Généralités

Alors que nous connaissons GML essentiellement pour l'information géographique en deux dimensions, CityGML traite des modèles urbains en trois dimensions. Cette grammaire XML reposant sur le même principe que GML (information géographique normalisée et échangeable via internet) est développée depuis l'année 2002 par les membres du Special Interest Group 3D (SIG 3D) en Allemagne.

Avant cela, d'autres modèles 3D avaient été développés mais ne tenaient compte que de la représentation géométrique des objets. L'avantage de CityGML par rapport à ses prédécesseurs est donc son aspect « SIG », en définissant les classes et les relations du point de vue de la topologie et de la sémantique. Cet aspect supplémentaire permet des analyses sophistiquées de type simulation, fouille de données, *etc.*… A l'avenir, CityGML devrait être approuvé par l'OGC comme schéma d'application des modèles 3D urbains.

Les applications visées par CityGML sont, par exemple, l'aménagement du territoire, le design architectural, le tourisme et le loisir, les simulations environnementales, la télécommunication, la gestion des désastres, la navigation (véhiculée ou pédestre), *etc.*…

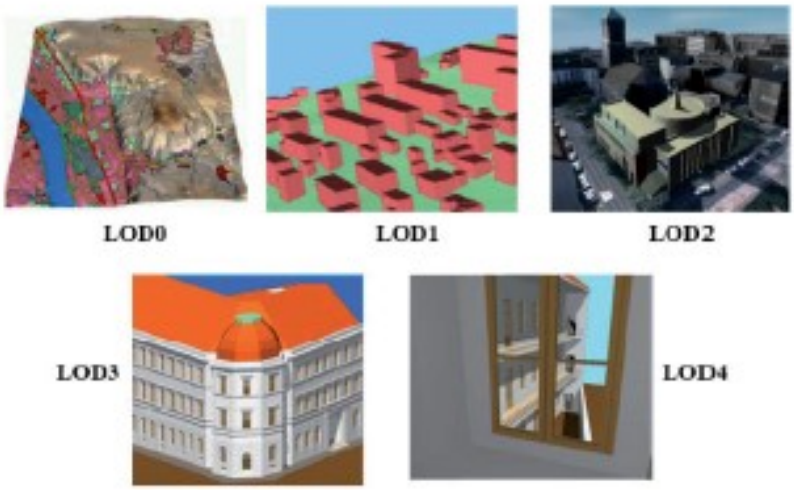
1.4.2.2. Les niveaux de détail

Les objets que la norme CityGML reconnaît sont d'abord les modèles numériques de terrains (MNT) sur lesquels sont posés les autres objets urbains, à savoir les bâtiments, les ponts, les tunnels, les murs de soutènement, les rivières, *etc.*… A ces objets s'ajoutent les routes, les chemins de fer, les voies navigables, le mobilier urbain, les feux tricolores, les réseaux de transport, les arbres, *etc.*…

La présence – ou l'absence – de ces objets dans un modèle est liée au niveau de détail défini par CityGML (Level Of Detail). Nous pouvons compter pour le moment cinq niveaux de détails :

- LOD 0 (Modèle régional) : modèle numérique de terrain (2,5D) permettant de montrer l'ensemble d'un paysage ;
- LOD 1 (Modèle urbain) : "modèle bloc" dans lequel les bâtiments sont schématisés sous forme de blocs sans structure du toit, donnant ainsi une idée de la répartition de la hauteur des bâtiments (niveau faible évoqué au point 1.3.4.1) ;
- LOD 2 (Modèle urbain) : même modèle que précédemment mais avec des textures pour les façades et les toitures (niveau moyen évoqué au point 1.3.4.1) ;
- LOD 3 (Modèle urbain) : même modèle que précédemment mais avec un niveau plus détaillé du point de vue architectural (niveau élevé évoqué au point 1.3.4.1) ;
- LOD 4 (Modèle intérieur) : modèle véritablement architectural "parcourable", c'est-à-dire avec un modèle de l'intérieur des bâtiments (niveau très élevé évoqué au point 1.3.4.1).

Figure 1.11. Niveaux de détails selon la norme CityGML



(Source : LAURINI & SERVIGNE 2008)

L'intérêt de ces niveaux de détail est double. D'une part, le travail effectué sur terrain peut s'adapter à l'application du modèle. D'autre part, puisque plusieurs niveaux peuvent – mais ne doivent pas – coexister pour un même modèle, le transfert de données ou le temps d'affichage peut être réduit en passant à un niveau inférieur. La définition des niveaux de détail n'est pas encore très précise et devrait être améliorée et complétée à l'avenir.

1.4.3. Utilité d'une géocodification 3D pour CityGML

Non seulement une géocodification 3D trouve sa place dans le contexte des niveaux de détail de la norme CityGML (voir point 1.3.4.1), mais nous pourrions également imaginer une implémentation automatique d'un SIG-3D au moyen des codes introduits de la même manière qu'une géocodification 2D permet le classement automatique des objets levés dans les différentes couches d'un plan. Cette perspective sera un peu plus développée dans le chapitre 4 de ce mémoire (*cf. infra*).

1.5. Objectifs du mémoire

Malgré l'engouement général ces dernières années pour les modèles 3D, le domaine de la géocodification 3D ne semble pas avoir été réellement développé. Pourtant, nous venons de voir que bien que la photogrammétrie et la lasergrammétrie soient des techniques relativement rentables, l'apport complémentaire d'une géocodification 3D efficace serait justifié par le gain de temps énorme au niveau du post-traitement.

L'objectif de ce mémoire est donc de développer et de tester sur terrain un prototype de géocodification 3D qui pourra éventuellement servir de base à de futurs travaux dans le domaine, si les résultats s'avèrent concluant. Nous nous souviendrons qu'une géocodification efficace ne doit pas être trop compliquée, malgré le fait que la troisième dimension élargit considérablement l'éventail de types de forme géométrique à lever par rapport aux autres géocodifications développées jusque-là. C'est pourquoi, même si théoriquement les lois de la géométrie permettent la conception d'une géocodification 3D, ce mémoire a pour principale

ambition d'apporter une réponse à la question : le développement d'une géocodification 3D à la fois simple et efficace est-il envisageable d'un point de vue pratique ?

Notre domaine d'application se limitera uniquement au bâti pour deux raisons. D'une part, ce cela rentre bien dans le cadre de la norme CityGML. D'autre part, les objets naturels ne présentent que très peu de régularités géométriques exploitables par une géocodification.

Afin de ne pas nous égarer dans des détails succins, nous nous focaliserons sur le problème amené par la troisième dimension : l'aspect géométrique. En effet, que nous soyons en 2D ou en 3D, l'aspect attributaire des objets (en analogie à leur répartition dans les différents calques d'un plan) n'est pas bien différent. De plus, la codification développée ciblera uniquement les objets à faces planes, la gestion des objets à faces non-planes s'avérant assez complexe du point de vue de la programmation – mais pas impossible. Cela dit, n'oublions pas que la géocodification 3D doit être vue comme une méthode complémentaire et non concurrente de la photogrammétrie et de la lasergrammétrie auxquelles le domaine des objets complexes peut être octroyé. L'aspect attributaire et la problématique des faces non-planes, bien que non programmés, seront malgré tout discutés à la fin de ce mémoire.

2. Méthodologie

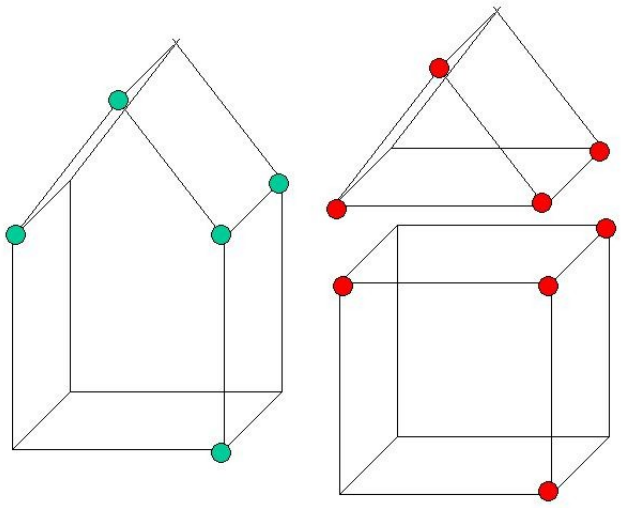
2.1. Philosophie générale et justification de la méthode développée

Les entités à moins de deux dimensions (les points et les lignes) ne sont pas codées de manière différente en 3D par rapport à la 2D. Notre méthodologie portera donc sur les entités zonales et surtout volumiques.

Plusieurs approches peuvent être envisagées dans la codification des objets tridimensionnels. A première vue, la plus évidente semble être la décomposition des objets sur le terrain en primitives géométriques (cubes, parallélépipèdes rectangles, prismes, *etc.*…) levés avec le nombre de points minimum pour les construire.

Prenons l'exemple d'une maison décomposable en deux primitives : un parallélépipède rectangle et un prisme. Un minimum de quatre points permet de construire un parallélépipède rectangle orienté (trois sommets d'une face et un point indiquant l'élévation), tandis qu'un minimum de quatre points permet la construction d'un prisme orienté (trois sommets d'une face triangulaire et un quatrième point indiquant l'élévation). Comme nous pouvons le voir sur la figure 2.1. (*cf. infra*), en considérant qu'un même point peut admettre plusieurs codes, un minimum de cinq points permet le levé de la maison.

Figure 2.1. Illustration de la géocodification par primitives géométriques



Cette méthode a donc l'avantage d'être extrêmement efficace du point de vue du nombre de points à mesurer. Cependant elle présente plusieurs inconvénients majeurs.

Tout d'abord, si nous développons une codification applicable à tous les objets que nous pouvons rencontrer sur le terrain, le nombre de primitives géométriques devient assez conséquent. Et puisqu'à chaque primitive, il faut associer un code particulier ainsi qu'une ou plusieurs méthodes de levé, l'information à mémoriser par l'utilisateur pour appliquer correctement la codification est beaucoup trop abondante. Ensuite, puisque les objets sont

construits à partir d'un minimum de points, les erreurs de mesure se répercutent de façon importante sur le tracé de la forme. Enfin, puisque des points bien précis doivent être levés pour reconstruire une primitive, si un ou plusieurs points s'avèrent inaccessibles, la codification ne peut tout simplement pas être appliquée.

Ainsi, nous abandonnons cette idée de primitives géométriques pour nous tourner vers une codification par construction de plans : nous levons indépendamment chaque face des objets en mesurant un minimum de trois points (minimum nécessaire pour retrouver l'équation d'un plan). En indiquant dans un code la connectivité des faces de chaque objet, un algorithme peut reconstruire toutes les arêtes de l'objet en question en utilisant les intersections des plans connectés. Nous sommes donc théoriquement capables de lever la totalité des objets à faces planes avec une seule codification et une seule méthode de levé. Notons que deux faces sont connectées lorsqu'elles ont une arête commune.

Cependant, plus le nombre de faces d'un objet devient important et plus l'application de cette méthode s'avère complexe aussi bien du point de vue du nombre de points à mesurer que de celui du nombre de codes à appliquer et à mémoriser (*cf. infra*, 2.3.). La codification par construction de faces peut donc nous servir de base mais nécessite une série de codifications complémentaires afin d'optimiser la méthode. Par « codification complémentaire », nous entendons une codification simple s'appuyant ou non sur la codification de base (par construction de plans) dont le domaine d'application est limité (applicable sur certains types d'objets bien particuliers). Les codifications complémentaires développées sont les suivantes :

- Liaison par face
- Liaison par parallélépipède
- Contour de face
- Proéminence et enfoncement
- Répétition d'objet

Dans ce chapitre, pour chaque méthode (basique ou complémentaire), nous dresserons une description générale avec domaine d'application, explication des codes et un ou plusieurs exemples. Enfin, nous expliquerons brièvement le fonctionnement de l'algorithme associé à ces codes.

2.2. Les codes communs

Les différentes codifications développées dans ce mémoire sont implémentées dans un seul programme. Elles suivent donc certaines normes afin d'être cohérentes entre elles.

2.2.1. Séparateur de code

Il est possible d'associer plusieurs codes à une mesure en utilisant le séparateur de code « / ». Par exemple, si à une mesure, on veut associer les codes X, Y et Z, on écrira :

X/Y/Z

2.2.2. Séparateur de paramètre

Un code est presque toujours suivi d'une série de paramètres alphanumériques. Le séparateur de paramètre est le « . ». Par exemple, si au code X, nous associons les paramètres a, b et c, on écrira :

X.a.b.c

2.2.3. Identification d'un objet

La codification développée dans ce mémoire nécessite l'identification des différents objets levés sur le terrain. Cela se traduit par le code « OBJ » suivi du nom de l'objet levé. Par exemple, si on lève l'objet « maison1 », on indiquera le code :

OBJ.maison1

Toutes les mesures et codes qui suivent cette instruction seront considérées comme faisant partie de l'objet en question. Le code « OBJ » a plusieurs raisons d'être :

- certains codes se basent sur des objets déjà construits et nécessitent l'introduction du nom de ces derniers lors de l'encodage ;
- chaque face levée est identifiée (par l'utilisateur ou par l'algorithme). L'association du nom de la face et de l'objet auquel elle appartient permet de lui donner un identifiant unique. Notons donc que les noms d'objets doivent également être uniques ;
- si nous souhaitons interrompre le levé d'un objet x pour entamer le levé d'un objet y, nous pourrons reprendre nos mesures là où elles s'étaient arrêtées pour l'objet x en réinscrivant le code «OBJ.x ».

Notons que les noms d'objets et de faces peuvent également servir à l'implémentation semi-automatique d'un SIG (*cf. supra*).

2.3. Méthode de base : codification par construction de plans

2.3.1. Généralités

Cette méthode permet de lever tout type d'objet tridimensionnel à face plane, qu'il s'agisse d'une forme bien connue (cube, parallélépipède, …) ou tout à fait quelconque. Comme décrit plus haut, le principe est de lever au moins trois points par face afin de déterminer le plan la décrivant. Il est évident que les arêtes et les sommets contiennent des points à privilégier puisqu'ils appartiennent respectivement à deux et trois faces au minimum.
Il suffit donc tout simplement, après le levé de chaque point, d'indiquer le nom des différentes faces auxquelles il appartient. Après avoir levé au minimum trois points par face, nous connaissons l'équation des plans formant chaque face de l'objet levé. Les arêtes sont formées par l'intersection de deux plans et les sommets par l'intersection de trois plans (ou deux arêtes).

Hélas, de cette manière, nous ne disposons pas d'assez d'information pour isoler les bonnes arêtes et les bonnes sections de plans. C'est pourquoi un nouveau code doit se greffer au premier : la connectivité entre faces. L'information « connectivité » permet de savoir quelles faces comportent des arêtes communes. En connaissant cette connectivité, nous savons quels plans sont supposés se recouper pour former une arête, et sommes donc en mesure dessiner l'objet complet. La connectivité peut être codée explicitement sur le terrain (méthode fastidieuse) ou implicitement (méthode rapide au cas par cas).

2.3.2 Codification

2.3.1.1. Codes associé à une mesure

Après chaque point levé, une ligne de code indique les faces contenant ce dernier. Elle commence par le code « F » qui indique la codification par construction de faces et est suivi des noms des différentes faces séparées par un « . ».

Exemple : F.F1.F2.F3

Note : chaque face comporte un identifiant unique par objet.

2.3.1.2. Connectivité (explicite)

A n'importe quel moment du levé, une ligne de code indiquant la connectivité peut être insérée. Celle-ci se rapporte à l'objet auquel l'identifiant a été codé pour la dernière fois et commence par le code « C ». Deux méthodes pour indiquer la connectivité peuvent être utilisées. Dans les deux cas, la lettre « C » est suivie d'une série de faces séparées par un « . ».

- Méthode « face par face » : la première face encodée est celle pour laquelle nous décrivons la connectivité. Celle-ci est suivie de la liste des faces ayant une arête commune avec elle (« faces adjacentes »).
- Méthode « arête par arête » : pour chaque arête, les deux faces qui la comprennent sont encodées.

La méthode « face par face » présente certaines redondances par rapport à la méthode « arête par arête » : des arêtes sont indiquées plusieurs fois. Cela dit, la méthode « face par face » laisse donc moins de chance à l'utilisateur d'oublier de coder une ou plusieurs arêtes.

Figure 2.2. Exemple : codification par construction de plans

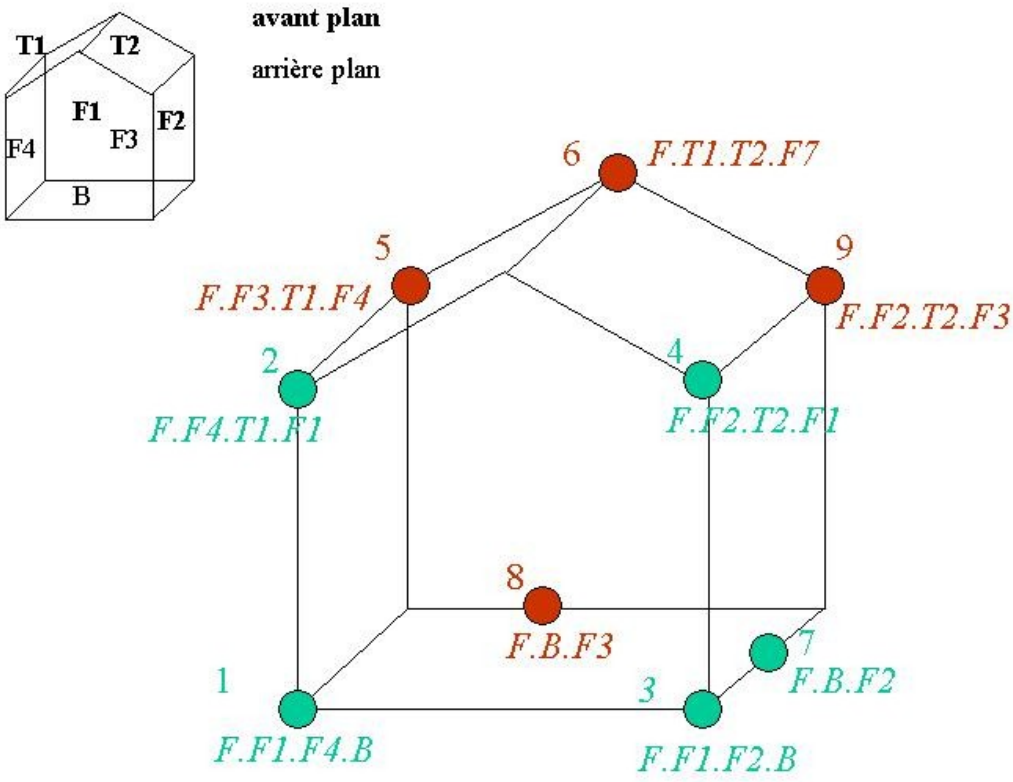


Tableau 2.1. Exemple : mesures et connectivité (encodage « face par face »)

Mesure	Code
/	OBJ.MAISON
1	F.F1.F4.B
2	F.F4.T1.F1
3	F.F1.F2.B
4	F2.T2.F1
5	F.F3.T1.F4
6	F.T1.T2.F7
7	F.B.F2
8	F.B.F3
9	F.F2.T2.F3
/	C.F1.B.F2.F3.F4.T1.T2
	C.F2.F1.F3.B.T2
	C.F3.F2.F4.B.T1.T2
	C.F4.F1.F3.B.T1
	C.B.F1.F2.F3.F4
	C.T1.T2.F1.F4.F3
	C.T2.F1.F2.F3.T1

Tableau 2.2. Exemple : connectivité (encodage « arête par arête »)

Mesure	Code
/	C.B.F1 C.B.F4 C.B.F2 C.B.F3 C.F1.F2 C.F1.F4 C.F2.F3 C.F1.T2 C.F1.T1 C.F2.T2 C.F4.T1 C.F3.T2 C.F3.T1 C.T1.T2 C.F4.F3

2.3.3. Connectivité implicite

Nous pouvons remarquer qu'aussi bien dans le cas de la connectivité « face par face » que « arête par arête », le code à inscrire est extrêmement long et donc source d'erreur. Pour remédier à ce problème, nous pouvons avoir recours à une connectivité implicite : il s'agit d'une seule ligne de code permettant de décrire la connectivité d'un objet ou un type d'objet particulier.

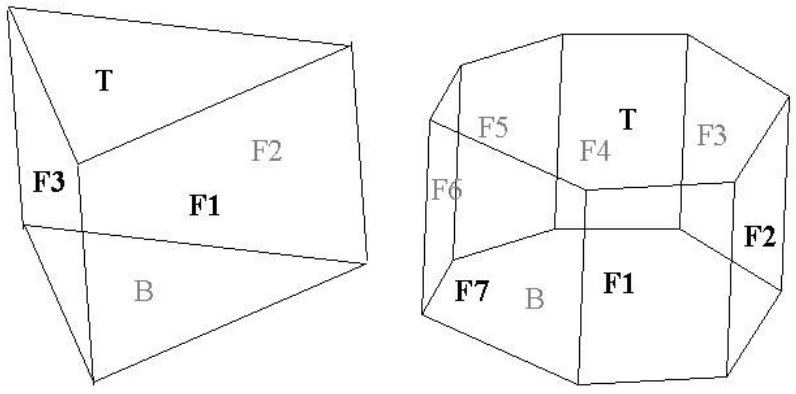
Imaginons que dans un levé, nous soyons amenés à rencontrer assez souvent des objets de type « maison » comme dans la figure 1.2. Il nous suffit alors de préparer un code « C.maison » dans notre codification qui indique à l'algorithme que la connectivité est celle du tableau 1.1 ou 1.2. La connectivité implicite est donc un code qui doit être facilement programmable par l'utilisateur même si la codification peut comporter une bibliothèque de connectivités implicites de base.

Une connectivité implicite peut également s'adapter à une série d'objets possédant des caractéristiques similaires au niveau de la connectivité. Prenons l'exemple d'objets ayant une face servant de base et une face servant de toit toutes les deux reliées par un certain nombre de faces latérales. Peu importe le nombre de faces latérales, comme le montre la figure 1.3, leur connectivité peut être décrite par un seul et même code :

BT.n

Où n est le nombre de faces latérales.

Figure 2.3. Exemples : connectivité implicite « BT »



Connectivité:

BT.3

Connectivité:

BT.7

Remarquons également que la connectivité implicite ne permet plus de nommer les faces d'objets comme on le souhaite : il faut respecter le code de connectivité sous-entendu. Ainsi, dans l'exemple de la connectivité « BT », la base et le toit de l'objet doivent s'appeler « B » et « T » (ou inversement) et les faces latérales doivent s'appeler « Fx » où x est un nombre naturel allant de 1 jusque n (n = nombre de faces latérales). Les faces doivent donc être nommées de 1 jusque n en tournant autour de l'objet dans un sens constant (trigonométrique ou horloger). En pratique, l'utilisateur doit juste mémoriser l'emplacement de la face « F1 » et le sens de rotation qu'il a choisi, les autres faces pouvant être retrouvées par déduction.

Tableau 2.3. Traduction de la connectivité implicite « BT » en « face par face »

C.B.F1.F2 … Fn
C.T.F1.F2 … Fn
C.F1.Fn.F2.B.T
…
C.Fx.F(x-1).F(x+1).B.T
…
C.Fn.F1.F(n-1).B.T

2.3.4. Intérêt particulier de la méthode

Non seulement la géocodification par plan permet le levé de la totalité des objets à faces planes, mais elle permet également de lever des objets pour lesquels une face n'est pas visible depuis la station. En effet, puisque les faces sont construites à partir d'équations de plan, les points décrivant ces faces ne doivent pas nécessairement être saisis sur les faces elles-mêmes mais sur le plan la décrivant. Par exemple, le bas d'une cheminée peut être levé au moyen de points du toit sur lequel elle repose.

2.4. Méthodes complémentaires

Bien qu'elle permette le levé de tous les objets à faces planes, la codification par construction de plans ne peut pas se suffire à elle-même en raison de sa complexité croissante avec la complexité de l'objet (liée au nombre et à la disposition des faces). C'est pourquoi d'autres méthodes dites complémentaires vont être développées. Ces méthodes ont toutes un domaine d'application plus restreint mais restent assez simple, peu importe le nombre ou la disposition des faces des objets levés.

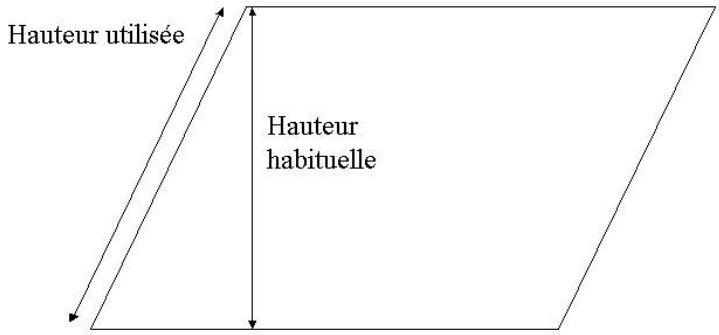
2.4.1. Liaison par face

2.4.1.1. Généralités

Cette méthode est une extrapolation 3D de la géocodification COVADIS (2D), le concept de base étant simplement de suivre le contour au sol des objets en levant les points de changement de direction du contour (liaison). Les liaisons en 2D sont donc effectuées au moyen d'une succession de lignes. En 3D, nous ne travaillons pas avec des lignes mais avec des faces en forme de parallélogramme. Un paramètre de hauteur est donc nécessaire par

rapport à la liaison par ligne. Précisions via la figure 1.4 que le terme « hauteur » d'un parallélogramme n'a pas ici la même valeur que celle à laquelle on est habitué (longueur du segment perpendiculaire à deux bases du parallélogramme).

Figure 2.4. Hauteur utilisée d'un parallélogramme



Une liaison par face est représentée dans l'espace par un ensemble de morceaux de plans (faces) délimités par des arêtes (intersections de plan).

Les points à mesurer sont les sommets des arêtes inférieures et/ou les sommets des arêtes supérieures. A chaque mesure de liaison, un code indique si nous ouvrons, poursuivons ou fermons une liaison. Un paramètre de hauteur peut être introduit dans le code des points. Une autre approche est de mesurer des points des arêtes supérieurs (resp. inférieurs) afin de coder implicitement le paramètre « hauteur ».

Cette codification permet de lever des faces rectangulaires d'objets impossibles à lever dans leur totalité (exemple : une dalle). Il permet également de lever des objets complets pour autant que leurs faces latérales soient des parallélogrammes et qu'ils soient composés d'une seule face à leur base supérieure et inférieure.

2.4.1.2. Codification

Le code associé à chaque point pour introduire une liaison par face est « lip ». Le paramètre (numérique) suivant ce code indique le rôle du point au niveau du tracé de la liaison. Ils sont au nombre de cinq:

- lip.0.x : ouverture (premier point de la liaison). x est un second paramètre permettant de coder directement la hauteur de la face. Dans ce dernier cas, la face aura toujours une hauteur orientée comme l'axe z du système de coordonnées.
- lip.1 : liaison (point suivant de la liaison).
- lip.2 : point donnant la hauteur. Notons que la hauteur est alors calculée à partir du point d'ouverture.

- lip.3 : clore la liaison sur le point d'ouverture. Dans ce cas, l'algorithme trace automatiquement les faces supérieures et inférieures de la liaison.
- lip.4 : terminer la liaison sans clore sur le premier point.

Notons que pour que cette codification soit compatible avec la méthode complémentaire « enfoncement et proéminence » (que nous verrons un plus loin dans ce chapitre), toutes les faces doivent être nommées. L'identification des faces se fait automatiquement par l'algorithme selon la configuration suivante. La première face de la liaison s'appelle F1, la seconde F2, *etc.*… La face inférieure (dont les sommets sont les points levés pour effectuer la liaison) est nommée « B » et la face supérieure (donnée par le paramètre de hauteur ou le point donnant la hauteur) est nommée « T ».

Notons également qu'une nouvelle liaison doit toujours être précédée d'un nouveau code « OBJ » afin que les faces puissent être identifiées de façon unique.

Figure 2.5. Exemple 1 : codification par liaison par plan

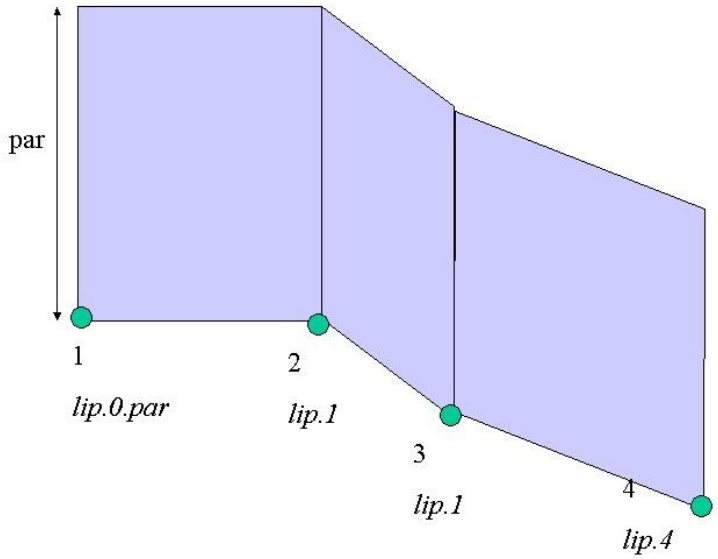


Tableau 2.4. Exemple 1 : mesures et codes

Mesure	Code
/	OBJ.FACADE
1	lip.0.par
2	lip.1
3	lip.1
4	lip.4

Figure 2.6. Exemple 2 : codification par liaison par plan

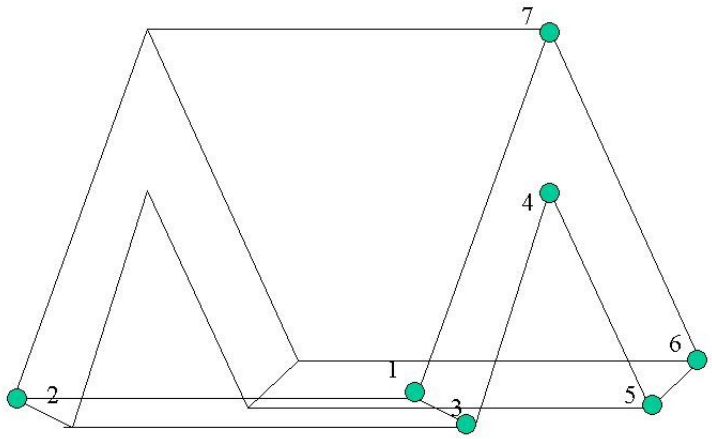


Tableau 2.5. Exemple 2 : mesures et codes

Mesure	Code
/	OBJ.TOIT
1	lip.0
2	lip.2
3	lip.1
4	lip.1
5	lip.1
6	lip.1
7	lip.3

2.4.2. Liaison par parallélépipède

2.4.2.1. Généralités

Nous venons de voir qu'il était possible d'établir des liaisons au moyen de parallélogrammes. Nous pouvons également imaginer une méthode permettant d'établir des liaisons au moyen de parallélépipèdes. Une liaison par parallélépipède est représentée dans l'espace par un ensemble de parallélépipèdes liés deux à deux par une face implicite (non représentée).

La méthode de levé est identique à celle de la liaison par face à l'exception du paramètre « épaisseur » qui peut être codé explicitement ou implicitement au moyen d'un point supplémentaire. En effet, un parallélogramme est un parallélépipède d'épaisseur nulle.

Remarquons que l'épaisseur utilisée est la même que la hauteur du parallélogramme évoquée précédemment (en suivant l'arête).

Cette codification permet donc de lever des objets pouvant être assimilés à un ensemble de parallélépipèdes liés deux à deux (exemple : mur).

2.4.2.2. Codification

Le code associé à chaque point est « lipr » et est suivi d'un paramètre numérique indiquant le rôle du point dans le tracé de la liaison. Ces paramètres sont identiques à la liaison par plan à l'exception de l'introduction du paramètre « épaisseur ». Voici la liste des paramètres :

- lipr.0.x.y : ouverture (premier point de la liaison). x est un second paramètre permettant de coder directement la hauteur du parallélépipède et y est un troisième paramètre permettant de coder directement l'épaisseur du parallélépipède. Dans ce dernier cas, le parallélépipède aura toujours une hauteur orientée comme l'axe z du système de coordonnées et une épaisseur orientée perpendiculairement à l'axe z.
- lipr.1 : liaison (point suivant de la liaison).
- lipr.2 : point donnant la hauteur. Notons que la hauteur est alors calculée à partir du point d'ouverture.
- lipr.3 : clore la liaison sur le point d'ouverture. Dans ce cas, l'algorithme trace automatiquement les faces supérieures et inférieures de la liaison.
- lip.4 : terminer la liaison sans clore sur le premier point.
- lipr.5 : point donnant l'épaisseur

La logique d'identification des faces est la même que pour la liaison par face. Cependant, à chaque nouveau parallélépipède, nous n'avons plus une mais quatre ou cinq nouvelles faces introduites. Ainsi, à chaque nom donné à un parallélépipède, l'algorithme nomme implicitement toutes les faces par :

- xG : face de gauche
- xD : face de droite
- xS : face supérieure
- xI : face inférieure
- xB : face d'ouverture de la liaison
- xE : face de fermeture de la liaison

x correspond à l'identifiant du parallélépipède. Le parallélépipède est identifié comme les faces de la liaison par face : le premier est nommé P1, le second P2, *etc.*…
La gauche et la droite sont déterminées en regardant vers le sens de levé de la liaison.

Notons également qu'une nouvelle liaison doit toujours être précédée d'un nouveau code « OBJ » afin que les faces puissent être identifiées de façon unique.

Figure 2.7. Exemple 1 : liaison par parallélépipède

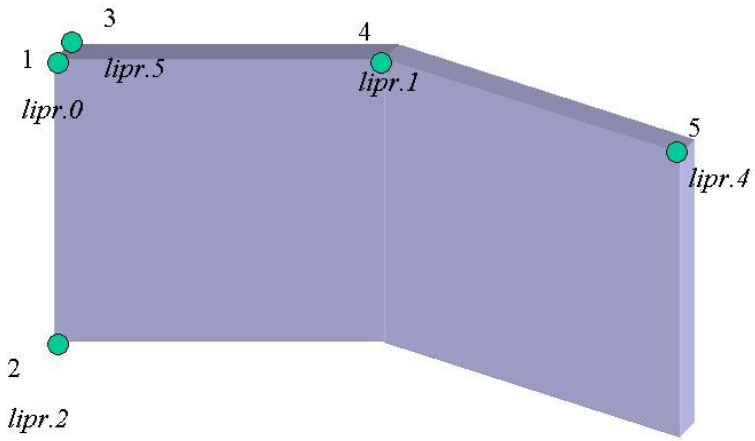


Tableau 2.6. Exemple 1 : mesures et codes

Mesure	Code
/	OBJ.MUR
1	lipr.0
2	lipr.2
3	lipr.5
4	lipr.1
5	lipr.4

Figure 2.8. Exemple 2: liaison par parallélépipède

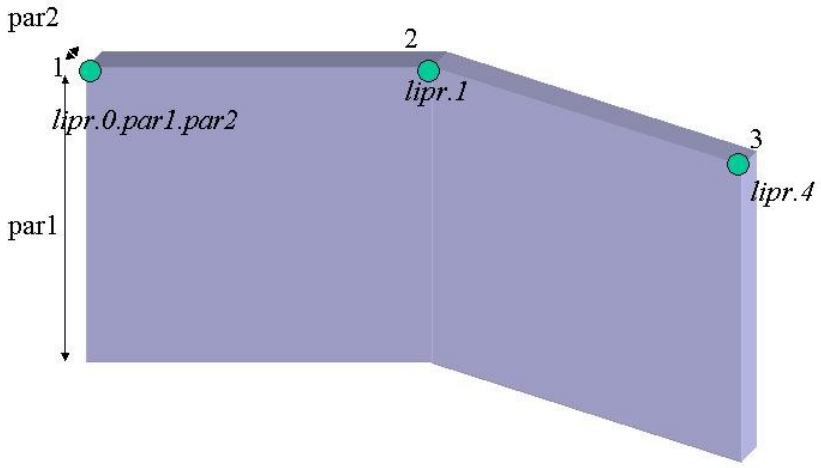


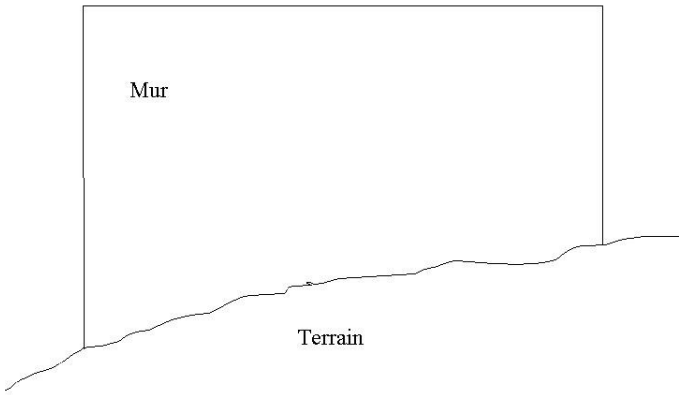
Tableau 2.7. Exemple 2 : Mesures et codes

Mesure	Code

/	OBJ.MUR
1	lipr.0.par1.par2
2	lipr.1
3	lipr.4

Remarquons qu'aussi bien dans le cas de la liaison par plan que par parallélépipède, nous nous sommes intéressés à des faces latérales avec des arêtes parallèles entre elles. Si nous prenons l'exemple du levé d'un mur (figure 2.9), nous nous rendons compte que ses faces inférieures ne sont pas toujours parallèles aux faces supérieures à cause du dénivelé du terrain.

Figure 2.9. Illustration du problème d'une liaison pour un objet rattaché au sol



Deux solutions peuvent alors être envisagées :

- La liaison du mur est effectuée normalement à partir des points supérieurs et le bas du mur sera obtenu postérieurement par intersection avec un modèle numérique de terrain (MNT)
- Nous envisageons une codification particulière où la liaison est effectuée par le bas du mur et où les points supérieurs appartiennent toujours à un même plan d'équation z = h (où h correspond à la coordonnée z du point donnant la hauteur).

2.4.3. Contour de face

2.4.3.1. Généralités

Le principe de cette méthode est simple : on mesure chaque sommet d'une face en longeant ses arêtes dans le même sens (trigonométrique ou horloger). Il est clair que mesurer un objet complet avec cette méthode serait extrêmement fastidieux (cela impliquerait d'ailleurs plusieurs mesures d'un même point). L'intérêt de cette méthode réside dans le fait de pouvoir mesurer des parties d'objet uniquement, en imaginant que le reste de l'objet soit inaccessible ou tout simplement parce que seul cette partie est utile pour le projet pour lequel le levé est effectué. Tout type de face plane peut être levé de cette façon.

2.4.3.2. Codification

A chaque mesure effectuée, le code à appliquer est « CT » suivi d'un paramètre : le nom de la face. Par exemple :

CT.F1

Une fois la face levée, elle est considérée comme faisant partie de l'objet pour lequel le code « OBJ » a été inscrit pour la dernière fois.

Figure 2.10. Exemple : contour de face

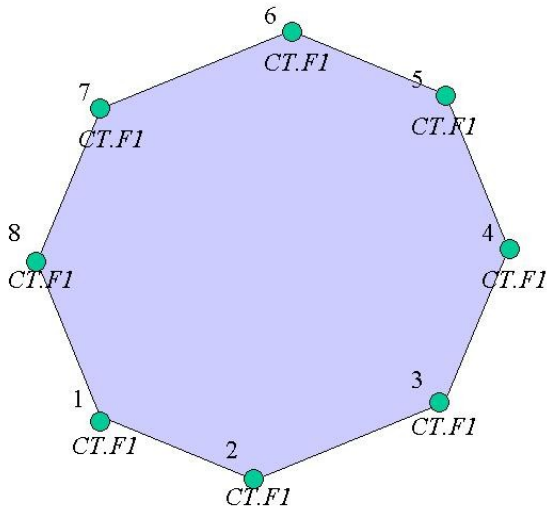


Tableau 2.8. Exemple : mesures et codes

Mesure	Code
/	OBJ.OCTOGONE
1	CT.F1
2	CT.F1
3	CT.F1
4	CT.F1
5	CT.F1
6	CT.F1
7	CT.F1
8	CT.F1

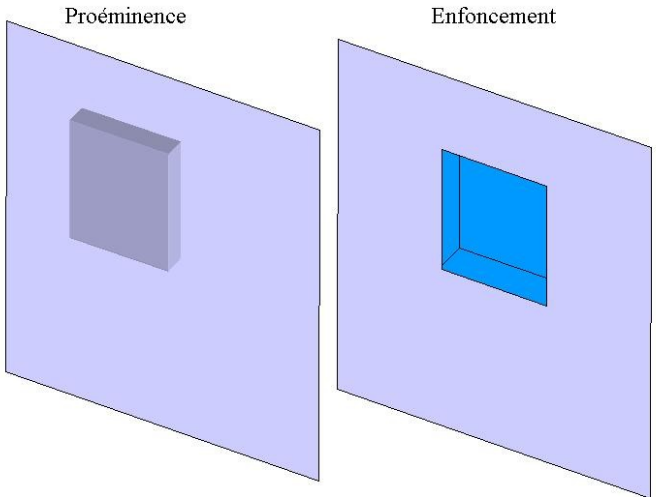
2.4.4. Proéminence et enfoncement

2.4.4.1. Généralités

Une proéminence ou un enfoncement d'un objet O1 désigne une face A de x arêtes (x > 2) pouvant être jointe par x faces (une face par arête) perpendiculairement à une face S de O1

que nous appellerons « support ». La proéminence se trouve à l'extérieur de l'objet O1 et l'enfoncement à l'intérieur.

Figure 2.11. Différence entre proéminence et enfoncement



2.4.4.2. Codification d'une proéminence

Une proéminence X sur un support F1 d'un objet O1 peut être géocodée de la manière suivante…

Lors du premier sommet levé de la face proéminente, nous entrons le code « PRO » suivi comme paramètres du nom de la proéminence (X) et du nom de la face support (F1) :

PRO.X.F1

Il s'agit en fait du point d'ouverture d'une liaison par plan, mis à part que le paramètre « hauteur » n'est plus nécessaire puisqu'il est donné par la distance entre la proéminence et la face support (F1).

Les autres points doivent être levés en parcourant la face proéminente le long des arêtes (que ce soit dans le sens horloger ou trigonométrique). Ceux-ci sont codés par le nom de la proéminence (X) et le paramètre « 1 »:

X.1

Le dernier point de la proéminence est codé par le nom de la proéminence suivi du paramètre « 3 » (clore sur le premier point):

X.3

La face proéminente étant ainsi déterminée, l'algorithme a toutes les informations nécessaires pour dessiner les autres faces. Il suffit en effet d'abaisser les arêtes de la face proéminente perpendiculairement au support.

Les faces d'une proéminence peuvent également servir de support à de nouvelles proéminences et enfoncements. C'est pourquoi les faces d'une proéminence X sont nommées implicitement par l'algorithme comme suit :

- face proéminence : XT
- première face de la liaison : XF1
- seconde face de la liaison : XF2
- …

Une proéminence d'un objet O doit toujours être levée après l'instruction « OBJ.O ». Dans le cas contraire, la face support ne serait pas reconnue ou serait confondue avec une face d'un autre objet (objet se rapportant à la dernière instruction « OBJ »).

Figure 2.12. Exemple 1 : proéminence

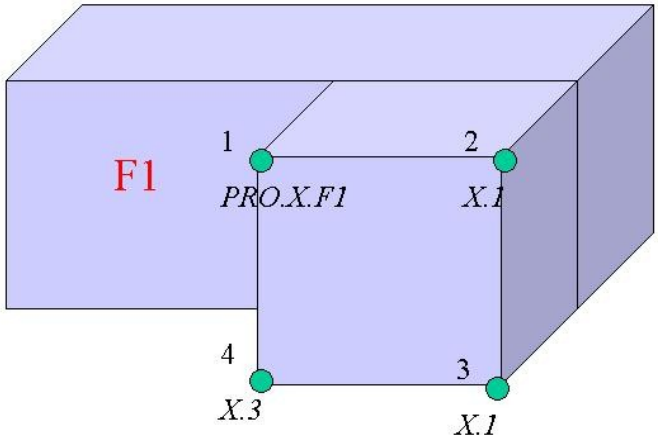


Tableau.2.9. Exemple 1 : mesures et codes

Mesure	Code
/	OBJ.O
1	PRO.X.F1
2	X.1
3	X.1
4	X.3

Remarquons qu'il serait intéressant de développer une codification tenant compte de proéminences avec des faces latérales non-perpendiculaires à la face support. Il suffirait alors

de coder un point supplémentaire donnant la direction d'une arête joignant le dessus de la face de la proéminence à la face support.

2.4.4.3. Codification d'un enfoncement

La méthode de l'enfoncement est similaire à celle de la proéminence, le code « PRO » étant remplacé par le code « ENF ». Cependant, il y a une forte probabilité pour que les points à mesurer pour établir la liaison au fond de l'enfoncement soient en partie invisibles depuis la station. C'est pourquoi il est nécessaire d'établir la liaison à l'aide des points à la surface de l'enfoncement et de mesurer un seul point au fond de l'enfoncement pour introduire la profondeur. La profondeur est déterminée par le calcul de la distance entre le point donnant la profondeur et le plan de la face support. Le point donnant la profondeur d'un enfoncement Y est codé de la manière suivante :

Y.PAR

Figure 2.13. Exemple 2 : enfoncement

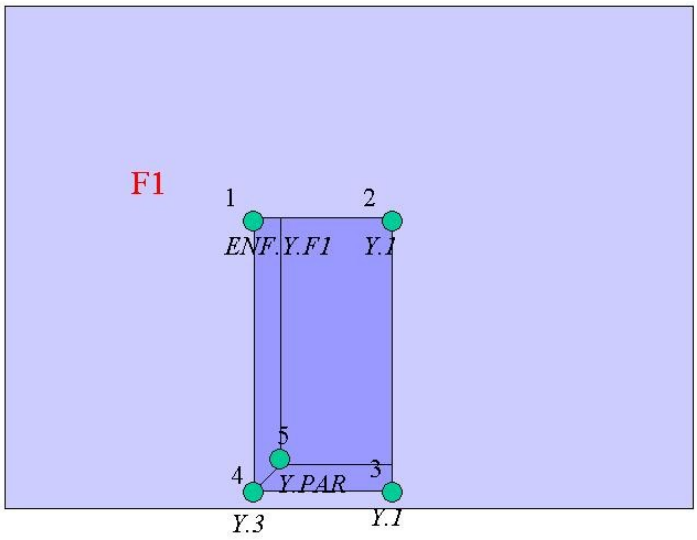


Tableau 2.10. Exemple 2 : mesures et codes

Mesure	Code
1	ENF.Y.F1
2	Y.1
3	Y.1
4	Y.3

2.4.5. Répétition d'objet

2.4.5.1. Généralités

En environnement bâti, nous pouvons retrouver plusieurs objets identiques du point de vue de leur géométrie et de leurs dimensions (exemple : piliers, marches d'escaliers, …) Cette caractéristique permet d'utiliser la méthode complémentaire de répétition d'objet. Celle-ci permet de dessiner un objet qui a déjà été levé à l'aide de trois points mesurés au maximum (peu importe la complexité de l'objet « modèle »). Cette méthode nécessite l'application de codes particuliers aussi bien sur l'objet modèle que l'objet copié.

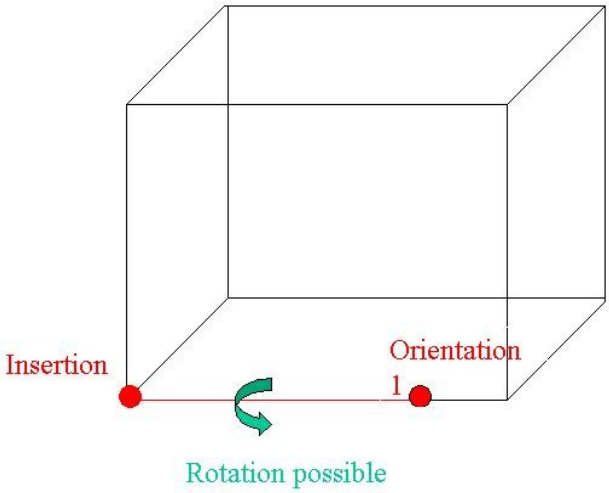
Evidemment, les points utilisés pour l'objet modèle peuvent également servir à sa propre codification. Il suffirait alors d'introduire un « / » séparateur de codes entre les deux codes pour les points concernés.

2.4.5.2. Objet modèle

Concernant l'objet modèle, il convient d'abord d'appliquer un code sur un point d'insertion. Si nous nous arrêtons à ce stade, ce point permettra de répéter l'objet modèle avec la même orientation dans l'espace (le point homologue devra être codé sur l'objet à copier).

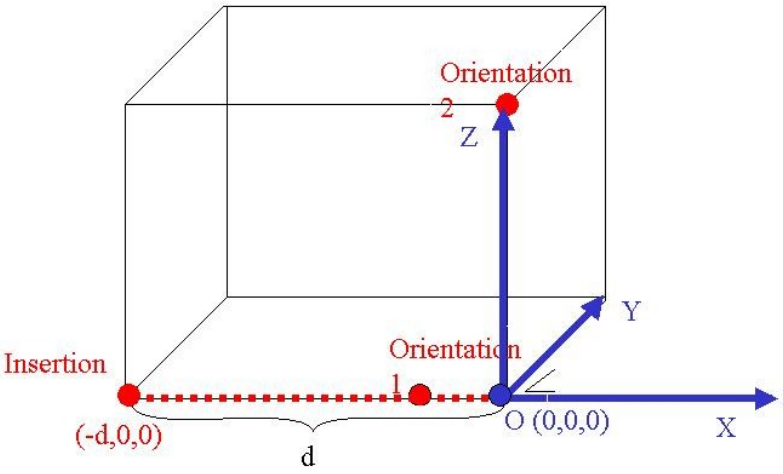
Si nous désirons copier l'objet modèle avec une orientation différente, deux autres points doivent être codés : les points d'orientation. Ces points peuvent être quelconques tant que leurs homologues peuvent être retrouvés facilement sur l'objet à copier, et que l'ensemble des trois points ne soit pas aligné. En effet, le premier point d'orientation sert à fixer l'orientation de l'objet autour de deux axes de l'espace. Le deuxième point permet de fixer l'orientation de l'objet autour du troisième axe de l'espace qui est formé par la droite joignant le point d'insertion et le premier point d'orientation. En d'autres termes, le point d'insertion et les points d'orientation permettent la construction d'un système de coordonnées tridimensionnel local propre à l'objet. Si le troisième point est aligné aux deux autres, le système ne peut être que bidimensionnel. Il s'agit évidemment d'un concept théorique, puisqu'en pratique, trois points ne peuvent quasiment jamais être alignés si nous travaillons avec une bonne précision. Cependant, trois points tendant vers l'alignement affectent négativement la précision de l'établissement du système de coordonnées locales.

Figure 2.14. Illustration de l'effet d'un point d'orientation sur les rotations possibles d'un objet copié



Comme le montre la figure 2.14, avec un point d'orientation, il n'y a plus qu'une seule rotation possible du dessin.

Figure 2.15. Illustration de l'effet de deux points d'orientation sur les rotations possibles d'un objet copié



Avec deux points d'orientation, il n'y a plus de rotation possible du dessin. La projection orthogonale du point d'orientation 2 sur la droite formée par le point d'insertion et le point d'orientation 1 nous donne l'origine du système de coordonnées locales. La droite formée par les deux premiers points est l'axe X (orienté du point d'insertion au point d'orientation 1), la droite de projection orthogonale du point d'orientation 2 est l'axe Z (orienté du point de la

projection du point d'orientation 2 au point d'orientation 2). L'axe Y est déduit des deux autres. Ainsi, le point d'insertion aura toujours des coordonnées de type (x,y,0).
Notons que dans le cas d'un objet modèle levé sans points d'orientation, le système de coordonnées local prend son origine au point d'insertion et est orienté comme le système de coordonnées global du levé.

2.4.5.3. Objet copié

Concernant l'objet à copier, le point d'insertion homologue est à coder ainsi que les éventuels points d'orientation homologue. Notons qu'il est possible de prendre n'importe quel point d'orientation homologue se trouvant dans la direction des axes fictifs du système de coordonnées local, pour autant que le système de coordonnées soit matérialisés sur l'objet (par exemple, par un coin à angles droits).
Enfin, notons la possibilité avec une seule ligne de code de copier un objet x fois. Après avoir copié un objet une fois, la différence de coordonnées entre les points d'insertion homologues ainsi que l'orientation de l'objet copié sont retenus par l'algorithme et peuvent être répétés pour les copies suivantes (exemple : escalier).

2.4.3.4. Codification de l'objet modèle

Les codes sur les points d'insertion, d'orientation 1 et d'orientation 2 du modèle sont respectivement :

INSM
OR1M
OR2M

2.4.3.5. Codification de l'objet copié

Avant de lever un objet copié, nous devons entrer l'instruction OBJ afin d'identifier l'objet que nous allons lever. Si l'objet copié est appelé « O2 », le code sera :

OBJ.O2

Soit un objet modèle « O1 », les codes sur les points d'insertion, d'orientation 1 et d'orientation 2 homologues sur un objet copié sont respectivement :

INS.O1
OR1.O1
OR2.O1

Si nous désirons effectuer plusieurs copies de l'objet à orientation constante et intervalles réguliers (comme pour un escalier par exemple), nous pouvons faire suivre le code d'insertion du paramètre « REP.x » où x est égal au nombre de copies de l'objet (incluant la première copie sur laquelle nous entrons le code) :

INS.O1.REP.x

Notons que dans les exemples des figures 1.15 et 1.16, l'objet modèle « O1 » est considéré comme déjà levé : les codes concernant son levé ne sont donc pas représentés.

Figure 2.16. Exemple 1 : répétition d'objets

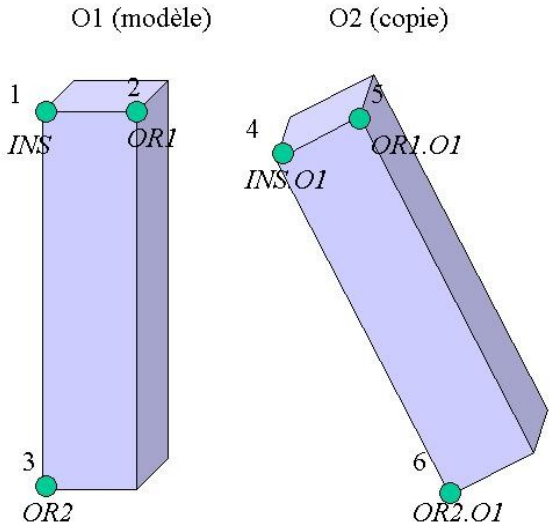


Tableau 2.11. Exemple 1 : mesures et codes

Mesure	Code
/	OBJ.O1
1	INS
2	OR1
3	OR2
/	OBJ.O2
4	INS.O1
5	OR1.O1
6	OR2.O1

Figure 2.17. Exemple2 : répétition d'objets

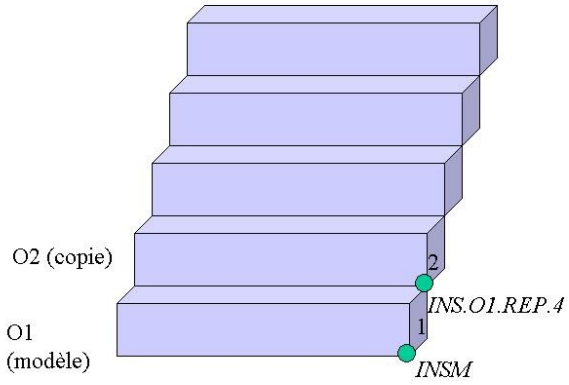


Tableau 2.12. Exemple 2: mesures et codes

Mesure	Code
/	OBJ.O1
1	INS
/	OBJ.O2
2	INS.O1.REP.4

2.5. Implémentation

2.5.1. Présentation et fonctionnement des outils de travail

2.5.1.1. Généralités

Les différentes méthodes de codification testées sont programmées au moyen de deux langages: Perl et VRML. Perl permet de traiter les informations provenant d'un fichier input et d'écrire un fichier de type VRML dans le langage du même nom (interface graphique).

2.5.1.2. VRML

Le langage de programmation VRML permet de dessiner des objets en trois dimensions. Il s'agit donc de l'interface graphique permettant de visualiser le résultat du levé d'objets topographiques au moyen d'une codification. Un fichier d'extension VRML peut être lu par

de nombreux navigateurs WEB pour peu qu'un plug-in VRML soit installé. L'interface VRML est assez rudimentaire : nous pouvons zoomer, effectuer des rotations et des translations sur le dessin général. Elle ne permet aucune retouche du dessin ni une distribution des différents éléments dans différentes couches, comme on pourrait le faire dans AUTOCAD. L'intérêt de ce langage réside principalement dans le réalisme des dessins et dans sa grande accessibilité, l'aspect géographique étant secondaire. Cela dit, nous nous en contentons dans ce mémoire puisque le but de l'interface est simplement de vérifier la faisabilité des différentes méthodes de codification du point de vue géométrique, et pas attributaire.

Dans ce qui suit, nous nous limiterons à l'explication des instructions VRML utilisées dans ce travail. VRML fonctionne selon un modèle hiérarchique de nœuds et de champs, un nœud pouvant contenir un ou plusieurs champs. Les champs peuvent à leur tour être considérés comme des nœuds comprenant des champs (récurrence).

Un objet est représenté par un nœud de type « Shape » comprenant deux champs : un champ de type appareance « Appareance » paramétrant l'attribut visuel de l'objet et un champ geometry de type « IndexedFaceSet » paramétrant la géométrie de l'objet. Le champ de type « Appareance » peut à son tour être considéré comme un nœud contenant un champ de type « Material » permettant de définir pour l'objet sa couleur, sa brillance, sa texture, *etc.*…

Il existe de nombreux types de champs pour décrire la géométrie d'un objet (champs pour décrire des sphères, des cubes, *etc.*…). Cependant, nous n'utilisons que le type « IndexedFaceSet » qui permet de décrire la totalité des objets tridimensionnels à faces planes. Le nœud geometry de type « IndexedFaceSet » contient le champ coord de type « Coordinate » permettant de décrire une liste de points (instruction « point [] ») avec leurs coordonnées et une liste de faces (instruction « coordIndex [] »). Dans cette dernière instruction, pour chaque face, nous indiquons la liste des points rencontrés en tournant autour de la face le long des arêtes dans le sens horloger et terminons par l'instruction « -1 » (fin de la face). L'index identifiant un point est son numéro d'ordre d'arrivée (en commençant par 0) dans l'instruction « point [] ».

Notons que le nœud IndexedFaceSet peut contenir l'instruction « Solid » qui peut prendre la valeur « TRUE » ou « FALSE ». Dans le cas de « TRUE », cela signifie que seul un côté des faces sera visible (celui pour lequel le sens de rotation dans l'instruction « coordIndex » est horloger). Ainsi, en indiquant « Solid FALSE », nous n'avons plus besoin de nous soucier dans quel sens tourner autour des faces lorsque nous les écrivons puisque les deux côtés sont représentés.

Voici un exemple de fichier VRML utilisant les notions décrites ci-dessus avec le dessin correspondant (figure 1.18) :

```
NavigationInfo {
  type "EXAMINE"
}

Shape {
        appearance Appearance {
                material Material {
                        diffuseColor 0 .5 .5
```

```
                specularColor .87 .25 .25
                ambientIntensity .157
                shininess .5
            }
        }
        geometry    IndexedFaceSet {
            solid FALSE
            coord Coordinate {
                point [
                    2 0 0,
                    2 2 0,
                    0 0 0,
                    0 2 0,
                    1 2 2,
                    1 0 2,
                ]
            }
            coordIndex [
                0, 1, 3, 2, -1,
                0, 5, 4, 1, -1,
                2, 5, 4, 3, -1,
                0, 5, 2, -1,
                1, 4, 3, -1,
            ]
        }
}
```

Figure 2.18. Dessin issu d'un fichier VRML

2.5.1.3. PERL

PERL est l'un des langages de programmation le plus en vogue sur internet. Sa syntaxe est assez similaire à C bien que généralement plus simple à implémenter et plus compacte.
Nous retrouvons les instructions habituelles telles que « for », « while do », « if else », *etc.*… PERL permet la manipulation de variables numériques ou de chaînes de caractères, la manipulation de tableaux à plusieurs dimensions, la lecture et écriture de fichiers, l'utilisation

de fonctions mathématiques, *etc.*… Ce langage ne requiert pas de compilateur pour transformer un programme en code opérationnel. Il suffit d'écrire le programme (fichier d'extension « pl ») et de signifier à PERL qu'il doit l'exécuter.

2.5.2. Description du programme PERL

Les codifications décrites dans ce chapitre n'ont pas toujours été programmées dans leur totalité : cet algorithme est un prototype permettant de démontrer la faisabilité de la codification au niveau de la programmation. Si la codification devait être programmée dans le but d'être utilisée dans des levés réels, nous ne pourrions pas nous contenter d'une interface VRML mais plutôt d'un logiciel plus adapté au travail des géomètres tel que AUTOCAD. L'algorithme est contenu dans un fichier nommé « geocod3D.pl ».

2.5.3.1. Structure du fichier input

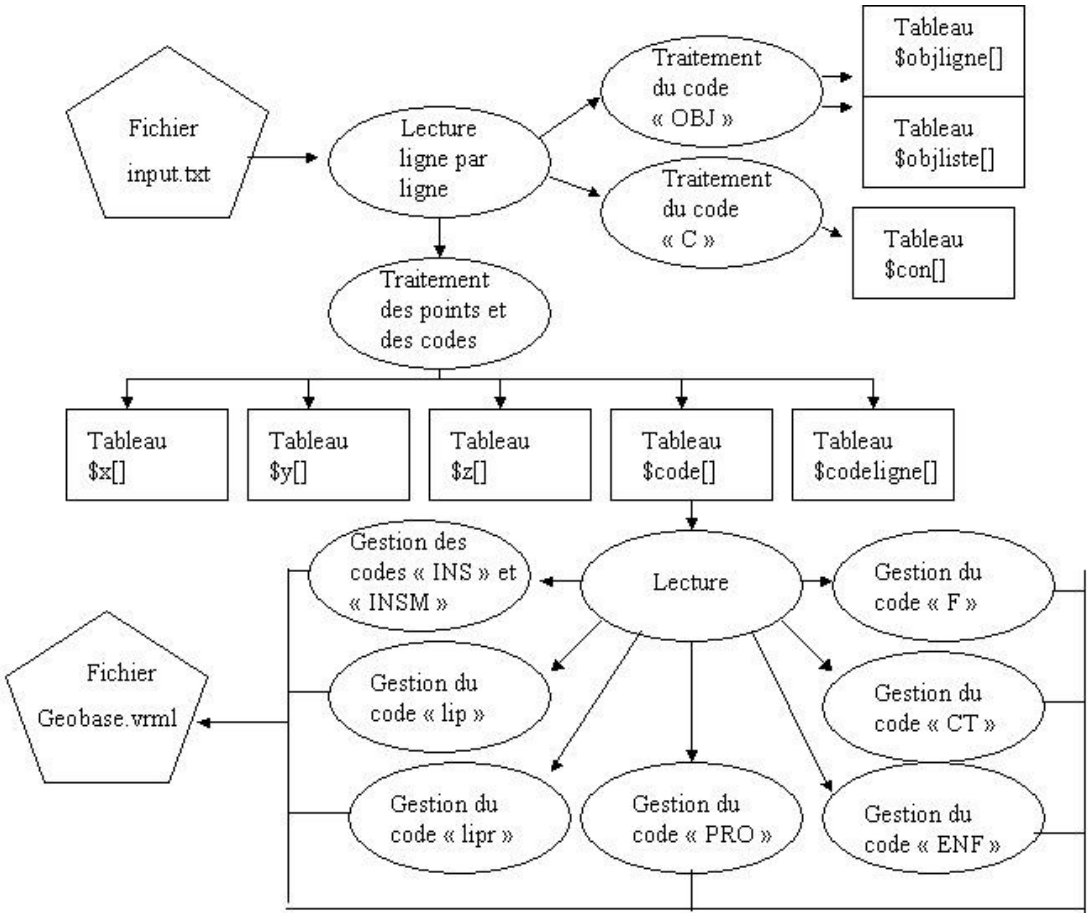
Après traitement des données provenant de la station totale, ces dernières doivent être écrites dans un fichier de type « txt » répondant à la structure suivante :

Identifiant ; X ; Y ; Z ; code ;

Les points sont bien évidemment placés ligne par ligne dans l'ordre dans lequel ils ont été levés ainsi que les codes sans mesure.

2.5.3.2. Structure générale du programme

Figure 2.19. Structure générale du programme PERL



Le fichier input.txt est lu ligne par ligne par le programme. Trois cas de figure peuvent alors se présenter :

1) Le programme tombe sur le code « OBJ » (objet) : il stocke le nom de l'objet dans le tableau $objligne avec comme index le numéro de la ligne dans le fichier input.txt. Ce tableau permet la correspondance entre une mesure et l'objet auquel elle se rapporte. A chaque nouvel objet rencontré, le nom de l'objet est également stocké dans le tableau $objliste (tableau comportant tous les noms d'objets sans redondance).
2) Le programme tombe sur le code « C » (connectivité) : le nom de chaque face est complété par le nom de l'objet correspondant (chaque face comporte ainsi un identifiant unique). Les faces sont ensuite stockées dans le tableau $con dans l'ordre codé par l'utilisateur. Notons qu'un tableau $countcon stocke le nombre de faces par ligne de connectivité afin de pouvoir différencier les lignes du fichier input, une ligne correspondant à une face ou à une arête selon la méthode de connectivité utilisée (voir le point 2.3.1.2). C'est aussi ici que la connectivité implicite « BT » est reconstruite et stockée en « face par face » dans le tableau $con (voir tableau 2.3).
3) Le programme tombe sur un point : il stocke les coordonnées du point dans les tableaux $x, $y et $z. Le code du point est stocké dans le tableau $code. Le numéro de la ligne de la mesure dans le fichier input est stocké dans le tableau $codeligne afin de pouvoir faire la correspondance entre mesure et objet correspondant (grâce au tableau $objligne).

Après lecture du fichier « input », le programme lit ligne par ligne le tableau « code ». En fonction du code sur lequel il tombe, il établit une série d'instructions :

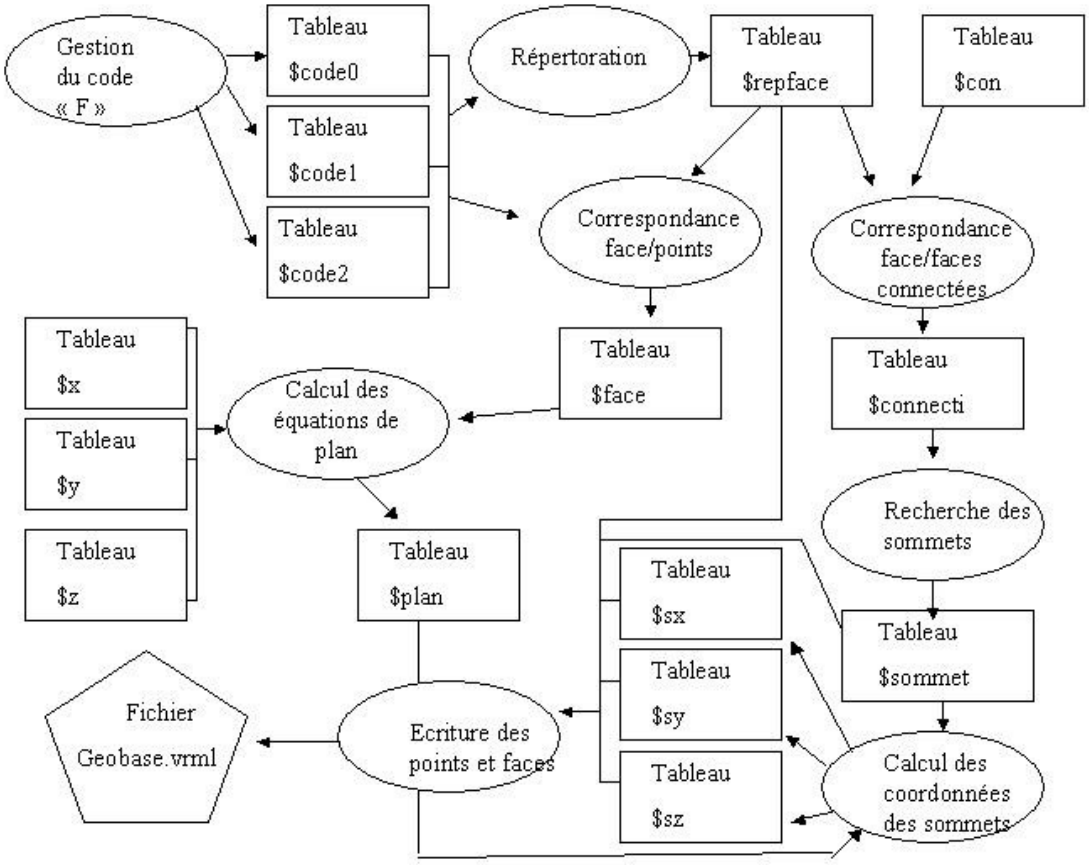
- Gestion du code « F » (codification par construction de plans)

- Gestion du code « CT » (contour de face)
- Gestion du code « ENF » (enfoncement)
- Gestion du code « PRO » (proéminence)
- Gestion du code « lipr » (liaison par parallélépipède)
- Gestion du code « lip » (liaison par face)
- Gestion des codes « INS » et « INSM » (répétition d'objet)

Dans tous les cas, ces différents traitements permettent l'écriture de points et de faces dans le fichier « geobase.vrml ».

2.5.3.3. Gestion du code « F » (codification par construction de plans)

Figure 2.20. Structure du traitement du code « F » (codification par construction de plans)



Lorsque le programme tombe sur le code « F », il stocke les noms des faces auquel le point appartient dans les tableaux $code0, $code1 et $code2. Ces faces sont ensuite répertoriées dans le tableau $repface (toutes les faces du levé y sont présentes sans redondance). Les tableaux $code0, $code1 et $code2 permettent alors de créer un tableau $face regroupant la liste des points levés pour chaque face du tableau $repface (au minimum trois).

Ensuite, pour chaque ligne du tableau $face, le programme calcule et stocke les quatre paramètres d'équation du plan de la face correspondante dans le tableau $plan. Rappelons que l'équation d'un plan est :

a*x + b*y + c*z = -d

Avec trois points, nous obtenons un système de trois équations à trois inconnues (l'inconnue d s'annule) :

a*x1 + b*y1 + c*z1 = 0
a*x2 + b*y2 + c*z2 = 0
a*x3 + b*y3 + c*z3 = 0

Une fois a, b et c trouvés, d peut être déduit d'une seule équation. Remarquons que seulement les trois premiers points rencontrés par l'algorithme concernant une face sont utilisés dans le calcul de l'équation du plan de celle-ci. L'ajustement à un nuage de plus de trois points par moindre carré peut être envisagé mais n'est pas développé dans cet algorithme.

Le tableau $con (liste des faces connectées) est ensuite trié afin d'obtenir une configuration « face par face » dans le tableau $connecti dans l'ordre du tableau $repface. Le tableau $connecti est traité à son tour pour rechercher les sommets des faces : un sommet est caractérisé par l'intersection de trois plans connectés entre eux (information disponible dans le tableau $connecti). Ainsi, pour chaque sommet, les trois faces d'intersections sont stockées dans le tableau $sommet.

Les coordonnées des sommets peuvent maintenant être calculées au moyen des tableaux $sommet et $plan ($repface permet de faire la correspondance entre les faces de $sommet et de $plan). Il suffit alors de résoudre le système de trois équations à trois inconnues suivant :

a1*x + b1*y + c1*z = -d1
a2*x + b2*y + c2*z = -d2,
a3*x + b3*y + c3*z = -d3

En formulant ce système sous forme matricielle cela donne :

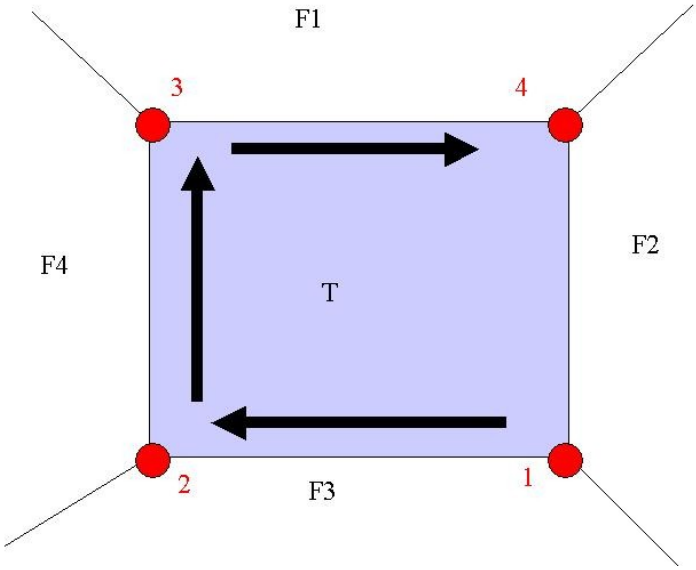
$$\begin{bmatrix} a1 & b1 & c1 \\ a2 & b2 & c2 \\ a3 & b3 & c3 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -d1 \\ -d2 \\ -d3 \end{bmatrix}$$

Cette équation matricielle peut être résolue par l'algorithme de Gauss-Jordan (source : BASTIN, 2004). Le principe de cet algorithme est de d'appliquer des inversions et des combinaisons linéaires entre les lignes des matrices (sans changer le système bien évidemment) afin d'obtenir la matrice identité pour la première matrice. Ainsi, la troisième prend la valeur de la matrice inconnue.

Les coordonnées des points sont ensuite écrites dans le fichier VRML simplement dans l'ordre dans lequel elles figurent dans les tableaux $sx, $sy et $sz. L'écriture des faces est un peu plus complexe et mérite une description un peu plus approfondie.

Nous avons vu qu'une face est décrite en VRML en dressant la liste des sommets qu'elle contient en la longeant le long de ses arêtes. Or, cet algorithme ne fait pas appel à la notion d'arête. La figure 2.21 illustre l'établissement d'un contour d'une face en connaissant uniquement les trois faces formant chaque sommet par intersection.

Figure 2.21. Illustration du contour de face dans la codification par construction de plans



Voici liste des faces comprenant chaque sommet :
1 : F2, F3, T
2 : F3, F4, T
3 : F4, F1, T
4 : F2, F1, T

Nous pouvons constater que deux sommets adjacents (appartenant à la même arête) appartiennent toujours à deux faces communes. Ainsi, en partant du point 1, l'algorithme choisit aléatoirement entre le point 2 (F3 et T en commun) et 4 (F2 et T en commun). Imaginons qu'il choisisse le point 2. Sachant qu'il est déjà passé par le point 1, il ne reste plus qu'un point candidat (appartenant à deux faces communes) : 3. L'algorithme s'arrête lorsqu'il ne trouve plus de points candidats.

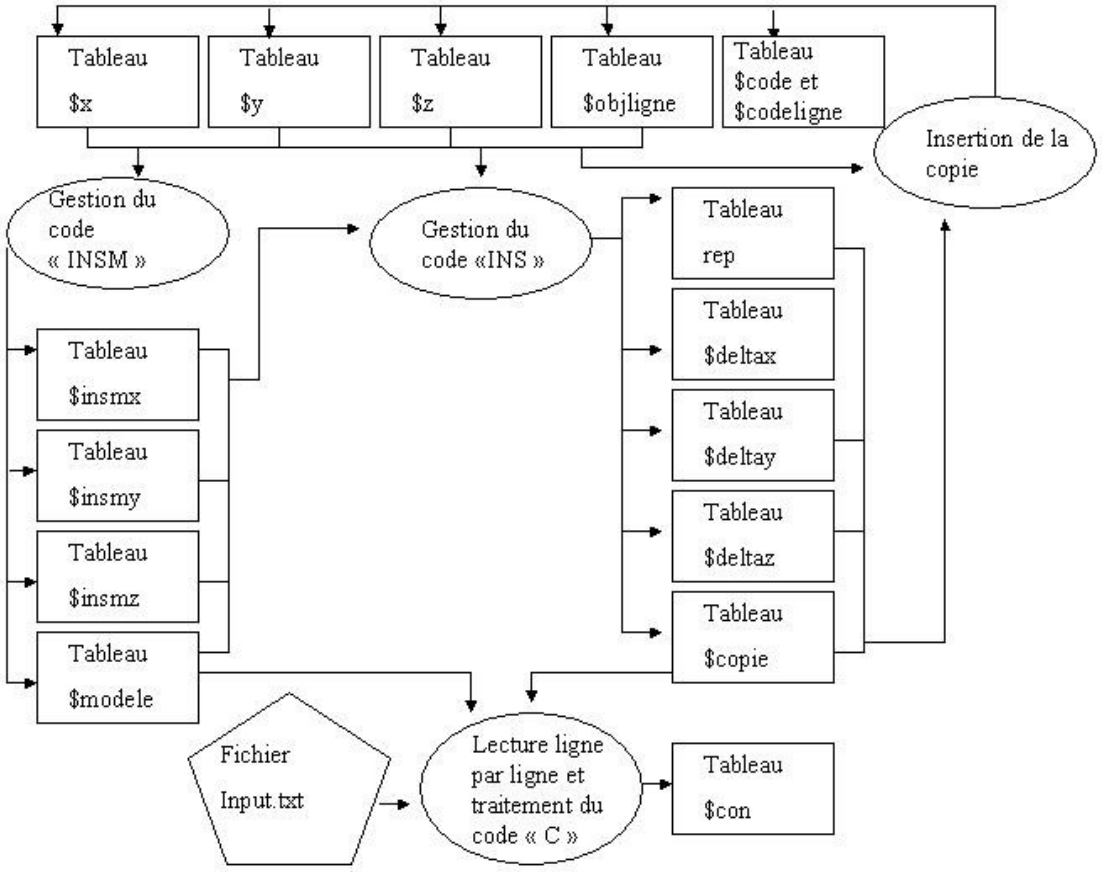
Malheureusement, cet algorithme ne fonctionne pas lorsqu'il est confronté à des sommets appartenant à plus de trois faces (le nombre de points candidats est alors plus grand que deux). Ce type d'objet nécessite donc une codification de la connectivité plus détaillée (notamment en ce qui concerne les sommets à plus de trois arêtes).

Remarquons également que l'algorithme calcule les paramètres d'équation des plans à partir de trois points levés. Nous pourrions imaginer un ajustement par moindre carré des plans à un nuage de plus de trois points (autant qu'on en a levé sur le terrain), ce qui augmenterait la précision du dessin.

2.5.3.4. Gestion des codes « INS » et « INSM » (répétition d'objets)

L'algorithme développé ne tient compte que des répétitions d'objet sans orientation (orientation identique à l'objet modèle). Une stratégie de programmation de la gestion des copies d'objets avec orientation sera néanmoins mentionnée à la fin de ce point.

Figure 2.22. Structure du traitement des codes « INS » et « INSM » (répétition d'objets)



Lorsque le programme tombe sur le code « INSM », il utilise les tableaux $x, $y, $z et $objligne pour stocker les coordonnées du point d'insertion dans les tableaux $insmx, $insmy, $insmz et le nom de l'objet modèle dans le tableau $modele.

Lorsque le programme tombe sur le code « INS », il utilise les tableaux de coordonnées de points ($x, $y, $z) et les tableaux de coordonnées de points d'insertion ($insmx, $insmy, $insmz) pour calculer les paramètres delta de translation. En effet, une copie d'objet sans orientation peut être dessinée en additionnant un paramètre delta aux coordonnées de tous les points de l'objet modèle. Les paramètres delta sont calculés au moyen des formules suivantes :

deltax = inscx - insmx
deltay = inscy - insmy
deltax = inscz – insmz

inscx, inscy et inscz sont les coordonnées du point d'insertion sur l'objet copié et insmx, insmy et insmz sont les coordonnées du point d'insertion sur l'objet modèle.

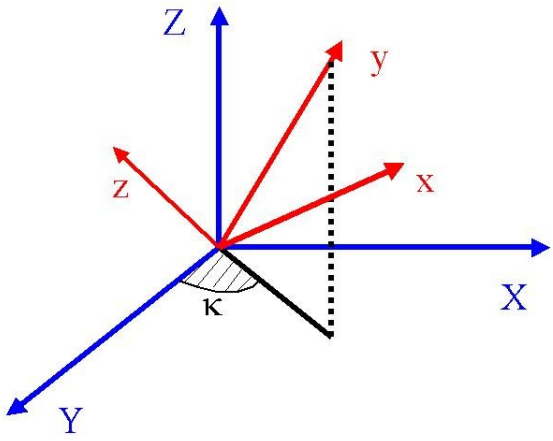
Une fois ces paramètres calculés, ils sont stockés dans des tableaux de type « delta » (un par coordonnée). Le nom de l'objet copié est stocké dans le tableau « copie » et le nombre de répétitions de la copie dans le tableau « rep » (issu du paramètre « REP » du code « INS.REP.x »).

Ensuite, à chaque point levé de l'objet modèle, l'algorithme additionne les paramètres delta pour obtenir les coordonnées des points (levés fictivement) de l'objet copié. Ces points sont ensuite stockés dans les tableaux $x, $y, $z. Leurs codes (identiques à ceux de l'objet modèle) sont stockés dans le tableau $code et $codeligne. Ainsi, les points de l'objet copié seront traités normalement par le programme comme s'ils avaient été levés. Cette étape est répétée autant de fois qu'il y a de copie de l'objet modèle. A chaque nouvelle itération, les paramètres delta sont incrémentés des delta de la copie précédente. Ainsi, les différentes copies d'un même modèle via le paramètre « REP » seront dessinées à intervalle régulier (voir figure 2.16).

Parallèlement à cela, le fichier « input.txt » est relu ligne par ligne et effectue l'instruction suivante quand il tombe sur une ligne de connectivité (code « C ») concernant l'objet modèle : il ajoute le nom de l'objet copié aux noms des faces et les stocke dans le tableau « con ». Ainsi, les faces de chaque objet copié sont identifiées de façon unique. Remarquons que pour chaque copie d'un modèle via le paramètre « REP », le nom de objet copié est suivi de son numéro de copie. Par exemple : COPIE1, COPIE2, COPIE3, *etc.*…

Voyons maintenant la stratégie à adopter pour gérer la copie d'objets avec orientation. Nous avons vu au point 2.4.5.2 qu'il était possible d'établir un système de coordonnées local « modèle » et « copie » à partir d'un point d'insertion et de deux points d'orientation. Au moyen des constructions géométriques mentionnées au point 2.4.5.2, nous pouvons retrouver les équations des droites représentant les trois axes dans l'espace ainsi que les coordonnées de l'origine du système local dans le système global. Considérons maintenant les droites parallèles aux axes du système local passant par l'origine du système global. Soient X, Y, Z les axes d'un système global, x, y, z les axes d'un système local, Ω, ϕ, κ les angles de rotation respectivement autour de x, y, z du système local par rapport au système global, la figure 2.22 nous montre que κ peut être retrouvé par l'angle entre la projection orthogonale de l'axe y dans le plan XY et l'axe Y.

Figure 2.23. Comment retrouver les angles de rotation d'un système d'axe local par rapport à un système d'axe global



De même, ϕ peut être retrouvé par l'angle entre la projection de y dans le plan YZ et Y et Ω peut être retrouvé par l'angle entre la projection de z dans le plan XZ et Z. Soient δx, δy, δz, les paramètres de translation d'un système d'axes local par rapport à un système d'axes global et ox, oy, oz les coordonnées de l'origine du système local par rapport au système global, δx, δy et δz correspondent respectivement à ox, oy et oz. Nous sommes maintenant en mesure de retrouver tous les paramètres de transformation d'un système global vers un système local. Les paramètres de transformation d'un système local vers un système global sont les opposés des paramètres d'un système global vers un système local. Les paramètres de transformation entre deux systèmes locaux peuvent être retrouvés en appliquant les constructions géométriques mentionnées ci-dessus entre les deux systèmes locaux dans le système global.

Pour effectuer la transformation des coordonnées d'un point d'un système global vers un système local, nous appliquons la formule matricielle suivante :

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = [R] * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} \delta x \\ \delta y \\ \delta z \end{bmatrix}$$

$$[R] = [R\kappa] * [R\phi] * [R\Omega]$$

$$[R\kappa] = \begin{bmatrix} \cos\kappa & -\sin\kappa & 0 \\ \sin\kappa & \cos\kappa & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad [R\phi] = \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix}$$

$$[R\Omega] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\Omega & -\sin\Omega \\ 0 & \sin\Omega & \cos\Omega \end{bmatrix}$$

Pour insérer un objet copié avec orientation, le programme effectue donc les opérations suivantes :

- Etablissement du système local « modèle »

- Etablissement du système local « copie »
- Transformation des coordonnées de l'objet modèle du système global vers le système local « modèle »
- Transformation des coordonnées de l'objet modèle du système local « modèle » vers le système local « copie » (on obtient un objet copié)
- Transformation des coordonnées de l'objet copié du système local « copie » vers le système global.

2.5.3.5. Gestion du code « lip » (liaison par plan)

Cet algorithme ne tient compte que des liaisons fermées (avec face inférieure et supérieure). L'identification des faces n'est pas non plus gérée par l'algorithme ainsi que la gestion des hauteurs au moyen d'un paramètre (tout comme pour le code « lipr »).

Les opérations effectuées par le programme dépendent du paramètre numérique sur lequel il tombe (0, 1, 2 ou 3)

Lorsque le programme tombe sur le paramètre « 0 » (ouverture d'une liaison), il effectue les opérations suivantes :
- Mémorisation de l'identifiant du point (point de départ)
- Stockage du numéro d'ordre d'arrivée du point (0) dans le tableau $sol
- Ecriture des coordonnées du point dans le fichier geobase.vrml

Lorsque le programme tombe sur le paramètre « 2 » (point donnant la hauteur), il calcule les paramètres deltax, deltay et deltaz. Ces paramètres permettent d'établir les coordonnées des points supérieurs lorsque la liaison effectuée à partir de points inférieurs (et vice versa). Ils sont obtenus grâce à la formule suivante :

deltax = x2 – x1
deltay = y2 – y1
deltaz = z2 – z1

(x1, y1, z1) sont les coordonnées du points d'ouverture et (x2, y2, z2) sont les coordonnées du point donnant la hauteur.

Le numéro d'ordre du point (1) est ensuite stocké dans le tableau $toit et les coordonnées du point sont écrites dans le fichier geobase.vrml.

Lorsque le programme tombe sur le paramètre 1, il calcule les coordonnées du point supérieur (ou inférieur) au moyen de la formule :

xs = xi + deltax
ys = yi + deltay
zs = zi + deltaz

(xi, yi, zi) sont les coordonnées du point rattaché au code en cours.

Il stocke le numéro d'ordre d'arrivée du point en cours (nombre pair) dans le tableau $sol et le numéro d'ordre d'arrivée du point calculé dans le tableau $toit (nombre impair), puis il écrit les coordonnées des points dans le fichier geobase.vrml (en respectant bien l'ordre d'arrivée).

Lorsque le programme tombe sur le paramètre « 3 », il effectue les mêmes opérations que pour le paramètre « 1 » puis écrit les faces dans le fichier geobase.vrml, les tableaux $sol et $toit permettant de contourner les faces :

- face latérale : $sol[i], $sol[i+1], $toit[i+1], $toit[i]
- face supérieure : tous les éléments du tableau $toit
- face inférieure : tous les éléments du tableau $sol

Précisons que même si les noms de variable peuvent laisser penser le contraire, cette codification ne s'applique pas uniquement à des liaisons avec des points au sol et des points servant de toit.

2.5.3.5. Gestion du code « lipr » (liaison par parallélépipède)

La structure de programmation du code « lipr » est assez similaire à celle du code « lip ». Nous n'entrerons donc pas trop dans le détail concernant les tableaux et variables utilisés. Précisons juste que les paramètres « 2 » et « 5 » permettent respectivement le calcul des paramètres deltah (hauteur) et deltal (largeur).

Dans le programme développé, les faces « frontales » sont construites à chaque point « liaison » puis sont reliées à leur prédécesseur par des faces latérales. Hélas, la construction des parallélépipèdes de cette façon est géométriquement incorrecte (rappelons qu'il ne s'agit que d'un prototype) : a chaque construction de face « frontale » (quand on tombe sur le paramètre « 1 » ou « 4 »), le programme utilise les paramètres deltah et deltal pour retrouver les trois points manquants (ou un seul point manquant dans le cas de la première face « frontale »):

(xl, yl, zl) = (x, y, z) + (deltalx, deltaly, deltalz)
(xh, yh, zh) = (x, y, z) + (deltahx, deltahy, deltahz)
(xlh, ylh, zlh) = (x, y, z) + (deltalhx, deltahy, deltahz) + (deltalx, deltaly, deltalz)

Figure 2.24. Construction géométriquement incorrecte des parallélépipèdes (vue 3D)

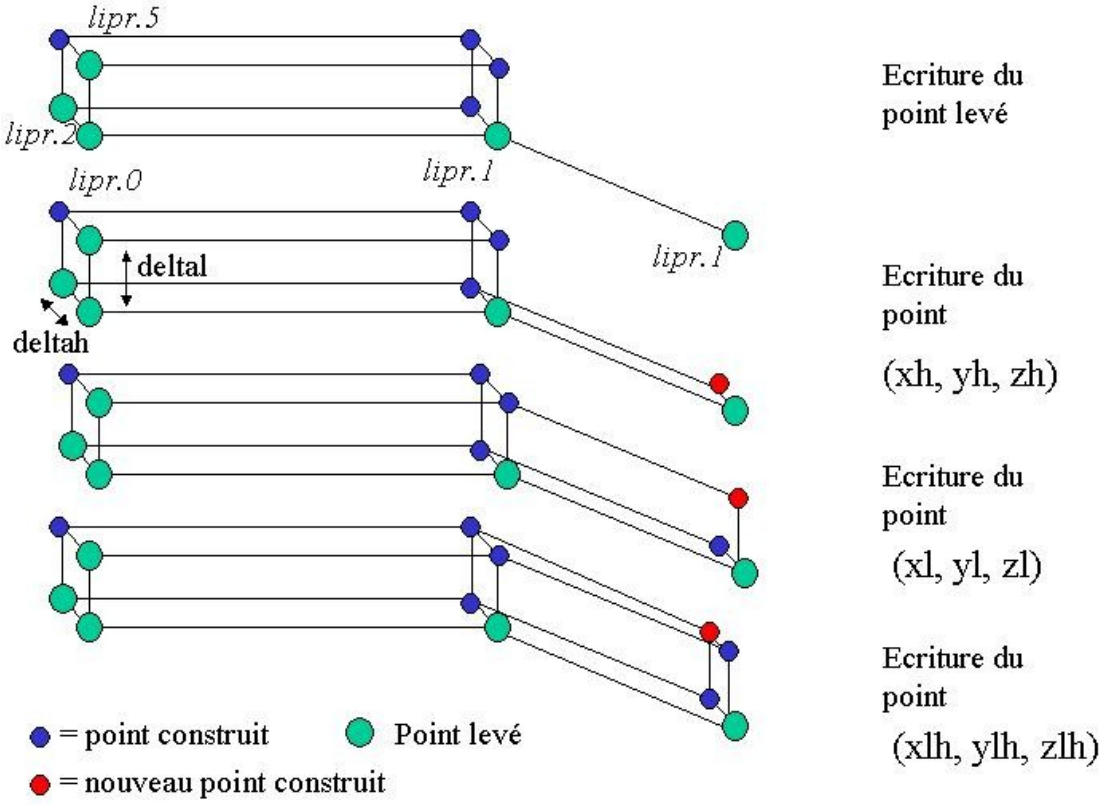
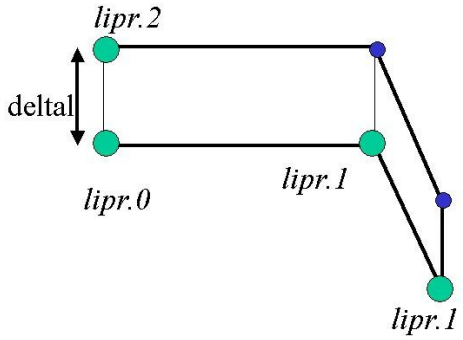


Figure 2.25. Construction géométriquement incorrecte des parallélépipèdes (vue 2D)



La figure 2.24 nous montre que la largeur des parallélépipèdes n'est pas constante avec cette construction. De plus, un angle plus grand ou égal à 90° dans la liaison donne lieu à une construction avec une face de moins ou deux faces qui se coupent.
Chaque parallélépipède de la liaison est construit en deux étapes. On calcule les coordonnées de la première face frontale puis on calcule les coordonnées de la seconde face frontale (les faces latérales sont déduites des deux autres).

Décrivons maintenant la stratégie algorithmique à appliquer pour obtenir une construction géométriquement correcte (gardant une largeur et une hauteur du parallélépipède constante). Le premier parallélépipède est construit comme pour la méthode précédente (le premier parallélépipède a toujours une construction géométriquement correcte). Le second

parallélépipède doit être de forme et de dimension en hauteur et largeur identique au second. Dans ce but, il est nécessaire de reconstruire la face arrière du parallélépipède (qui n'est plus la même que la face avant du parallélépipède précédent). La face avant sera construite de manière parfaitement identique à la face arrière (en appliquant les deltah et deltal trouvés à partir de la face arrière sur la face avant).

A partir des points donnant la hauteur et la largeur du premier parallélépipède nous pouvons calculer leur distance par rapport au point d'ouverture (respectivement dh et dl) au moyen du théorème de Pythagore appliqué en 3D :

$D(x1\text{-}x2) = \sqrt{\{(x2\text{-}x1)^2 + (y2\text{-}y1)^2 + (z2\text{-}z1)^2\}}$

Soient A un point de liaison terminant le parallélépipède précédent, B le point de liaison terminant le parallélépipède en cours, L1 le point donnant la largeur de la face avant du parallélépipède précédent, η le plan comprenant les points A, B, L1, nous construisons une droite h perpendiculaire au plan η et comprenant le point B. Le point H2 donnant la hauteur de la face arrière du parallélépipède en cours de construction est le point appartenant à h se trouvant à une distance dh de B.

Soient A un point de liaison terminant le parallélépipède précédent, B le point de liaison terminant le parallélépipède en cours, H1 le point donnant la hauteur de la face avant du parallélépipède précédent, λ le plan comprenant les points A, B, H1, nous construisons une droite l perpendiculaire au plan λ et comprenant le point B. Le point L2 donnant la largeur de la face arrière du parallélépipède en cours de construction est le point appartenant à h se trouvant à une distance dh de B.

Le dernier point de la face arrière (HL2) est trouvé en additionnant les paramètres deltah et deltal trouvés respectivement à partir des couple de points (B, H2) et (B, L2).

Figure 2.26. Construction géométriquement correcte des parallélépipèdes (vue 3D)

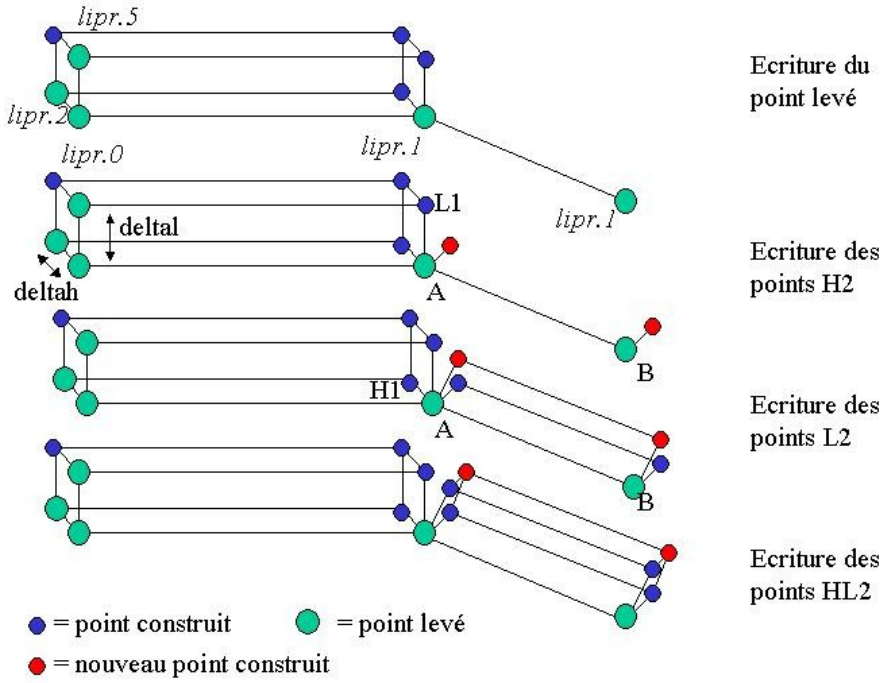
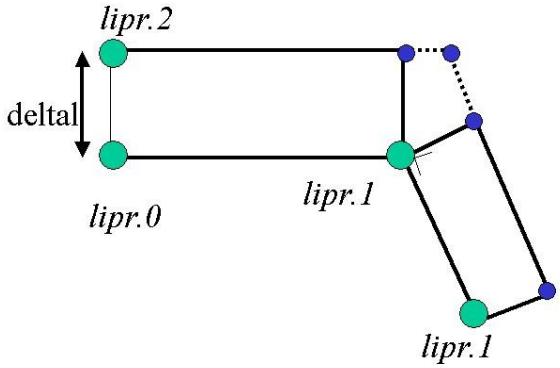


Figure 2.27. Construction géométriquement correcte des parallélépipèdes (vue 2D)



Comme le montre la figure 2.27, les largeurs (et les hauteurs) des parallélépipèdes sont bien constantes. Les quatre segments latéraux homologues peuvent être prolongés afin d'établir, au moyen de leur intersection, une face de jointure remplaçant les faces avant et arrière de deux parallélépipèdes consécutifs.

2.5.3.6. Gestion du code « CT » (contour de face)

L'algorithme développé pour le code « CT » permet d'associer une face à un objet défini au moyen d'autres codifications mais ne permet pas d'utiliser une face issue du code « CT » comme base d'un enfoncement ou d'une proéminence.

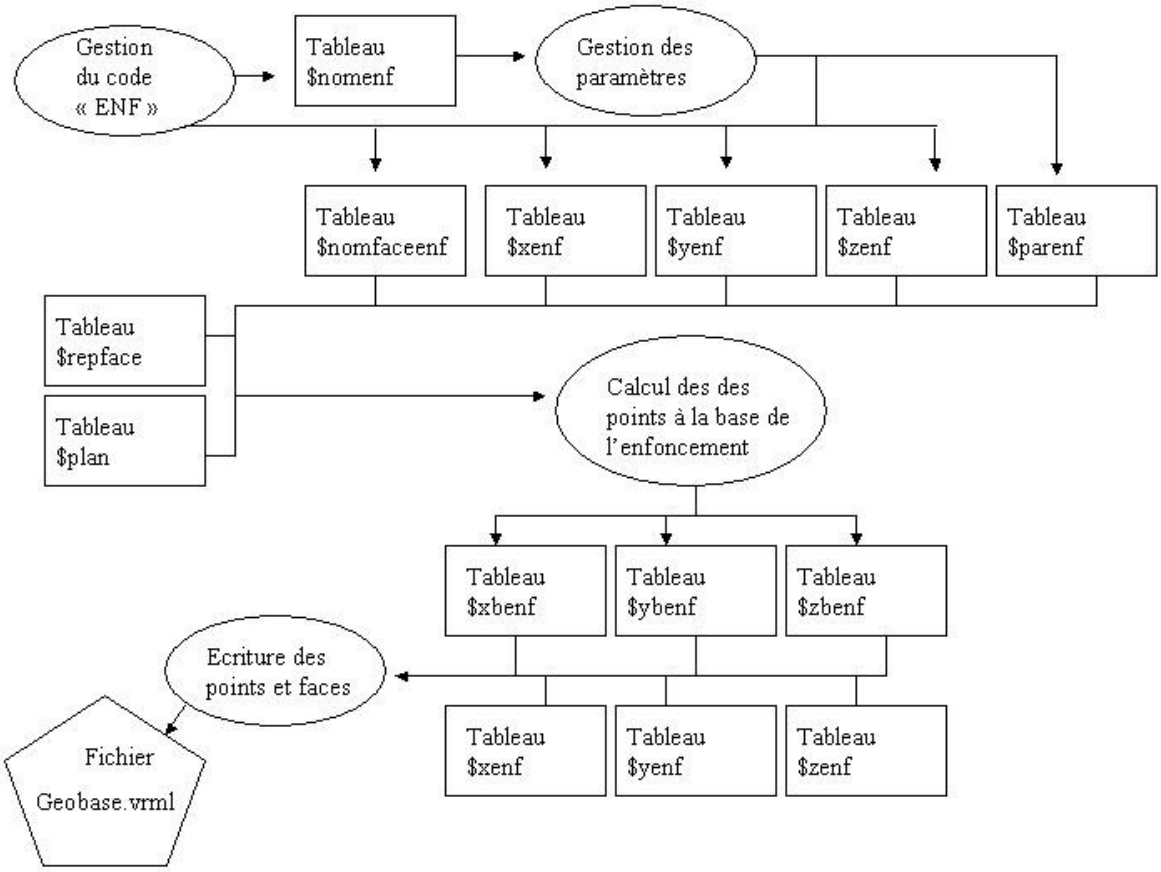
Lorsque le programme tombe sur le code « CT », il stocke les coordonnées du point dans les tableaux $xct, $yct et $zct ainsi que le nom de la face traitée (trouvé dans le paramètre du code) dans le tableau $nomct. Notons qu'au préalable, le nom de l'objet auquel appartient la face (dernier code « OBJ » dans le fichier input.txt) a été ajouté à l'identifiant de la face pour le rendre unique.

Une fois la boucle sur les différents codes terminée, pour chaque face issue du code « ct », toutes les coordonnées retrouvées dans les tableaux de type « ct » sont écrites dans le fichier geobase.vrml. Les faces sont ensuite écrites dans le fichier geobase.vrml en se référant au numéro d'arrivée des points et au tableau « $nomct » (qui établit le lien entre un point et la face à laquelle il appartient).

2.5.3.7. Gestion du code « ENF »

Cet algorithme ne permet pas l'écriture de plus d'un enfoncement par face pour une raison technique évoquée à la fin de ce point.

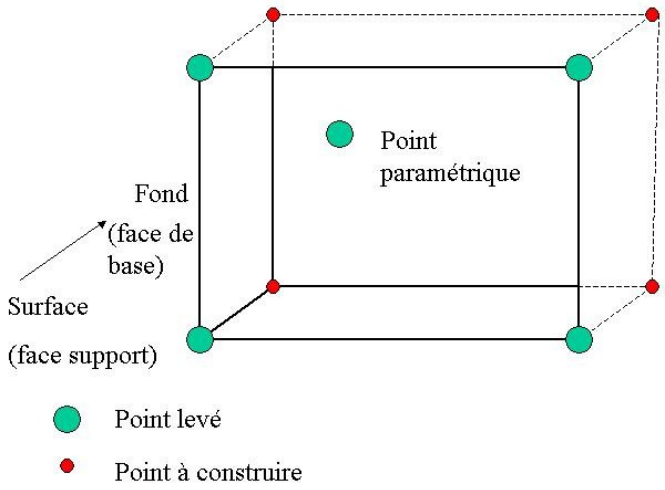
Figure 2.28. Structure du traitement du code « ENF » (enfoncement)



Lorsque le programme tombe sur le code « enf » (premier point d'un enfoncement), il stocke le nom de l'enfoncement (paramètre du code) dans le tableau « nomenf », les coordonnées du point en cours dans les tableaux $xenf, $yenf, $zenf et le nom de la face support de l'enfoncement paramétré dans le code (auquel on ajoute toujours le nom de l'objet) dans le tableau $nomfaceenf. Le tableau « nomenf » permet à l'algorithme de faire le lien avec l'enfoncement en cours et donc sa face support pour les points suivants.

Concernant les autres points d'un enfoncement, ils sont gérés comme pour une liaison (en fonction des paramètres d'ouverture, de liaison et de fermeture). Tous les points formant la liaison d'un enfoncement sont stockés dans les tableaux $xenf, $yenf et $zenf ainsi que leur face support dans le tableau $nomfaceenf. Un point particulier doit cependant être géré séparément : le point paramétrique (qui donne la profondeur de l'enfoncement). Celui-ci est stocké dans le tableau $parenf. A l'aide de ces différents tableaux et des tableaux $repface (répertoire des faces) et $plan (ensemble des paramètres d'équation de plan de toutes les faces rencontrées dans le levé), nous pouvons calculer les points formant la face du fond de l'enfoncement (face de base). Rappelons que les points mesurés sont du côté extérieur et forment le contour de l'enfoncement au niveau de sa surface (face support).

Figure 2.29. Illustration de l'enfoncement et des éléments utilisés dans l'algorithme



Rappelons l'équation d'un plan :

a*x + b*y + c*z = d

L'algorithme recherche les paramètres a', b', c', d' de l'équation du plan de la face support à l'aide des tableaux $nomfaceenf et $plan (le tableau $repface permet de faire le lien entre les deux). Une fois cette opération réalisée, connaissant les coordonnées du point paramétrique (tableau $parenf), l'algorithme peut calculer les paramètres a, b, c, d de l'équation du plan de la face de base en procédant comme suit.

Puisque le plan de base est parallèle au plan support, les vecteurs directeurs des deux plans sont identiques. Autrement dit :

a = a'
b = b'
c = c'

Il ne reste donc plus qu'un paramètre à retrouver : d'. Celui-ci peut facilement être calculé en résolvant l'équation à une inconnue (d) :

a*x + b*y + c*z = d

Où (x, y, z) sont les coordonnées du point paramétrique.

En se référant à la figure 2.29, nous pouvons remarquer que les points formant le contour de la face de base peuvent être retrouvés par projection orthogonale des points levés sur la face de base. Autrement dit, pour chaque point levé (tableaux de type « $enf »), le problème géométrique est le suivant : calculer les coordonnées du point appartenant à la fois au plan de base et à la droite d perpendiculaire au plan de base comprenant le point levé de coordonnées x0, y0, z0. Puisque d est perpendiculaire au plan de base, son vecteur directeur est identique

au vecteur directeur du plan de base (vecteur normal). L'équation paramétrique de cette droite est donc :

x = x0 + a*r
y = y0 + b*r
z = z0 + c*r

Où r ε R

Cela nous donne un système de 3 équations à 4 inconnues (x, y, z, r). Puisque le point recherché appartient au plan de base, nous pouvons rajouter l'équation paramétrique du plan de base comme quatrième équation dans le système en gardant le même nombre d'inconnues :

a*x + b*y + c*z = d
x = x0 + a*r
y = y0 + b*r
z = z0 + c*r

Ce système peut être reformulé par:

a*x + b*y + c*z = d
x – a*r = x0
y – b*r = y0
z – c*r = z0

Et donc sous forme matricielle :

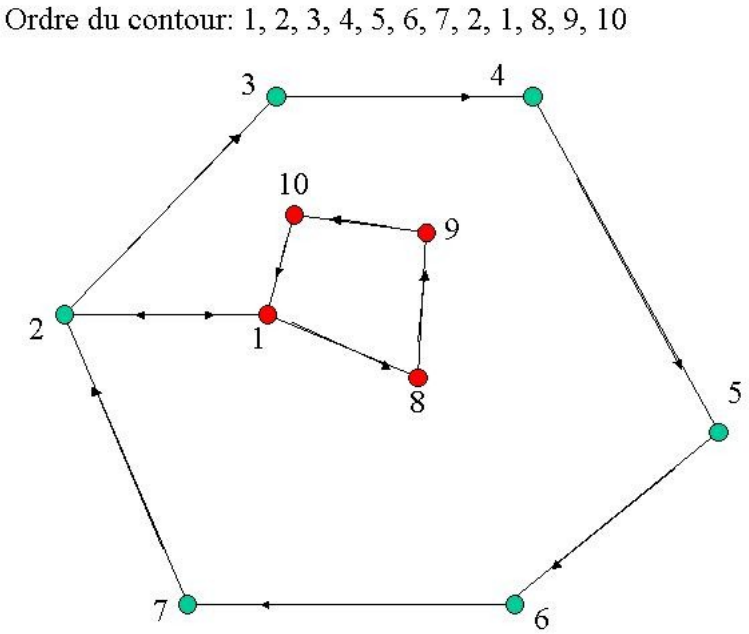
$$\begin{bmatrix} a & b & c & 0 \\ 1 & 0 & 0 & -a \\ 0 & 1 & 0 & -b \\ 0 & 0 & 1 & -c \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ r \end{bmatrix} = \begin{bmatrix} d \\ x0 \\ y0 \\ z0 \end{bmatrix}$$

Cette équation matricielle peut être résolue par l'algorithme de Gauss-Jordan. Les valeurs des points formant le contour de la face de base sont stockées dans les tableaux $xbenf, $ybenf, $zbenf.

Les coordonnées contenues dans les tableaux $xenf, $yenf, $zenf, $xbenf, $ybenf, $zbenf sont écrites dans le fichier geobase.vrml en alternant point levé et point construit homologue. Cette alternance est ensuite utilisée pour décrire les contours des faces de base et latérales dans le fichier geobase.vrml.

Un dernier détail reste à régler pour compléter l'algorithme : lorsque la face support est écrite dans le fichier geobase.vrml, il ne faut pas qu'elle « rebouche » l'enfoncement. C'est pourquoi le programme vérifie à chaque écriture de face si elle ne contient pas un enfoncement (grâce au lien entre les tableaux $repface et $nomfaceenf). Si il en détecte un, il établit le contour de face comme décrit dans la figure 2.30 :

Figure 2.30. Ecriture d'une face contenant un trou en VRML



Il n'y a pas d'autre moyen de dessiner une face avec un trou en VRML (impossibilité de « lever le crayon »). C'est pourquoi la gestion de plusieurs enfoncements par face deviendrait extrêmement complexe du point de vue de l'algorithme.

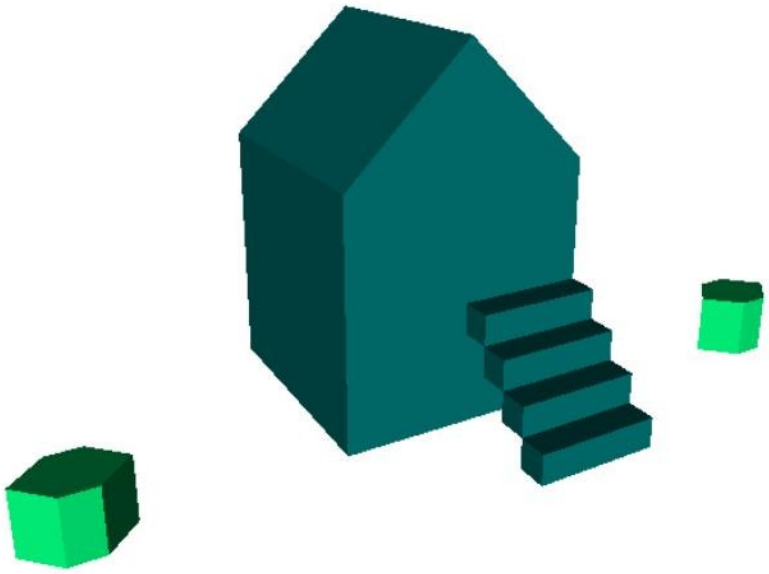
2.5.3.8. Gestion du code « PRO » (proéminence)

L'algorithme traitant le code « PRO » est similaire à celui gérant le code « ENF », à la différence que la face support et la face de base sont identiques. L'algorithme du code « PRO » est donc beaucoup plus simple. Remarquons que le problème des « faces trouées » en VRML est inexistant dans le cas de la proéminence et que donc, plusieurs proéminences par face peuvent être gérées par l'algorithme.

2.6. Validation

Les figures suivantes sont les résultats obtenus en géocodant quelques objets fictifs via notre programme. Tous les types de codes développés y sont représentés.

Figure 2.31. Exemple d'une géocodification par face, d'une répétition d'objet et de liaisons par plan



Dans la figure 2.31, la maison a été géocodée par la méthode « gécodification par construction de plans », l'escalier par la méthode « répétition d'objet » et les parallélépipèdes hexagonaux par la méthode « liaison par face ».

Figure 2.32. Exemple de contours de face

Figure 2.33. Exemple de liaison par parallélépipède

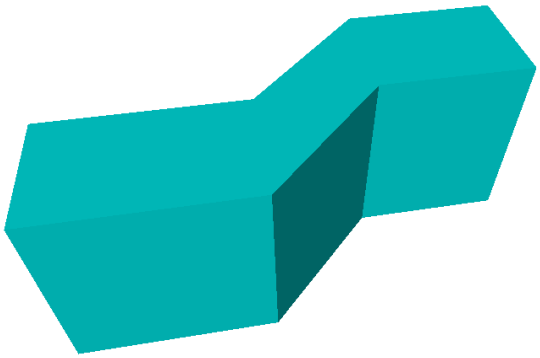
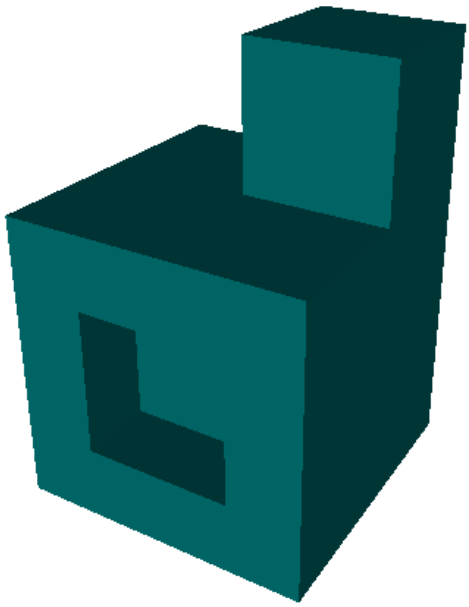


Figure 2.34. Exemple d'enfoncement et de proéminence



3. Estimation

Dans ce chapitre, nous allons montrer les résultats obtenus après l'application de notre codification sur deux levés de bâtiment. Nous critiquerons et comparerons ensuite les méthodes de codifications développées via une série de facteurs d'efficacité.

3.1. Tests de la codification

Deux levés ont été effectués pour tester notre codification. Le premier est une partie de l'institut de géographie (B11) et le second est le bâtiment de géographie physique (B12). Ces deux bâtiments sont situés au Sart-Tilman dans la commune d'Angleur, en province de Liège (Belgique).

Notons que pour des raisons de clarté, les points ne sont pas nommés sur les photographies comme ils l'ont été lors du levé.

3.1.1. Matériel utilisé et remarques sur les levés effectués

Voici la liste du matériel utilisé pour effectuer ces levés :

- une station totale Leica TCR 1205
- trois trépieds
- trois embases
- deux prismes circulaires
- un mètre ruban

Les deux prismes circulaires servent uniquement à mesurer les stations entre elles, tous les points des bâtiments étant levés au laser. La station totale nous permet une précision de l'ordre du millimètre au niveau des coordonnées X et Y, mais étant donné que les hauteurs de station sont mesurées au mètre ruban, la précision en Z est de l'ordre de 1 ou 2 centimètre(s). Les mesures ne sont pas toujours compensées puisque le but du travail effectué est simplement de tester notre codification. Après traitement des mesures, les points sont obtenus dans un système de coordonnées local propre au levé.

3.1.2. Test sur le B11

Ce bâtiment est levé selon un niveau de détail faible en utilisant les méthodes de codification suivantes :

- Codification par construction de plans : forme générale du bâtiment
- Enfoncement : creux dans le bâtiment
- Liaison par parallélépipède : murs décoratifs

Figure 3.1. Position des stations pour le levé du B11

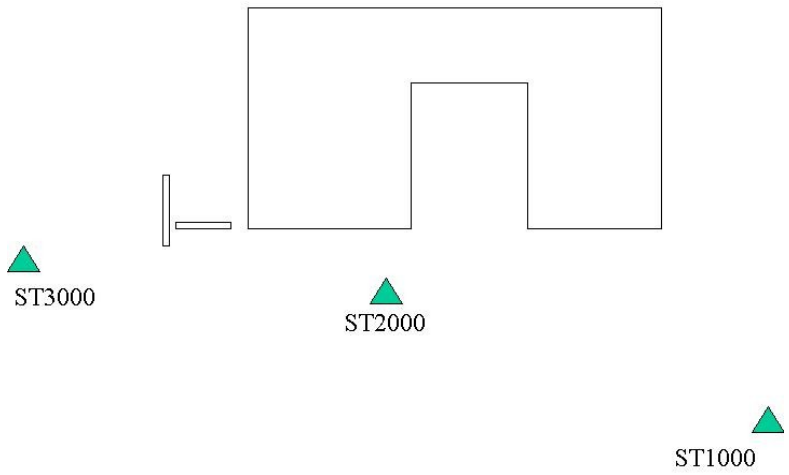


Figure 3.2. Représentation des points levés depuis la station ST1000

Tableau 3.1. Mesures et codes depuis la station ST1000

Mesure	Code
/	OBJ.MEAN
/	C.BT.4
PT1000	F.F3.F2.T
PT1001	F.F2
PT1002	F.F2.F1
PT1004	F.T.F1
PT1005	F.T.F1
PT1006	F.F1.F4.B
PT1007	F.F3.F2

Figure 3.3. Représentation des points levés depuis la station ST2000 (photo 1)

Remarquons que comme le montre la figure 3.3, la visibilité du point 4 est gênée par la voiture. Ainsi, la hauteur du point levé au laser est mesurée depuis le sol au mètre ruban puis indiquée dans la station totale comme « hauteur de prisme ».

Figure 3.4. Représentation des points levés depuis la station ST2000 (photo 2)

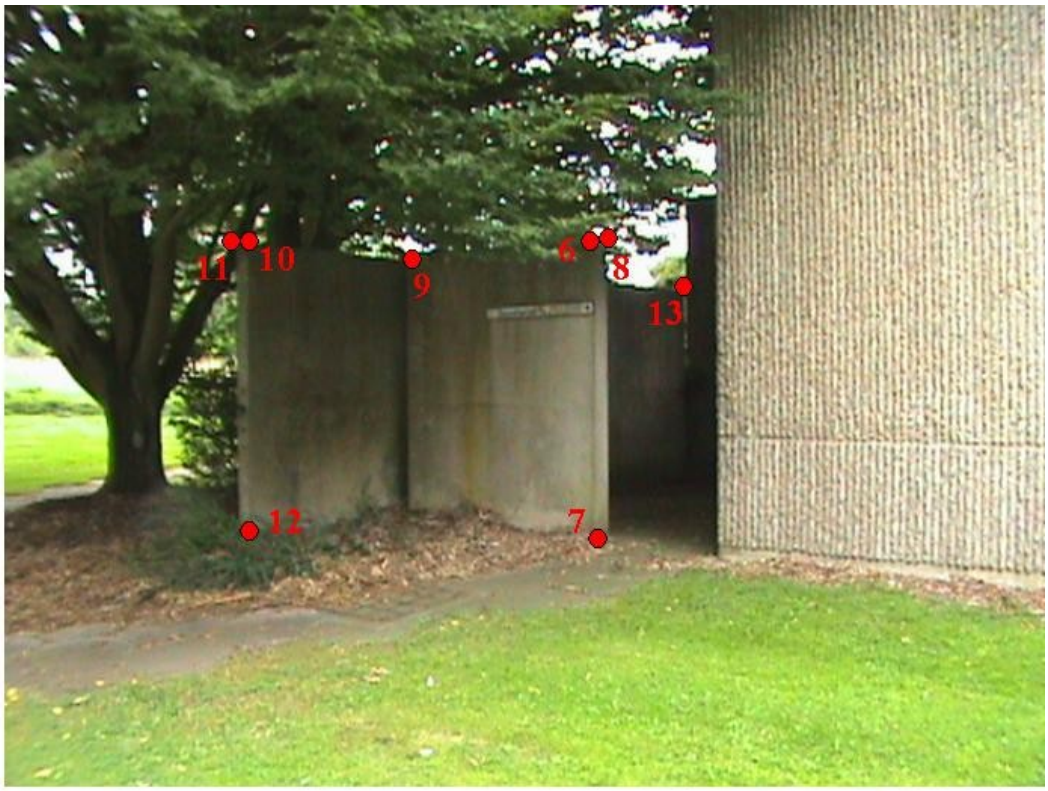


Tableau 3.2. Mesures et codes depuis la station ST2000

Mesure	Code
PT2000	ENF.CON.F1
PT2001	CON.PAR
PT2002	CON.1
PT2003	CON.1
PT2004	F.F2.F1CON.3
PT2005	F.F1.B
/	OBJ.MUR1
PT2006	lipr.0
PT2007	lipr.2
PT2008;	lipr.5
PT2009	lipr.4
/	OBJ.MUR2
PT2010	lipr.0
PT2011	lipr.5
PT2012	lipr.2
PT2013	lipr.4

Figure 3.5. Représentation des points levés depuis la station ST3000

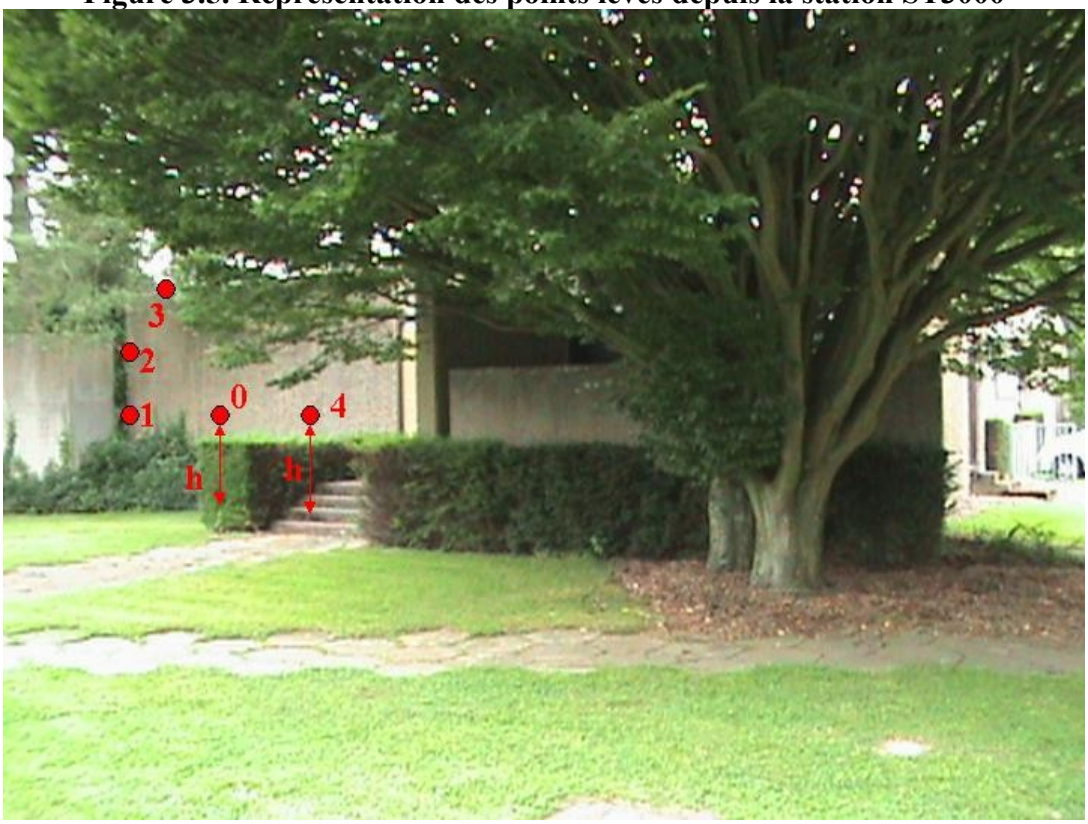
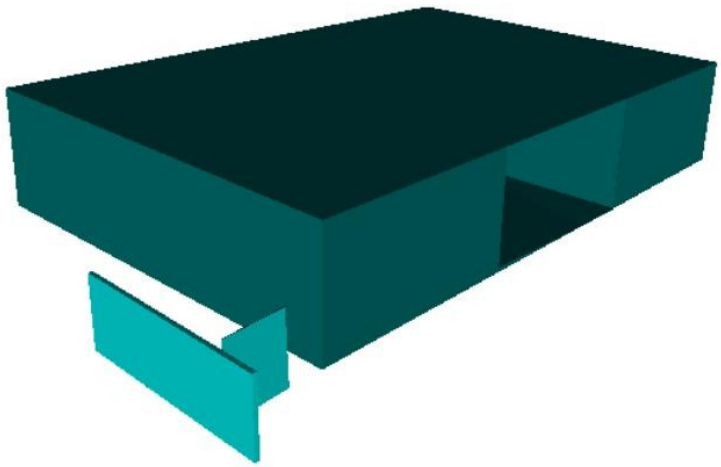


Tableau 3.3. Mesures et codes depuis la station ST3000

Mesure	Code
/	OBJ.MEAN
PT3000	F.F4.B
PT3001	F.F3.F4
PT3002	F.F4.F3
PT3003	F.T.F4
PT3004	F.B.F4

Figure 3.6. Représentation du modèle 3D du B11 depuis la station ST3000 (vue du dessus)



Comme le montre la figure 3.8, il y a deux faces (inférieure et supérieure) en trop au niveau du renfoncement. Cela résulte du fait qu'au lieu d'avoir considéré la forme planimétrique du bâtiment comme un « U », elle a été considérée comme un rectangle auquel on ajoute un enfoncement. Ce genre de détail devrait être corrigé à posteriori, une codification ne générant que rarement un dessin parfait (en pratique).

3.1.3. Test sur le B12

Ce bâtiment est levé selon un niveau de détail faible dans son ensemble et un niveau de détail moyen pour son entrée. Les méthodes de codification utilisées sont les suivantes :

- Codification par construction de plans : forme générale et une marche d'escalier
- Contour de face : face latérale située en dessous de la face de base du bâtiment (à cause du dénivelé important d'un côté à l'autre du bâtiment)
- Répétition d'objet : les autres marches de l'escalier
- Enfoncement : porte d'entrée
- Proéminence : palier de l'escalier et construction en béton soutenant l'escalier

Figure 3.7. Position des stations pour le levé du B12

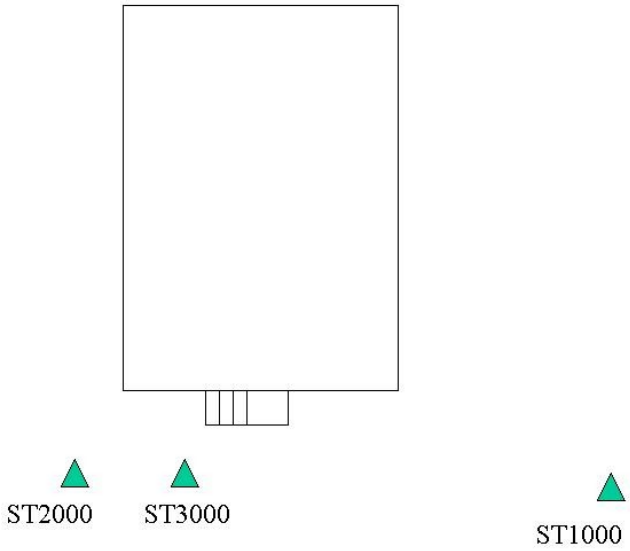


Figure 3.8. Représentation des points levés depuis la station ST1000

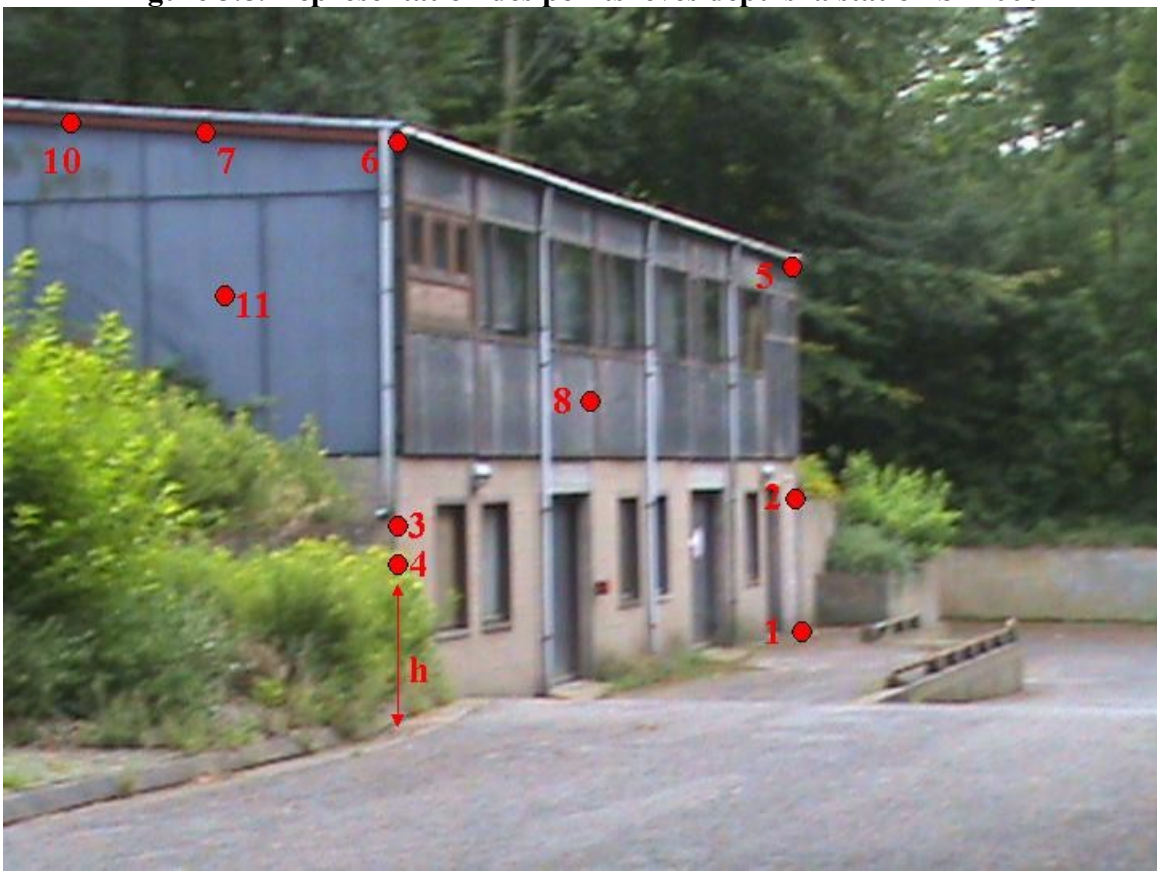


Tableau 3.4. Mesures et codes depuis la station ST1000

Mesure	Code
/	OBJ.MEAN
/	C.B.F1.F2.F3.F4
/	C.F1.B.F2.T2.T1.F4
/	C.F2.B.F1.T2.F3
/	C.F3.B.F2.T2.T1.F4
/	C.F4.B.F1.T1.F3
/	C.T1.F3.F4.F1.T2
/	C.T2.F1.F2.F3.T1
PT1001	CT.X
PT1002	CT.X/F.F2.F3.B
PT1003	CT.X/F.F2.F1.B
PT1004	CT.X
PT1005	F.T2.F2.F3
PT1006	F.T2.F1.F2
PT1007	F.T2.F1
PT1008	F.F2
PT1010	F.F1.T2
PT1011	F.F1

Rappelons que bien que ce ne soit pas le cas ici, la connectivité (code « C ») peut toujours être indiquée de façon implicite si elle a été encodée au préalable par l'utilisateur.

Figure 3.9. Représentation des points levés depuis la station ST2000

Tableau 3.5. Mesures et codes depuis la station ST2000

Mesure	Code
PT2001	F.F1.F4.B
PT2002	F.F1.F4.T1
PT2003	F.F4.B
PT2004	F.F4.F3
PT2005	F.F4.F3
PT2006	F.T1.F4
PT2007	F.T1.F1
PT2008	F.F4
PT2009	F.T1.F1

Figure 3.10. Représentation des points levés depuis la station ST3000

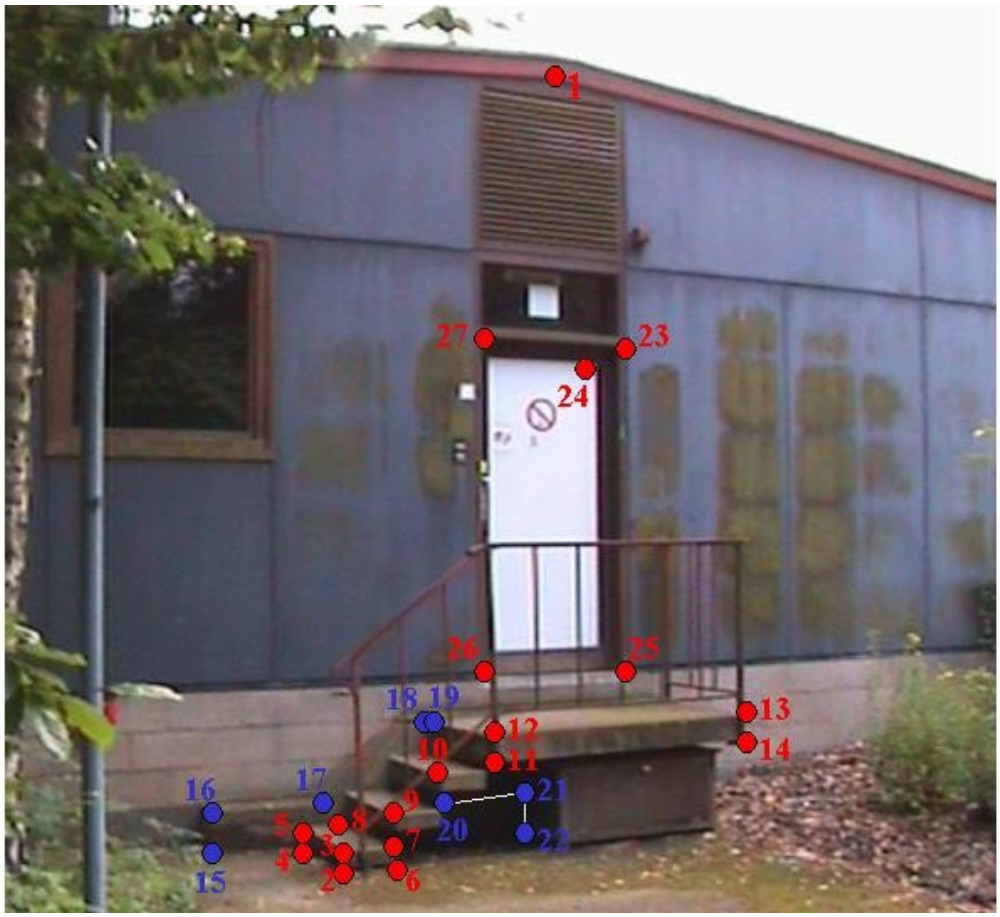


Tableau 3.6. Mesures et codes depuis la station ST3000

Mesure	Code
PT3001	F.T1.T2.F1
/	OBJ.MARCHE
/	C.B.F1.F2.F3.F4
/	C.T.F1.F2.F3.F4
/	C.F1.B.T.F2.F4
/	C.F2.F1.F3.B.T
/	C.F3.F2.F4.B.T
/	C.F4.F3.F1.B.T
PT3002	F.F1.B.F4/INSM
PT3003	F.F4.F1.T
PT3004	F.F4.F3.B
PT3005	F.F3.F4.T
PT3006	F.F1.B.F2
PT3007	F.F1.F2.T
PT3008	F.T.F2.F3
OBJ.ESCA1;	/
PT3009	INS.MARCHE.REP.1
OBJ.ESCA2;	/
PT3010	INS.MARCHE.REP.1
OBJ.MEAN;	/
PT3011	PRO.BLOC.F1
PT3012	BLOC.1
PT3013	BLOC.1
PT3014	BLOC.3
PT3015	PRO.BLOC2.F1
PT3016	BLOC2.1
PT3017	BLOC2.1
PT3018	BLOC2.1
PT3019	BLOC2.1
PT3020	BLOC2.1
PT3021	BLOC2.1
PT3022	BLOC2.3
PT3023	ENF.PORTE.F1
PT3024	PORTE.PAR
PT3025	PORTE.1
PT3026	PORTE.1
PT3027	PORTE.3

Figure 3.11. Représentation du modèle 3D du B12 depuis la station ST1000 (vue du dessus)

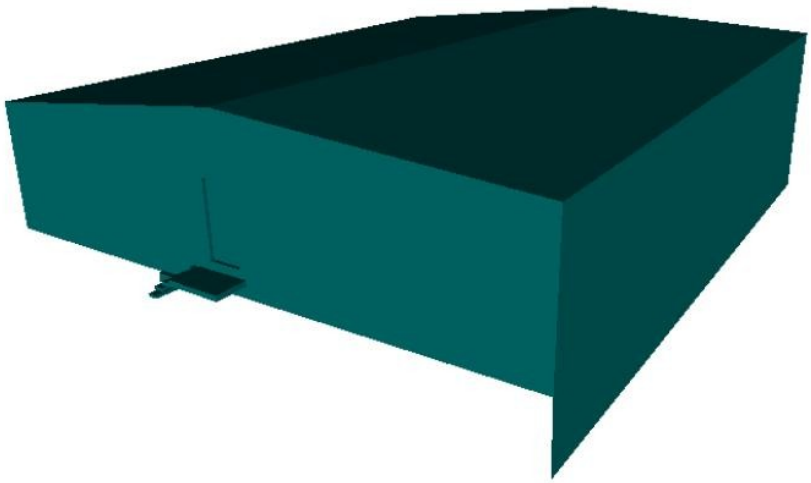
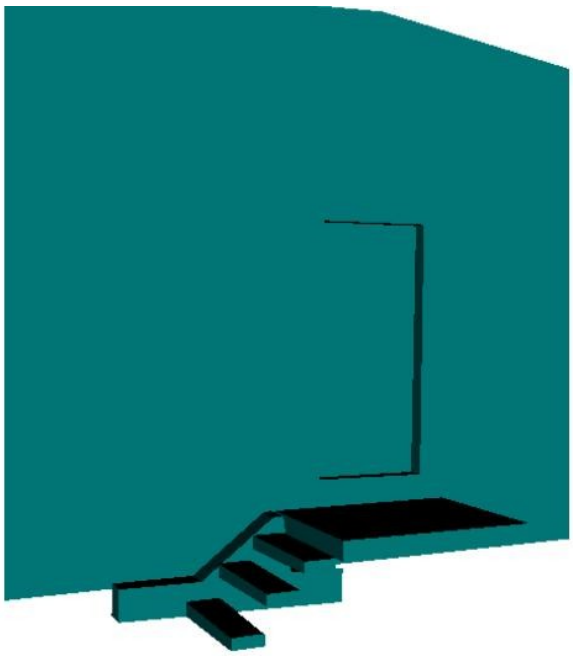


Figure 3.12. Représentation du modèle 3D du B12 depuis la station ST3000



Comme nous pouvons le voir sur la figure 3.14, la base du bâtiment est située au dessus de la première marche de l'escalier. Cela résulte du fait que la dénivellation du terrain est si importante que la base du bâtiment ne peut être considérée comme un plan. Dans ce cas, il est nécessaire de prolonger le bâtiment vers le bas jusqu'à l'intersection avec un MNT, ou d'envisager une autre technique de codification, par exemple : une liaison par plan par la base du bâtiment avec une hauteur de coordonnées z constante pour le corps du bâtiment et une codification par construction de plans pour la partie supérieure (en forme de prisme).

3.2. Critique et comparaison des différents codes

Nous allons maintenant discuter de l'efficacité des différentes méthodes de codification au moyen de sept facteurs :

- Liberté dans le choix des points
- Nombre de mesures
- Complexité et risque d'erreur
- Temps d'encodage
- Domaine d'application
- Précision du dessin
- Respect de la topologie

3.2.1. Liberté dans le choix des points

Une codification efficace doit pouvoir laisser à l'utilisateur une certaine liberté dans le choix des points qu'il va mesurer. En effet, il arrive bien souvent que des points sur le terrain soient inaccessibles (spécialement en trois dimensions) depuis une station.

La méthode de base par construction de plans laisse une grande liberté dans le choix des points à mesurer : la seule condition à remplir est de lever au moins trois points non-alignés par face. La répétition d'objets laisse également le choix à l'utilisateur par rapport à ses points d'insertion et points d'orientation.

Les méthodes utilisant des liaisons (liaison par face, liaison par parallélépipède, proéminence et enfoncement, contour de face) obligent l'utilisateur à mesurer des points bien particuliers, ce qui peut poser des problèmes par rapport aux points inaccessibles.

3.2.2. Nombre de mesures

Plus une codification nécessite un grand nombre de mesures et moins elle est efficace, l'idéal étant de mesurer un minimum de points pour lever un objet.

Le code nécessitant le moins de mesures est la répétition d'objet. Une copie d'objet peut être levée avec une ou trois mesures, peu importe sa complexité. Le nombre de mesures requises par les codifications par liaison tend vers le minimum nécessaire pour reproduire les formes géométriques qu'elles représentent. Par exemple, un parallélépipède rectangle levé au moyen d'une liaison par parallélépipède nécessite 4 mesures (nombre de points minimum pour dessiner la forme). Remarquons que lorsqu'un objet entier est mesuré au moyen de la méthode par contour de face, le nombre de mesure devient vite très grand (présence de mesures redondantes).

La codification par construction de plans nécessite toujours plus de mesures que le minimum requis. Plus un objet est quelconque et plus le nombre de mesures de la codification par construction de plans tend vers le minimum.

3.2.3. Complexité et risque d'erreur

Plus un code est complexe et moins il est efficace. La complexité se retrouve aussi bien dans la méthode de levé (quels points choisir et dans quel ordre) que dans le code lui-même (spécialement dans les paramètres). Autrement dit, plus l'utilisateur doit réfléchir et mémoriser pendant son levé et plus le code est complexe. Non seulement un code complexe est long à appliquer mais il est surtout une importante source d'erreur.

Concernant les liaisons par plan et par parallélépipède, l'utilisateur doit juste retenir sa ligne et son sens de liaison. Une fois que les paramètres de hauteur et de largeur sont connus, ils ne doivent en principe plus intervenir dans le levé. Le contour de face est également assez simple, bien que l'utilisateur doive retenir le nom de la face levée. Les proéminences et enfoncements nécessitent non seulement la mémorisation du nom de la proéminence/enfoncement en cours, mais également la mémorisation préalable du nom de la face support : elles sont donc de complexité moyenne.

Le code le plus complexe est la codification par construction de plans. Le nom de chaque face d'un objet doit être retenu et l'utilisateur doit bien veiller à avoir au moins levé trois points par face. De plus, l'écriture d'une connectivité de manière explicite peut s'avérer assez complexe sur le terrain. La répétition d'objet peut également s'avérer assez complexe puisque entre un et trois points doivent être retenus.

3.2.4. Temps d'encodage

Plus un code prend du temps à être écrit et plus le levé est long, et donc inefficace. Le temps d'encodage est directement lié au nombre de paramètres du code.

Les liaisons par face et par parallélépipède et les contours de face nécessitent presque toujours un seul paramètre. Les proéminences et enfoncements nécessitent deux paramètres lors de leur première mesure. La répétition d'objet nécessite un ou deux paramètre(s) en fonction de la situation. La codification par construction de plans nécessite entre un et trois paramètres en fonction de la situation.

Notons que bien que nous n'en ayons pas encore parlé dans ce mémoire, un code identique à celui de la mesure précédente peut être géré par la répétition automatique : si l'algorithme ne voit pas de code pour une mesure, il lui associe le code de la mesure précédente. Autrement dit, les codifications favorisant des codes qui se répètent permettent un gain de temps considérable (essentiellement les codes basés sur des liaisons).

3.2.5. Domaine d'application

Plus un code peut être appliqué à un grand nombre d'objets et plus il est efficace.

Comme nous l'avons vu, la codification par construction de plans et le contour de face peuvent s'appliquer à la totalité des objets à faces planes (pour tant qu'ils soient levés entièrement).

Les autres méthodes ont un domaine d'application beaucoup plus restreint. La codification par face se limite aux objets avec deux faces leur servant de base, les proéminences et enfoncements également (en plus de la condition de perpendicularité par rapport à une face

support), et la répétition d'objets se limite aux objets identiques aux niveau de leur géométrie et de leurs dimensions. C'est pourquoi ces codifications ne sont que complémentaires.

3.2.6. Précision du dessin

Une codification entraîne la construction géométrique d'un grand nombre de points (nous ne parlons pas ici des points levés). L'efficacité d'une codification est également liée à la précision avec laquelle ces points sont construits.

La méthode la plus précise est la codification par construction de plans. Si nous levons uniquement trois points par face, plus ces points sont dispersés et moins l'erreur sur les points se fait ressentir dans le tracé de la face. De plus, si nous levons plus de trois points, l'ajustement par moindre carré au nuage de points permet d'accroître la précision du dessin. Mais le gain de précision par rapport aux autres méthodes prend sa source essentiellement dans le fait qu'elle ne nécessite pas forcément le levé de points très précis comme des coins. En effet, un coin n'est pas toujours parfaitement matérialisé sur le terrain et est donc une source d'erreur importante si une codification se base dessus pour la construction du dessin.

Les autres méthodes de codification impliquent la construction du dessin à partir de points très précis (comme des coins). L'erreur sur ces points se répercute donc sur une partie de l'objet, voir sur l'objet entier dans le cas de points paramétriques (cas de la liaison par face, la liaison par parallélépipède et l'enfoncement). La méthode par répétition d'objets est de loin la moins précise puisqu'une erreur sur un seul point d'un objet copié se répercute sur l'ensemble de l'objet. De plus, deux objets ne sont théoriquement jamais identiques : l'interprétation de la similitude entre objets joue énormément sur la précision du dessin.

3.2.7. Respect de la topologie

Le respect de la topologie n'est pas vraiment lié aux codes mais à la façon dont on les combine. Par exemple, si on utilise une liaison par face pour dessiner une cheminée sur un toit résultant d'une autre liaison par face, la cheminée ne sera pas reliée au toit d'un point de vue topologique. Ainsi, le meilleur moyen pour que deux objets levés avec deux codes différents soient reliés d'un point de vue topologique est d'utiliser des points communs entre les deux objets (et donc d'utiliser le séparateur de code « / »), mais ce n'est pas toujours possible. Deux codes favorisent particulièrement bien le respect de la topologie lorsqu'ils sont combinés : la proéminence et l'enfoncement. En effet, leur construction se base sur des faces d'objets déjà construits au préalable.

3.2.8. Comparaison des différents types de code

Nous allons maintenant procéder à une comparaison de nos différents types de code en fonction des facteurs d'efficacité précités.

Pour chaque codification, nous attribuons une cote entre 1 et 5 par facteur. Plus la cote est élevée et plus le facteur est favorable à la méthode. Nous calculons ensuite pour chaque facteur la moyenne pondérée des sept cotes attribuées.

La pondération est appliquée en fonction de l'importance relative du facteur. Ainsi, pour obtenir ces pondérations, nous avons demandé à quatre géomètres d'attribuer une cote entre 1 et 5 à chaque facteur en fonction de l'importance qu'il lui accordait (plus la cote est élevée et plus le facteur est important). La pondération est obtenue en effectuant la moyenne arithmétique des cotes données par les quatre géomètres. Les résultats sont mentionnés dans le tableau 3.7.

Tableau 3.7. Comparaison des différents types de code

	Liberté dans le choix des points	**Nombre de mesures**	**Complexité**	**Temps d'encodage**	**Domaine**	**Précision du dessin**	**Respect de la topologie**	**Moyenne pondérée**
Construction de plans	5	2	1	3	5	5	3	**2.93**
Contour de face	1	1	4	5	5	3	3	**2.90**
Liaison par face	2	4	5	5	3	2	3	**3.26**
Liaison par parallélépipède	2	4	4	5	2	2	3	**2.95**
Proéminence	1	5	3	4	3	3	5	**3.01**
Enfoncement	1	4	3	4	3	2	5	**2.72**
Répétition d'objet	4	5	2	3	1	1	3	**2.40**
Pondération	3	4	4	4	4	3	3	

Bien entendu, ces résultats ne peuvent pas être considérés comme vraiment fiables, étant donné le peu d'objectivité dans l'attribution des cotes aux types de code et le peu de données concernant les pondérations (4). Cela dit, précisons que les cotes fournies par les quatre géomètres semblaient assez concordantes. L'utilité de ces résultats est donc de nous donner une petite idée de l'efficacité de notre géocodification qui semble globalement moyenne.

4. Perspectives

Comme précisé dans le chapitre 1 « *introduction* », la géocodification développée dans ce mémoire ne tient compte ni des objets à faces non planes ni de l'aspect attributaire. Etant donné l'importance de ces deux points, ce chapitre a pour but de suggérer quelques pistes pour solutionner ces problématiques.

4.1. Objets à faces non-planes

Nous allons décrire une solution au problème de la géocodification d'objets à face non-planes au moyen de primitives géométriques. Celle-ci est directement inspirée de la construction d'un modèle 3D à partir d'un nuage de point segmenté (*cf.* 1.3.3.3).

Une quadrique peut être décrite par une équation mathématique. En levant un nombre minimum de points sur un objet ou une face pouvant être assimilée à une quadrique, nous sommes donc en mesure de retrouver son équation. Mais nous savons que l'équation d'une face seule ne suffit pas au tracé de cette dernière : nous avons également besoin des informations de connectivité afin de retrouver ses limites par intersection avec les autres faces. Cette codification de connectivité peut par exemple s'effectuer de façon assez similaire à celle de la proéminence et de l'enfoncement : on spécifie le nom de la face plane support de la face quadrique, qu'elle soit matérialisée sur le terrain ou non. Cette face support sert à délimiter les arêtes de la face quadrique par intersection.

Prenons l'exemple d'un renfoncement cylindrique. L'équation paramétrique d'un cylindre dont la directrice est l'axe Z s'écrit :

$x^2 + y^2 = r^2$

L'équation d'un cylindre quelconque s'écrit en appliquant une rotation (3 paramètres : κ, Ω, φ) et une translation (trois paramètres δx, δy, δz) de notre système de coordonnées via la formule matricielle que nous avons déjà vue :

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = [R] * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} \delta x \\ \delta y \\ \delta z \end{bmatrix}$$

$$[R] = [R\kappa] * [R\phi] * [R\Omega]$$

$$[R\kappa] = \begin{bmatrix} \cos\kappa & -\sin\kappa & 0 \\ \sin\kappa & \cos\kappa & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad [R\phi] = \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix}$$

$$[R\Omega] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\Omega & -\sin\Omega \\ 0 & \sin\Omega & \cos\Omega \end{bmatrix}$$

Ainsi, les coordonnées x', y' et z' d'un point transformé dépendent toutes les trois de cinq inconnues.

$x' = f(x, \kappa, \Omega, \varphi, \delta x)$
$y' = f(y, \kappa, \Omega, \varphi, \delta y)$
$z' = f(z, \kappa, \Omega, \varphi, \delta z)$

L'équation d'un cylindre quelconque s'écrit donc :

$f(x, \kappa, \Omega, \varphi, \delta x)^2 + f(y, \kappa, \Omega, \varphi, \delta y)^2 = r^2$

Cette équation dépend de 6 paramètres (κ, Ω, φ, δx, δy, r) qui peuvent donc être retrouvés par un système de 6 équations à 6 inconnues au moyen de 3 couples de coordonnées x, y (via trois points).

Un cylindre peut donc être géocodé au moyen de 3 points. Puisque cette quadrique s'étend à l'infini, deux des trois points doivent être levés sur les bases du cylindre et codés comme tels.

La géocodification se rapportant à notre exemple peut donc être construite de la manière suivante…

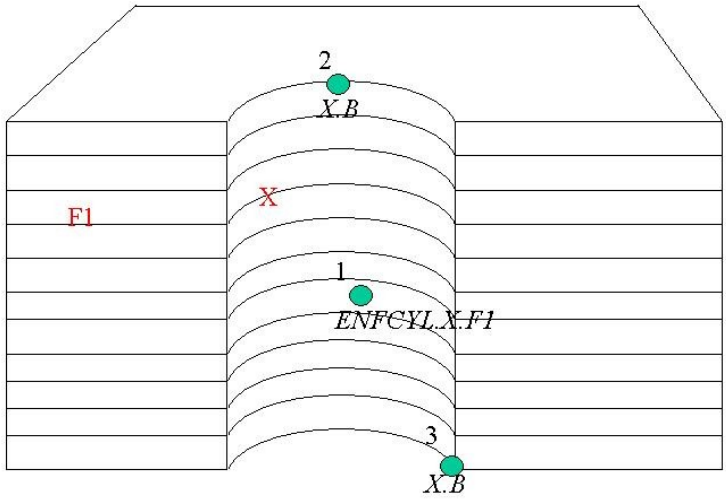
Pour un point quelconque de la face cylindrique, nous indiquons le code « enfoncement cylindrique » comprenant en premier paramètre le nom de l'enfoncement (x) et en second paramètre le nom de la face support (f).

ENFCYL.x.f

Pour les deux points matérialisant la base de la face cylindrique, nous indiquons le code « x » avec comme paramètre « B ».

x.B

Figure 4.1. Illustration d'un enfoncement cylindrique



A l'inverse d'un enfoncement, nous pouvons également imaginer un code « PROCYL » qui serait construit de la même façon mais pour des proéminences cylindriques.

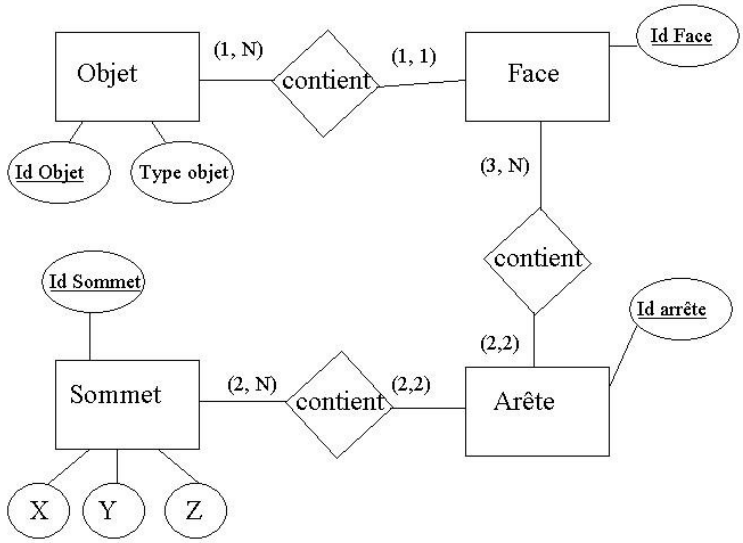
Remarquons qu'un objet entièrement constitué d'une quadrique (comme un pilier cylindrique par exemple) ne nécessite pas de face support. Dans ce cas, nous pourrions remplacer le code « ENFCYL.x.f » par « CYL.x ».

Certes, d'autres moyens peuvent être envisagés pour la géocodification de faces non-planes comme par exemple la méthode d'interpolation T.I.N. Mais puisqu'il s'agit d'une perspective, notre développement s'arrête là.

4.2. Aspect attributaire : implémentation d'un SIG 3D

Sans entrer dans le détail, nous allons maintenant démontrer que les informations introduites via notre géocodification peuvent également permettre l'implémentation partielle d'un SIG 3D. Nous appuierons certains de nos propos sur un diagramme entité-relation extrêmement simplifié pouvant représenter un SIG 3D (figure 4.2).

Figure 4.2. Diagramme entité-relation simplifié d'un SIG 3D



Nous avons vu que notre codification permettait d'identifier de façon unique toutes les faces des objets. Nous avons vu également que les objets doivent être nommés de façon unique. De plus, nous pourrions imaginer que l'algorithme identifie de façon unique chaque sommet et chaque arête construit via la géocodification. Ainsi, les attributs clés des quatre entités « Sommet », « Arête », « Face » et « Objet » peuvent être implémentés via la codification.

Lorsque nous introduisons le code « OBJ », nous pourrions introduire un paramètre supplémentaire (au moyen d'un « = ») en plus du nom de l'objet permettant de définir la catégorie à laquelle il appartient (par exemple pour un bâtiment : résidence, ferme, commerce, etc…). Ceci est illustré par l'attribut « Type objet » de l'entité « Objet » dans la figure 4.2.

Exemple : OBJ.MAISON1 = RESIDENCE

Avec un SIG faisant intervenir des classes d'objet (plus sophistiqué que celui de la figure 4.2), nous pourrions encoder les attributs des différents types d'objet sur le terrain. Par exemple, pour un objet de la classe « RESIDENCE », nous pourrions encoder son adresse :

OBJ.MAISON1 = RESIDENCE
ATTRIBUT.RUE = Place du Géloury
ATTRIBUT.NUMERO = 17
ATTRIBUT.CODE_SPOTAL = 4624
ATTRIBUT.LOCALITE = Romsée

Pour un objet de type « PORTE », nous pourrions encoder le bâtiment auquel elle appartient :

OBJ.PORTE1.PORTE
ATTRIBUT.BATIMENT = MAISON1

Remarquons que notre codification ne nous permet pas d'atteindre une topologie optimale : deux objets différents ne peuvent être reliés topologiquement. C'est pourquoi d'après la figure 4.2, une face ne peut appartenir qu'à un seul objet.

Nous voyons donc que la géocodification pourrait permettre le stockage de nombreuses informations dans un SIG-3D directement sur le terrain. Certes, encoder ces informations *in situ* ne représente pas un gain de temps particulier par rapport au post-traitement (ce qu'on ne fait plus en post-traitement, on le fait sur le terrain), mais certaines informations peuvent véritablement nécessiter une expertise sur place (le type de matériaux, l'état de dégradation du béton, la nuisance sonore, *etc*…), ce qui pourrait être combiné avec le levé.

5. Conclusion

L'objectif de ce mémoire étant le développement d'un prototype de géocodification, il a tout d'abord fallu s'assurer que cette méthode présente une quelconque utilité dans le cadre de la modélisation 3D. Dans ce but, sachant que les techniques d'acquisition de données de prédilection dans ce domaine sont la photogrammétrie et la lasergrammétrie, et qu'elles présentent l'avantage de saisir énormément de points en peu de temps sur le terrain, nous avons survolé les différentes étapes de leur déroulement respectif depuis la saisie des données sur le terrain jusqu'à la modélisation. Nous avons pu constater que ces techniques nécessitent toutes les deux un post-traitement très lourd (consolidation, nettoyage, orientations, extraction d'un nuage de points, segmentation et construction du modèle) alors que la géocodification permet la génération d'un modèle quasi instantané après le calcul des coordonnées des points avec des fichiers de données beaucoup plus légers. Nous en avons déduit que le levé à la station totale avec géocodification 3D trouvait effectivement sa place parmi les autres techniques d'acquisition de données non pas par concurrence, mais par complémentarité afin d'améliorer le rendement global de la conception de modèles 3D, son domaine d'application étant les niveaux de détail faibles et moyens.

La raison de ce travail étant justifiée, nous avons développé notre codification en utilisant l'interpréteur PERL pour le traitement des données et l'interface graphique VRML pour l'affichage du modèle. Dans un premier temps, nous avons mis au point un type de code unique permettant la saisie de la totalité des objets à face plane : la codification par construction de plans. Ainsi, nous nous sommes assurés que l'utilisateur ne se retrouve jamais bloqué sur le terrain devant un problème sans solution. Cependant, puisque la complexité de cette codification croît avec la complexité de l'objet levé (ce qui la rend de moins en moins efficace) il a fallu développer une série de types de code complémentaires à cette méthode de base. Ils sont au nombre de six : la liaison par plan, la liaison par parallélépipède, le contour de face, la proéminence, l'enfoncement et la répétition d'objet. Ces méthodes aux domaines d'application particuliers sont directement inspirées de la géocodification 2D et gardent une simplicité constante.

Au niveau de leur implémentation en PERL, les techniques de codification développées n'ont pas toujours été programmées dans leur totalité (le programme n'est qu'un prototype). Cela dit, au moins une explication théorique de la stratégie de programmation à adopter a été dressée pour les parties manquantes jugées importantes.

Une fois tous nos types de code validés dans notre programme PERL, nous avons pu tester notre géocodification 3D directement sur le terrain. Deux demi-journées de terrain ont permis de lever une partie du B11 et le B12 entier dans un niveau de détail entre faible et moyen. Un niveau de détail globalement moyen aurait pu être atteint si certains codes avaient été plus complets au niveau de leur implémentation ; je pense particulièrement au type de code « enfoncement » ne pouvant être utilisé qu'une fois par face d'objet.

Nous avons ensuite établi une liste de facteurs d'efficacité permettant de critiquer et comparer nos méthodes : le nombre de mesures, la liberté dans le choix des points, la complexité du code, le temps d'encodage, le respect de la topologie, la précision du dessin et le domaine d'application. La moyenne des cotes attribuées à chaque facteur pour chaque codification pondérée par l'importance relative des facteurs nous a révélé que nos différents types de code étaient tous d'une efficacité moyenne. Bien entendu, ces résultats statistiques sont loin d'être

totalement fiables étant donné le peu de données et leur manque d'objectivité, mais ont pu malgré tout donner une idée de la qualité du travail effectué.

Pour terminer, nous avons lancé quelques pistes sur le développement d'une géocodification 3D pouvant être appliquée aux objets à faces non-planes. De plus, puisque notre travail portait uniquement sur l'aspect géométrique de la géocodification, nous avons brièvement abordé son aspect attributaire en imaginant l'implémentation automatique d'un SIG-3D.

En conclusion, je me permettrai d'apporter ma réponse personnelle à la question : le développement d'une géocodification 3D simple et efficace est-il envisageable d'un point de vue pratique ? La géocodification développée dans ce mémoire n'est certainement pas celle qui se rapproche le plus de la méthode optimale puisqu'il ne s'agit que d'une première approche. Malgré cela, les résultats obtenus suite aux deux levés effectués sur le terrain semblent assez encourageants. Une fois la méthode assimilée, c'est-à-dire pour le second levé effectué (le B12), le temps passé à installer les stations et à lever les points pour obtenir un modèle 3D n'a pas dépassé trois heures. Sachant que mon expérience sur le terrain est encore relativement limitée, j'imagine qu'un géomètre expérimenté aurait pu effectuer ce levé dans les mêmes conditions en un temps nettement plus court. Le même levé (avec le même niveau de détail) effectué par lasergrammétrie ou photogrammétrie prendrait certainement plus de temps en considérant les diverses étapes de post-traitement induites pour obtenir un modèle 3D. C'est pourquoi il me semble que le développement d'une géocodification 3D simple et efficace est bel et bien envisageable. Pour confirmer cette réponse, il serait intéressant de réellement effectuer un test de comparaison entre photogrammétrie, lasergrammétrie et géocodification sur un levé en fonction du niveau de détail avec une géocodification plus complète du point de vue de l'implémentation.

Bibliographie

BASTIN F. 2003. *Mathématiques générales, partim B*, notes de cours, Université de Liège, Département de Mathématique, inédit, 247 p.

BASTIN F. 2004. *Compléments de mathématiques générales*, notes de cours, Université de Liège, Département de Mathématique, inédit, 73 p.

BASTIN T. 2004. *Méthodes numériques de la physique*, notes de cours, Université de Liège, Département de Physique, inédit, 143 p.

COLLIGNON A. 2006. *Notions de photogrammétrie*, notes de cours, Université de Liège, Département de Géomatique, inédit, 173 p.

DEVEAU M., PAPARODITIS N., PIERROT-DESEILLIGNY M., CHEN X., THIBAULT G. 2005. Strategy for the extraction of 3D architectural objects from laser and image data acquired from the same viewpoint, *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences,* **36**, Venise-Mestre.

DONNAY J.-P. 2004. *Analyse spatiale*, notes de cours, Université de Liège, Département de Géomatique, inédit, 235 p.

EL-HAKIM S., WHITING E, GONZO L., GIRARDI S., 3-D Reconstruction of complex architectures from multiple data, *3D Virtual Reconstruction and Visualization of Complex Architectures*, Venise-Mestre.

FastCAO. s.d. *Présentation AutoCAD TopoFast 1.05*, (http://www.fastcao.com/?n=TopoFast), consultation le 22 juillet 2008.

GEOMEDIA. (éditeur) 2001. *Covadis Topo 2D version 2000-2. La nouvelle géocodification*, Brest.

KOLBE T. H. s.d. *What is CityGML ?*, CityGML (http://www.citygml.org/1533/), consultation le 2 juillet 2008.

LAURINI R. & SERVIGNE S. 2008. Panorama des potentialités SIG en 3 dimensions : vers des modèles virtuels 3D de villes, *Revue XYZ*, **114**, pp. 22-26.

LEICA. s.d. *Leica HDS 6000* (http://www.leica-geosystems.com/corporate/en/ndef/lgs_64228.htm), consultation le 25 aout 2008.

LENOIR J. 2003. *Cours de VRML*, notes de cours, TELECOM Lille1, inédit, 90 p.

PENARD L., PAPARODITIS N. & PIERROT-DESEILLIGNY M. 2006. Reconstruction 3D automatique de façades de bâtiments en multi-vues, *Reconnaissance des Formes et Intelligence Artificielle*, Tours.

SOFT STRADA s.d. *Strada Polaris* (http://www.softstrada.com/Pdf/Polaris-Topo-Dao.pdf), consultation le 22 juillet 2008.

TAILLANDIER F. 2005. Automatic Building Reconstruction from Cadastral Maps and Aerial Images, *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, **36**, Vienne.

TILL D. 2000. *Perl 5*, collection Le programmeur. Paris : CampusPress, 809 p.

WIKIPEDIA. s.d. *Photogrammétrie* (http://fr.wikipedia.org/wiki/Photogramm%C3%A9trie), consultation le 5 août 2008.

WILLEM B. 2007. *Analyse de données lasergrammétriques*, Mémoire de licence en Sciences Géographiques, Option géomatique et géométrologie, Université de Liège, Faculté des sciences, inédit, 98 p.

Liste des annexes

Annexe 1. Fichier « output.txt » de la station totale du levé du B11

50=JPK060808

50=MISE EN STATION
2=STS1000 3=1.5170 4=ST
50=OUVERTURE AZ CONNU
62=ST2000 6=1.6390 7=319.06569 8=99.76318 9=0.00

5=ST2000 6=1.6390 7=319.06569 8=99.76318 9=0.00
50=OffL0.00 OffT0.00 OffH0.00
4=OBJ.MEAN
4=C.BT.4
5=PT1000 6=0.0000 7=372.07347 8=94.79423 9=35.1432
50=OffL0.00 OffT0.00 OffH0.00
4=F.F3.F2.T
5=PT1001 6=0.0000 7=371.34588 8=95.46323 9=34.1857
50=OffL0.00 OffT0.00 OffH0.00
4=F.F2
5=PT1002 6=0.0000 7=355.98104 8=97.12247 9=22.2141
50=OffL0.00 OffT0.00 OffH0.00
4=F.F2.F1
5=PT1003 6=0.0000 7=370.38937 8=94.44599 9=33.1525
50=OffL0.00 OffT0.00 OffH0.00
4=F.T.F2
5=PT1004 6=0.0000 7=351.64618 8=92.21411 9=23.6218
50=OffL0.00 OffT0.00 OffH0.00
4=F.T.F1
5=PT1005 6=0.0000 7=329.00089 8=95.01878 9=36.7502
50=OffL0.00 OffT0.00 OffH0.00
4=F.T.F1
5=PT1006 6=0.0000 7=327.06749 8=101.65422 9=38.7611
50=OffL0.00 OffT0.00 OffH0.00
4=F.F1.F4.B
5=PT1007 6=0.0000 7=372.06020 8=96.36442 9=35.0886
50=OffL0.00 OffT0.00 OffH0.00
4=F.F3.F2
5=ST2000 6=0.0000 7=319.06167 8=99.76294 9=32.5776
50=OffL0.00 OffT0.00 OffH0.00

50=MISE EN STATION
2=ST2000 3=1.6400 4=ST
50=OUVERTURE REF CONNUE
62=STS1000 6=1.5220 7=0.00000 8=100.24766 9=0.00

5=STS1000 6=1.5220 7=0.00000 8=100.24766 9=0.00
50=OffL0.00 OffT0.00 OffH0.00
5=PT2000 6=0.0000 7=340.95173 8=85.74953 9=12.4183
50=OffL0.00 OffT0.00 OffH0.00

4=ENF.CON.F1
5=PT2001 6=0.0000 7=313.12269 8=90.87766 9=19.3572
50=OffL0.00 OffT0.00 OffH0.00
4=CON.PAR
5=PT2002 6=0.0000 7=307.47213 8=77.33521 9=7.8704
50=OffL0.00 OffT0.00 OffH0.00
4=CON.1
5=PT2003 6=0.0000 7=307.52966 8=108.59505 9=7.4419
50=OffL0.00 OffT0.00 OffH0.00
4=CON.1
5=PT2004 6=1.5150 7=340.90899 8=98.64946 9=12.1034
50=OffL0.00 OffT0.00 OffH0.00
4=CON.3
5=PT2005 6=0.0000 7=275.96072 8=111.00430 9=6.6446
50=OffL0.00 OffT0.00 OffH0.00
4=F.F1.B
5=PT2006 6=0.0000 7=237.14479 8=92.57244 9=8.3216
50=OffL0.00 OffT0.00 OffH0.00
4=lipr.0
5=PT2007 6=0.0000 7=237.14464 8=108.80024 9=8.2636
50=OffL0.00 OffT0.00 OffH0.00
4=lipr.2
5=PT2008 6=0.0000 7=237.68703 8=92.72558 9=8.3368
50=OffL0.00 OffT0.00 OffH0.00
4=lipr.2
5=PT2009 6=0.0000 7=226.93229 8=93.63905 9=9.4633
50=OffL0.00 OffT0.00 OffH0.00
4=lipr.4
5=PT2010 6=0.0000 7=218.13154 8=93.05025 9=8.7259
50=OffL0.00 OffT0.00 OffH0.00
4=lipr.0
5=PT2011 6=0.0000 7=217.58294 8=93.11585 9=8.8319
50=OffL0.00 OffT0.00 OffH0.00
4=lipr.5
5=PT2012 6=0.3950 7=218.28264 8=105.73483 9=8.6918
50=OffL0.00 OffT0.00 OffH0.00
4=lipr.2
5=PT2013 6=0.3950 7=242.02100 8=95.36470 9=13.1818
50=OffL0.00 OffT0.00 OffH0.00
4=lipr.4
5=STS1000 6=1.5220 7=0.00422 8=100.24937 9=32.5774
50=OffL0.00 OffT0.00 OffH0.00
5=ST3000 6=1.5670 7=191.14176 8=97.90634 9=19.1774
50=OffL0.00 OffT0.00 OffH0.00
5=STS1000 6=1.5220 7=0.00632 8=100.24813 9=32.5775
50=OffL0.00 OffT0.00 OffH0.00

50=MISE EN STATION
2=ST3000 3=1.5690 4=ST
50=OUVERTURE REF CONNUE

62=ST2000 6=1.6060 7=0.00000 8=102.10567 9=0.00

5=ST2000 6=1.6060 7=0.00000 8=102.10567 9=0.00
50=OffL0.00 OffT0.00 OffH0.00
5=PT3000 6=1.5060 7=340.18714 8=101.41058 9=19.8884
50=OffL0.00 OffT0.00 OffH0.00
4=F.F2.B
5=PT3001 6=0.0000 7=331.78886 8=100.14224 9=22.7241
50=OffL0.00 OffT0.00 OffH0.00
4=F.F3.F2
5=PT3002 6=0.0000 7=331.79173 8=95.44949 9=22.7752
50=OffL0.00 OffT0.00 OffH0.00
4=F.F2.F3
5=PT3003 6=0.0000 7=333.51922 8=93.90500 9=22.1974
50=OffL0.00 OffT0.00 OffH0.00
4=F.T.F2
5=PT3004 6=1.2780 7=344.32028 8=101.63457 9=18.8407
50=OffL0.00 OffT0.00 OffH0.00
4=F.B.F2
5=STS1000 6=1.4780 7=5.59310 8=100.93473 9=51.6391
50=OffL0.00 OffT0.00 OffH0.00
5=ST2000 6=1.6060 7=0.00381 8=102.10335 9=19.1706
50=OffL0.00 OffT0.00 OffH0.00

Annexe 2. Calcul des coordonnées des points du levé du B11

Identifiant	Hauteur point	Angle horizontal (gons)	Angle horizonatl (rad)	Angle vertival (gon)	Angle vertical (rad)	distance oblique (m)	distance horizontale (m)	Ecart de fermeture horizontal (rad)	Ecart de fermeture vertical (rad)
STS1000	1.517	ST							
ST2000	1.639	319.066	5.012	99.763	1.567	0.000	0.000		
OBJ.MEAN									
C.BT.4									
PT1000	0.000	372.073	5.845	94.794	1.489	35.143	35.026		
PT1001	0.000	371.346	5.833	95.463	1.500	34.186	34.099		
PT1002	0.000	355.981	5.592	97.122	1.526	22.214	22.191		
PT1003	0.000	370.389	5.818	94.446	1.484	33.153	33.026		
PT1004	0.000	351.646	5.524	92.214	1.448	23.622	23.445		
PT1005	0.000	329.001	5.168	95.019	1.493	36.750	36.638		
PT1006	0.000	327.067	5.138	101.654	1.597	38.761	38.748		
PT1007	0.000	372.060	5.844	96.364	1.514	35.089	35.031		
ST2000	1.639	319.062	5.012	99.763	1.567	32.578	32.577	Ehz (rad)	Ev (rad)
								0.000	0.000
ST2000	1.640	ST							
STS1000	1.522	0.000	0.000	100.248	1.575	0.000	0.000		
PT2000	0.000	340.952	5.356	85.750	1.347	12.418	12.108		
PT2001	0.000	313.123	4.919	90.878	1.428	19.357	19.159		
PT2002	0.000	307.472	4.830	77.335	1.215	7.870	7.377		
PT2003	0.000	307.530	4.831	108.595	1.706	7.442	7.374		
PT2004	1.515	340.909	5.355	98.649	1.550	12.103	12.101		
PT2005	0.000	275.961	4.335	111.004	1.744	6.645	6.546		
PT2006	0.000	237.145	3.725	92.572	1.454	8.322	8.265		
PT2007	0.000	237.145	3.725	108.800	1.709	8.264	8.185		
PT2008	0.000	237.687	3.734	92.726	1.457	8.337	8.282		
PT2009	0.000	226.932	3.565	93.639	1.471	9.463	9.416		
PT2010	0.000	218.132	3.426	93.050	1.462	8.726	8.674		
PT2011	0.000	217.583	3.418	93.116	1.463	8.832	8.780		
PT2012	0.395	218.283	3.429	105.735	1.661	8.692	8.657		
PT2013	0.000	242.021	3.802	95.365	1.498	13.182	13.147		
ST3000	1.567	191.142	3.002	97.906	1.538	19.177	19.167		
STS1000	1.522	0.006	0.000	100.248	1.575	32.578	32.577	Ehz (rad)	Ev (rad)
								0.000	0.000
ST3000	1.569	ST							
ST2000	1.606	0.000	0.000	102.106	1.604	0.000	0.000		
PT3000	1.506	340.187	5.344	101.411	1.593	19.888	19.884		
PT3001	0.000	331.789	5.212	100.142	1.573	22.724	22.724		
PT3002	0.000	331.792	5.212	95.449	1.499	22.775	22.717		
PT3003	0.000	333.519	5.239	93.905	1.475	22.197	22.096		
PT3004	1.278	344.320	5.409	101.635	1.596	18.841	18.834		
STS1000	1.478	5.593	0.088	100.935	1.585	51.639	51.634		

ST2000	1.606	0.004	0.000	102.103	1.604	19.171	19.160	Ehz (rad) Ev (rad)
								-0.0001 0.0000

Identifiant	n	Angle horizontal compensé (rad)	Angle vertical compensé (rad)	Gisement local (rad)	Gisement global (rad)	X(m)	Y(m)	Z(m)
STS1000						100.000	100.000	100.000
ST2000					0.000	100.000	67.423	
OBJ.MEAN								
C.BT.4								
PT1000	1.000	5.845	1.489	0.833	0.833	74.091	76.431	104.388
PT1001	2.000	5.833	1.500	0.821	0.821	75.040	76.768	103.951
PT1002	3.000	5.592	1.526	0.580	0.580	87.841	81.436	102.521
PT1003	4.000	5.818	1.484	0.806	0.806	76.166	77.138	104.406
PT1004	5.000	5.524	1.448	0.512	0.512	88.518	79.559	104.399
PT1005	6.000	5.168	1.493	0.156	0.156	94.304	63.808	104.389
PT1006	7.000	5.138	1.597	0.126	0.126	95.141	61.558	100.510
PT1007	8.000	5.844	1.514	0.832	0.832	74.090	76.423	103.520
ST2000	9.000	5.012	1.567	0.000	0.000	100.000	67.423	99.999
ST2000						100.000	67.423	99.999
STS1000					3.142			
PT2000	1.000	5.356	1.347	5.356	2.214	90.311	74.685	104.396
PT2001	2.000	4.919	1.428	4.919	1.777	81.247	71.344	104.403
PT2002	3.000	4.830	1.215	4.830	1.688	92.674	68.286	104.382
PT2003	4.000	4.831	1.706	4.831	1.689	92.677	68.293	100.638
PT2004	5.000	5.355	1.550	5.355	2.213	90.313	74.674	100.381
PT2005	6.000	4.335	1.744	4.335	1.193	93.916	65.009	100.496
PT2006	7.000	3.725	1.454	3.725	0.583	95.447	60.525	102.608
PT2007	8.000	3.725	1.709	3.725	0.583	95.491	60.592	100.501
PT2008	9.000	3.734	1.457	3.734	0.592	95.379	60.549	102.590
PT2009	10.000	3.565	1.471	3.565	0.423	96.135	58.836	102.583
PT2010	11.000	3.426	1.462	3.426	0.285	97.563	59.098	102.590
PT2011	12.000	3.418	1.463	3.418	0.276	97.606	58.975	102.592
PT2012	13.000	3.429	1.661	3.429	0.287	97.549	59.120	100.462
PT2013	14.000	3.802	1.498	3.802	0.660	91.940	57.037	102.598
ST3000	15.000	3.002	1.538	3.002	6.144	102.660	48.441	100.703
STS1000	16.000	0.000	1.575	0.000	3.142	100.000	100.000	99.990
ST3000						102.660	48.441	100.703
ST2000					3.002			
PT3000	1.000	5.344	1.593	5.344	2.063	85.135	57.834	100.325
PT3001	2.000	5.212	1.573	5.212	1.931	81.393	56.448	102.221
PT3002	3.000	5.212	1.499	5.212	1.931	81.400	56.446	103.898
PT3003	4.000	5.239	1.475	5.239	1.958	82.201	56.785	104.393
PT3004	5.000	5.409	1.596	5.409	2.128	86.672	58.396	100.510
STS1000	6.000	0.088	1.586	0.088	3.090	100.006	100.006	100.034
ST2000	7.000	0.000	1.604	0.000	3.002	100.001	67.416	100.032

Annexe 3. Fichier « input.txt » du levé du B11 pour le programme PERL

OBJ.MEAN;
C.BT.4;
PT1000;74.091;76.431;104.388;F.F3.F2.T;
PT1001;75.040;76.768;103.951;F.F2;
PT1002;87.841;81.436;102.521;F.F2.F1;
PT1004;88.518;79.559;104.399;F.T.F1;
PT1005;94.304;63.808;104.389;F.T.F1;
PT1006;95.141;61.558;100.510;F.F1.F4.B;
PT1007;74.090;76.423;103.520;F.F3.F2;
PT2000;90.311;74.685;104.396;ENF.CON.F1;
PT2001;81.247;71.344;104.403;CON.PAR;
PT2002;92.674;68.286;104.382;CON.1;
PT2003;92.677;68.293;100.638;CON.1;
PT2004;90.313;74.674;100.381;CON.3;
PT2005;93.916;65.009;100.496;F.F1.B;
PT3000;85.135;57.834;100.325;F.F4.B;
PT3001;81.393;56.447;102.220;F.F3.F4;
PT3002;81.400;56.446;103.898;F.F4.F3;
PT3003;82.200;56.785;104.393;F.T.F4;
PT3004;86.671;58.396;100.509;F.B.F4;
OBJ.MUR1;
PT2006;95.447;60.525;102.608;lipr.0;
PT2007;95.491;60.592;100.501;lipr.2;
PT2008;95.379;60.549;102.590;lipr.5;
PT2009;96.135;58.836;102.583;lipr.4;
OBJ.MUR2;
PT2010;97.563;59.098;102.590;lipr.0;
PT2011;97.606;58.974;102.592;lipr.5;
PT2012;97.548;59.120;100.462;lipr.2;
PT2013;91.939;57.036;102.598;lipr.4;

Annexe 4. Fichier geobase.vrml du levé du B11

#VRML V2.0 utf8

```
NavigationInfo {
  type "EXAMINE"
}

Shape {
   appearance  Appearance {
    material     Material {
          diffuseColor 0 1 1
          specularColor .87 .25 .25
          ambientIntensity .157
          shininess .5
    }
   }
geometry         IndexedFaceSet {
solid FALSE
coord Coordinate {
point [
95.447 60.525 102.608,
95.491 60.592 100.501,
                                    95.423 60.616 100.483,
                                    95.379 60.549 102.590,
95.447 60.525 102.608,
                                    95.491 60.592 100.501,
                                    95.423 60.616 100.483,
                                    95.379 60.549 102.59,
                                    96.135 58.836 102.583,
                                    96.179 58.903 100.476,
                                    96.111 58.927 100.458,
                                    96.067 58.86 102.565,]
                  }
                  coordIndex [
0, 1, 2, 3, -1
                                    4, 5, 6, 7, -1,
                                          4, 8, 9, 5, -1,
                                          7, 11, 10, 6, -1,
                                          7, 4, 8, 11, -1,
                                          5, 6, 10, 9, -1,
                                          8, 9, 10, 11, -1
]
                              }
                              }
Shape {
   appearance  Appearance {
    material     Material {
          diffuseColor 0 1 1
          specularColor .87 .25 .25
          ambientIntensity .157
          shininess .5
    }
   }
geometry         IndexedFaceSet {
solid FALSE
coord Coordinate {
point [
97.563 59.098 102.590,
97.548 59.120 100.462,
                                    97.591 58.996 100.464,
                                    97.606 58.974 102.592,
97.563 59.098 102.590,
```

```
                                97.548 59.12 100.462,
                                97.591 58.996 100.464,
                                97.606 58.974 102.592,
                                91.939 57.036 102.598,
                                91.924 57.058 100.47,
                                91.967 56.934 100.472,
                                91.982 56.912 102.6,]
                }
                coordIndex [
0, 1, 2, 3, -1
                                4, 5, 6, 7, -1,
                                        4, 8, 9, 5, -1,
                                        7, 11, 10, 6, -1,
                                        7, 4, 8, 11, -1,
                                        5, 6, 10, 9, -1,
                                        8, 9, 10, 11, -1
]
                        }
                        }
        Shape {
   appearance  Appearance {
    material    Material {
        diffuseColor 0 .5 .5
        specularColor .87 .25 .25
        ambientIntensity .157
        shininess .5
    }
   }
        geometry        IndexedFaceSet {
        solid FALSE
        coord Coordinate {
                point [
90.311 74.685 104.396,
81.2436877629105 71.3542358934865 104.335683789448,
92.674 68.286 104.382,
83.5958382442453 64.9512504580677 104.321611617912,
92.677 68.293 100.638,
83.6158728104396 64.9645078935862 100.577724932691,
90.313 74.674 100.381,
81.2710174283983 71.352540430314 100.320852283639,
]
                }
                coordIndex [
0, 1, 3, 2, -1,
2, 3, 5, 4, -1,
0, 1, 7, 6, -1,
1, 3, 5, 7, -1,
4, 5, 7, 6, -1,
]
                                        }
                                        }
        Shape {
   appearance  Appearance {
    material    Material {
        diffuseColor 0 .5 .5
        specularColor .87 .25 .25
        ambientIntensity .157
        shininess .5
    }
   }
```

```
        geometry        IndexedFaceSet {
        solid FALSE
        coord Coordinate {
                point [
81.39957362785 56.4556845583148 104.375301170328,
81.3870089335271 56.4390850758 100.255703743024,
74.091 76.431 104.388,
74.1114141312536 76.3230636940326 100.171198366621,
87.8133374384432 81.4772751481301 104.40021787515,
87.8718463879141 81.3899742829737 100.425493282985,
95.1281523691371 61.5644424531146 104.387575609455,
95.1544696029479 61.5630130722944 100.510249038232,
90.311 74.685 104.396,
92.674 68.286 104.382,
92.677 68.293 100.638,
90.313 74.674 100.381,
]
                        }
                        coordIndex [
0, 1, 3, 2, -1,
2, 4, 5, 3, -1,
0, 6, 4, 2, -1,
8, 4, 6, 7, 5, 4, 8, 9, 10, 11, -1,
0, 6, 7, 1, -1,
1, 7, 5, 3, -1,
]
                                        }
                                        }
        Shape {
   appearance  Appearance {
    material    Material {
        diffuseColor 0 .5 .5
        specularColor .87 .25 .25
        ambientIntensity .157
        shininess .5
    }
   }
        geometry        IndexedFaceSet {
        solid FALSE
        coord Coordinate {
                point [
]
                }
                coordIndex [
]
                                        }
                                        }
```

Annexe 5. Fichier « output.txt » de la station totale du levé du B12

50=JPK210808

50=MISE EN STATION
2=ST1000 3=1.5130 4=ST
50=OUVERTURE AZ CONNU
62=ST3000 6=1.5190 7=56.76968 8=97.79925 9=31.7984

5=ST3000 6=1.5190 7=56.76968 8=97.79925 9=31.7984
50=OffL0.00 OffT0.00 OffH0.00
4=OBJ.MEAN
5=PT1001 6=0.0000 7=92.45200 8=102.79733 9=51.9251
50=OffL0.00 OffT0.00 OffH0.00
4=CT.X
5=PT1002 6=0.0000 7=92.46102 8=100.35699 9=51.8846
50=OffL0.00 OffT0.00 OffH0.00
4=CT.X/F.F2.F3.B
5=PT1003 6=0.0000 7=85.28273 8=100.55329 9=32.3364
50=OffL0.00 OffT0.00 OffH0.00
4=CT.X/F.F2.F1.B
5=PT1004 6=1.6730 7=85.28171 8=100.55299 9=32.3360
50=OffL0.00 OffT0.00 OffH0.00
4=CT.X
5=PT1005 6=0.0000 7=92.50800 8=95.99725 9=51.8126
50=OffL0.00 OffT0.00 OffH0.00
4=F.T2.F2.F3
5=PT1006 6=0.0000 7=85.33304 8=93.56195 9=32.4797
50=OffL0.00 OffT0.00 OffH0.00
4=F.T2.F1.F2
5=PT1007 6=0.0000 7=81.86447 8=93.36268 9=33.0684
50=OffL0.00 OffT0.00 OffH0.00
4=F.T2.F1
5=PT1008 6=0.0000 7=86.95607 8=98.14272 9=35.2195
50=OffL0.00 OffT0.00 OffH0.00
4=F.F2
5=PT1009 6=0.0000 7=85.91733 8=100.54110 9=33.4205
50=OffL0.00 OffT0.00 OffH0.00
4=F.B.F2
5=PT1010 6=0.0000 7=76.38438 8=93.06429 9=34.2854
50=OffL0.00 OffT0.00 OffH0.00
4=F.F1.T2
5=PT1011 6=0.0000 7=76.50641 8=95.62030 9=34.1239
50=OffL0.00 OffT0.00 OffH0.00
4=F.F1
5=ST2000 6=1.4620 7=50.56949 8=97.74603 9=36.0231
50=OffL0.00 OffT0.00 OffH0.00
5=ST3000 6=1.5190 7=56.76817 8=97.79918 9=31.7978
50=OffL0.00 OffT0.00 OffH0.00

50=MISE EN STATION
2=ST2000 3=1.4470 4=ST
50=OUVERTURE REF CONNUE

62=ST1000 6=1.5100 7=250.56949 8=102.26094 9=36.0235

5=ST1000 6=1.5100 7=250.56949 8=102.26094 9=36.0235
50=OffL0.00 OffT0.00 OffH0.00
5=PT2001 6=0.0000 7=118.89763 8=114.18444 9=7.2360
50=OffL0.00 OffT0.00 OffH0.00
4=F.F1.F4.B
5=PT2002 6=0.0000 7=118.71240 8=82.20161 9=7.3502
50=OffL0.00 OffT0.00 OffH0.00
4=F.F1.F4.T1
5=PT2003 6=0.0000 7=112.85840 8=108.63126 9=11.8438
50=OffL0.00 OffT0.00 OffH0.00
4=F.F4.B
5=PT2004 6=0.0000 7=107.76565 8=101.42345 9=26.8800
50=OffL0.00 OffT0.00 OffH0.00
4=F.F4.F3
5=PT2005 6=0.0000 7=107.76666 8=99.42191 9=26.8759
50=OffL0.00 OffT0.00 OffH0.00
4=F.F4.F3
5=PT2006 6=0.0000 7=113.12796 8=88.47541 9=11.1362
50=OffL0.00 OffT0.00 OffH0.00
4=F.T1.F4
5=PT2007 6=0.0000 7=137.84540 8=82.53893 9=8.2581
50=OffL0.00 OffT0.00 OffH0.00
4=F.T1.F1
5=PT2008 6=0.0000 7=113.03637 8=99.07292 9=11.1488
50=OffL0.00 OffT0.00 OffH0.00
4=F.F4
5=PT2009 6=0.0000 7=158.47933 8=84.68713 9=10.7849
50=OffL0.00 OffT0.00 OffH0.00
4=F.T1.F1
5=ST3000 6=1.5190 7=211.45237 8=102.09547 9=5.3611
50=OffL0.00 OffT0.00 OffH0.00
5=ST1000 6=1.5100 7=250.57051 8=102.26121 9=36.0235
50=OffL0.00 OffT0.00 OffH0.00

50=MISE EN STATION
2=ST3000 3=1.5300 4=ST
50=OUVERTURE REF CONNUE
62=ST2000 6=1.4490 7=11.45494 8=97.91448 9=5.3611

5=ST2000 6=1.4490 7=11.45494 8=97.91448 9=5.3611
50=OffL0.00 OffT0.00 OffH0.00
5=PT3001 6=0.0000 7=136.66339 8=79.34874 9=9.0792
50=OffL0.00 OffT0.00 OffH0.00
4=F.T1.T2.F1
4=OBJ.MARCHE
4=BT.4
5=PT3002 6=0.0000 7=127.58212 8=113.95929 9=7.2618
50=OffL0.00 OffT0.00 OffH0.00

4=F.F1.B.F4/INSM
5=PT3003 6=0.0000 7=127.57296 8=113.29220 9=7.2461
50=OffL0.00 OffT0.00 OffH0.00
4=F.F4.F1.T
5=PT3004 6=0.0000 7=125.54217 8=113.00477 9=7.8916
50=OffL0.00 OffT0.00 OffH0.00
4=F.F4.F3.B
5=PT3005 6=0.0000 7=125.54415 8=112.21590 9=7.8934
50=OffL0.00 OffT0.00 OffH0.00
4=F.F3.F4.T
5=PT3006 6=0.0000 7=129.54842 8=113.79040 9=7.3523
50=OffL0.00 OffT0.00 OffH0.00
4=F.F1.B.F2
5=PT3007 6=0.0000 7=129.54452 8=113.12961 9=7.3271
50=OffL0.00 OffT0.00 OffH0.00
4=F.F1.F2.T
5=PT3008 6=0.0000 7=127.38481 8=112.08255 9=7.9737
50=OffL0.00 OffT0.00 OffH0.00
4=F.T.F2.F3
4=OBJ.ESCA1
5=PT3009 6=0.0000 7=129.58620 8=111.70364 9=7.3036
50=OffL0.00 OffT0.00 OffH0.00
4=INS.MARCHE.REP.1
4=OBJ.ESCA2
5=PT3010 6=0.0000 7=131.70520 8=110.01837 9=7.3870
50=OffL0.00 OffT0.00 OffH0.00
4=INS.MARCHE.REP.1
4=OBJ.MEAN
5=PT3011 6=0.0000 7=133.92868 8=109.39044 9=7.4642
50=OffL0.00 OffT0.00 OffH0.00
4=PRO.BLOC.F1
5=PT3012 6=0.0000 7=133.89049 8=108.23006 9=7.4507
50=OffL0.00 OffT0.00 OffH0.00
4=BLOC.1
5=PT3013 6=0.0000 7=144.66833 8=107.35812 9=8.2489
50=OffL0.00 OffT0.00 OffH0.00
4=BLOC.1
5=PT3014 6=0.0000 7=144.67473 8=108.44169 9=8.2708
50=OffL0.00 OffT0.00 OffH0.00
4=BLOC.3
5=PT3015 6=0.0000 7=121.63874 8=113.17827 9=7.7686
50=OffL0.00 OffT0.00 OffH0.00
4=PRO.BLOC2.F1
5=PT3016 6=0.0000 7=121.64447 8=111.39084 9=7.7330
50=OffL0.00 OffT0.00 OffH0.00
4=BLOC2.1
5=PT3017 6=0.0000 7=126.37201 8=111.16195 9=7.9078
50=OffL0.00 OffT0.00 OffH0.00
4=BLOC2.1

5=PT3018 6=0.0000 7=130.49399 8=107.67633 9=8.0522
50=OffL0.00 OffT0.00 OffH0.00
4=BLOC2.1
5=PT3019 6=0.0000 7=131.09267 8=107.57289 9=8.0598
50=OffL0.00 OffT0.00 OffH0.00
4=BLOC2.1
5=PT3020 6=0.0000 7=131.69105 8=110.76276 9=8.1777
50=OffL0.00 OffT0.00 OffH0.00
4=BLOC2.1
5=PT3021 6=0.0000 7=134.75862 8=110.58092 9=8.3673
50=OffL0.00 OffT0.00 OffH0.00
4=BLOC2.1
5=PT3022 6=0.0000 7=134.75981 8=112.47968 9=8.4149
50=OffL0.00 OffT0.00 OffH0.00
4=BLOC2.3
5=PT3023 6=0.0000 7=139.40079 8=90.99211 9=8.9258
50=OffL0.00 OffT0.00 OffH0.00
4=ENF.PORTE.F1
5=PT3024 6=0.0000 7=139.05588 8=91.13979 9=9.0083
50=OffL0.00 OffT0.00 OffH0.00
4=PORTE.PAR
5=PT3025 6=0.0000 7=139.51441 8=105.42731 9=8.8738
50=OffL0.00 OffT0.00 OffH0.00
4=PORTE.1
5=PT3026 6=0.0000 7=133.46088 8=105.72466 9=8.4367
50=OffL0.00 OffT0.00 OffH0.00
4=PORTE.1
5=PT3027 6=0.0000 7=133.41837 8=87.21419 9=8.5630
50=OffL0.00 OffT0.00 OffH0.00
4=PORTE.3
5=ST1000 6=1.5100 7=256.78081 8=102.20972 9=31.7927
50=OffL0.00 OffT0.00 OffH0.00
5=ST2000 6=1.4490 7=11.45855 8=97.91493 9=5.3612
50=OffL0.00 OffT0.00 OffH0.00

Annexe 6. Calcul des coordonnées des points du levé du B12

Identifiant	Hauteur point (m)	Angle horizontal (gon)	Angle horizontal (rad)	Angle vertical (gons)	Angle vertical (rad)	Distance oblique (m)	Distance horizontale (m)	Gisement local (rad)	Gisement global (rad)
ST1000	1.5130								
ST3000	1.5190	56.7697	0.8917	97.7993	1.5362	31.7984	31.7794	0.0000	0.0000
OBJ.MEAN									
PT1001	0.0000	92.4520	1.4522	102.7973	1.6147	51.9251	51.8750	0.5605	0.5605
PT1002	0.0000	92.4610	1.4524	100.3570	1.5764	51.8846	51.8838	0.5606	0.5606
PT1003	0.0000	85.2827	1.3396	100.5533	1.5795	32.3364	32.3352	0.4479	0.4479
PT1004	1.6730	85.2817	1.3396	100.5530	1.5795	32.3360	32.3348	0.4479	0.4479
PT1005	0.0000	92.5080	1.4531	95.9973	1.5079	51.8126	51.7102	0.5614	0.5614
PT1006	0.0000	85.3330	1.3404	93.5620	1.4697	32.4797	32.3138	0.4487	0.4487
PT1007	0.0000	81.8645	1.2859	93.3627	1.4665	33.0684	32.8888	0.3942	0.3942
PT1008	0.0000	86.9561	1.3659	98.1427	1.5416	35.2195	35.2045	0.4742	0.4742
PT1009	0.0000	85.9173	1.3496	100.5411	1.5793	33.4205	33.4193	0.4579	0.4579
PT1010	0.0000	76.3844	1.1998	93.0643	1.4619	34.2854	34.0821	0.3081	0.3081
PT1011	0.0000	76.5064	1.2018	95.6203	1.5020	34.1239	34.0432	0.3100	0.3100
ST2000	1.4620	50.5695	0.7943	97.7460	1.5354	36.0231	36.0005	6.1858	6.1858
ST3000	1.5190	56.7682	0.8917	97.7992	1.5362	31.7978	31.7788	6.2832	6.2832
ST2000	1.4470								
ST1000	1.5100	250.5695	3.9359	102.2609	1.6063	36.0235	36.0008	0.0000	3.0442
PT2001	0.0000	118.8976	1.8676	114.1844	1.7936	7.2360	7.0571	4.2149	0.9759
PT2002	0.0000	118.7124	1.8647	82.2016	1.2912	7.3502	7.0648	4.2120	0.9730
PT2003	0.0000	112.8584	1.7728	108.6313	1.7064	11.8438	11.7351	4.1200	0.8810
PT2004	0.0000	107.7657	1.6928	101.4235	1.5932	26.8800	26.8733	4.0400	0.8010
PT2005	0.0000	107.7667	1.6928	99.4219	1.5617	26.8759	26.8748	4.0400	0.8011
PT2006	0.0000	113.1280	1.7770	88.4754	1.3898	11.1362	10.9542	4.1243	0.8853
PT2007	0.0000	137.8454	2.1653	82.5389	1.2965	8.2581	7.9494	4.5125	1.2735
PT2008	0.0000	113.0364	1.7756	99.0729	1.5562	11.1488	11.1476	4.1228	0.8838
PT2009	0.0000	158.4793	2.4894	84.6871	1.3303	10.7849	10.4744	4.8366	1.5977
ST3000	1.5190	211.4524	3.3215	102.0955	1.6037	5.3611	5.3582	5.6687	2.4298
ST1000	1.5100	250.5705	3.9360	102.2612	1.6063	36.0235	36.0008	0.0000	3.0442
ST3000	1.5300								
ST2000	1.4490	11.4549	0.1799	97.9145	1.5380	5.3611	5.3582	0.0000	5.5713
PT3001	0.0000	136.6634	2.1467	79.3487	1.2464	9.0792	8.6057	1.9668	1.2549
OBJ.MARCHE									
C.BT.4									
PT3002	0.0000	127.5821	2.0041	113.9593	1.7901	7.2618	7.0879	1.8241	1.1123
PT3003	0.0000	127.5730	2.0039	113.2922	1.7796	7.2461	7.0887	1.8240	1.1121
PT3004	0.0000	125.5422	1.9720	113.0048	1.7751	7.8916	7.7275	1.7921	1.0802
PT3005	0.0000	125.5442	1.9720	112.2159	1.7627	7.8934	7.7485	1.7921	1.0803
PT3006	0.0000	129.5484	2.0349	113.7904	1.7874	7.3523	7.1805	1.8550	1.1432
PT3007	0.0000	129.5445	2.0349	113.1296	1.7770	7.3271	7.1718	1.8549	1.1431
PT3008	0.0000	127.3848	2.0010	112.0826	1.7606	7.9737	7.8305	1.8210	1.1092
OBJ.ESCA1									
PT3009	0.0000	129.5862	2.0355	111.7036	1.7546	7.3036	7.1805	1.8556	1.1438
OBJ.ESCA2									
PT3010	0.0000	131.7052	2.0688	110.0184	1.7282	7.3870	7.2957	1.8889	1.1770
OBJ.MEAN									
PT3011	0.0000	133.9287	2.1037	109.3904	1.7183	7.4642	7.3831	1.9238	1.2120

PT3012	0.0000	133.8905	2.1031	108.2301	1.7001	7.4507	7.3885	1.9232	1.2114
PT3013	0.0000	144.6683	2.2724	107.3581	1.6864	8.2489	8.1939	2.0925	1.3807
PT3014	0.0000	144.6747	2.2725	108.4417	1.7034	8.2708	8.1982	2.0926	1.3808
PT3015	0.0000	121.6387	1.9107	113.1783	1.7778	7.7686	7.6027	1.7308	1.0189
PT3016	0.0000	121.6445	1.9108	111.3908	1.7497	7.7330	7.6095	1.7309	1.0190
PT3017	0.0000	126.3720	1.9850	111.1620	1.7461	7.9078	7.7866	1.8051	1.0933
PT3018	0.0000	130.4940	2.0498	107.6763	1.6914	8.0522	7.9937	1.8699	1.1580
PT3019	0.0000	131.0927	2.0592	107.5729	1.6898	8.0598	8.0028	1.8793	1.1674
PT3020	0.0000	131.6911	2.0686	110.7628	1.7399	8.1777	8.0611	1.8887	1.1768
PT3021	0.0000	134.7586	2.1168	110.5809	1.7370	8.3673	8.2520	1.9368	1.2250
PT3022	0.0000	134.7598	2.1168	112.4797	1.7668	8.4149	8.2537	1.9369	1.2250
PT3023	0.0000	139.4008	2.1897	90.9921	1.4293	8.9258	8.8366	2.0098	1.2979
PT3024	0.0000	139.0559	2.1843	91.1398	1.4316	9.0083	8.9212	2.0044	1.2925
PT3025	0.0000	139.5144	2.1915	105.4273	1.6560	8.8738	8.8416	2.0116	1.2997
PT3026	0.0000	133.4609	2.0964	105.7247	1.6607	8.4367	8.4026	1.9165	1.2046
PT3027	0.0000	133.4184	2.0957	87.2142	1.3700	8.5630	8.3909	1.9158	1.2040
ST1000	1.5100	256.7808	4.0335	102.2097	1.6055	31.7927	31.7736	3.8536	3.1417
ST2000	1.4490	11.4586	0.1800	97.9149	1.5380	5.3612	5.3583	0.0001	5.5714

Identifiant	X (m)	Y (m)	Z (m)	Code
ST1000	100.0000	100.0000	100.0000	
ST3000	100.0000	68.2206	101.0930	
OBJ.MEAN				
PT1001	72.4229	56.0623	99.2321	CT.X
PT1002	72.4120	56.0588	101.2221	CT.X/F.F2.F3.B
PT1003	85.9970	70.8542	101.2320	CT.X/F.F2.F1.B
PT1004	85.9977	70.8543	99.5591	CT.X
PT1005	72.4720	56.2261	104.7686	F.T2.F2.F3
PT1006	85.9833	70.8845	104.7920	F.T2.F1.F2
PT1007	87.3688	69.6334	104.9544	F.T2.F1
PT1008	83.9257	68.6795	102.5404	F.F2
PT1009	85.2280	70.0227	101.2289	F.B.F2
PT1010	89.6644	67.5228	105.2409	F.F1.T2
PT1011	89.6141	67.5798	103.8587	F.F1
ST2000	103.5006	64.1701	101.3261	
ST3000	100.0008	68.2212	101.0930	
ST2000	103.5006	64.1701	101.3261	
ST1000	100.0000	100.0003	99.9840	
PT2001	97.6559	60.2151	101.1742	F.F1.F4.B
PT2002	97.6610	60.1938	104.8014	F.F1.F4.T1
PT2003	94.4482	56.7024	101.1723	F.F4.B
PT2004	84.2034	45.4674	102.1722	F.F4.F3
PT2005	84.2020	45.4667	103.0172	F.F4.F3
PT2006	95.0211	57.2352	104.7781	F.T1.F4
PT2007	95.8999	61.8417	105.0099	F.T1.F1
PT2008	94.8816	57.1004	102.9355	F.F4
PT2009	93.0300	64.4513	105.3423	F.T1.F1
ST3000	100.0005	68.2271	101.0777	
ST1000	100.0005	100.0003	99.9839	
ST3000	100.0005	68.2271	101.0777	
ST2000	103.5007	64.1701	101.3343	

PT3001 91.8206 65.5538 105.5015 F.T1.T2.F1
OBJ.MARCHE
C.BT.4
PT3002 93.6447 65.0898 101.0281 F.F1.B.F4/INSM
PT3003 93.6444 65.0886 101.1057 F.F4.F1.T
PT3004 93.1843 64.5865 101.0068 F.F4.F3.B
PT3005 93.1657 64.5768 101.1023 F.F3.F4.T
PT3006 93.4666 65.2492 101.0275 F.F1.B.F2
PT3007 93.4747 65.2524 101.1073 F.F1.F2.T
PT3008 92.9896 64.7394 101.1034 F.T.F2.F3
OBJ.ESCA1
PT3009 93.4648 65.2531 101.2726 INS.MARCHE.REP.1
OBJ.ESCA2
PT3010 93.2631 65.4280 101.4500 INS.MARCHE.REP.1
OBJ.MEAN
PT3011 93.0876 65.6343 101.5107 PRO.BLOC.F1
PT3012 93.0841 65.6283 101.6472 BLOC.1
PT3013 91.9543 66.6786 101.6564 BLOC.1
PT3014 91.9499 66.6786 101.5142 BLOC.3
PT3015 93.5264 64.2411 101.0110 PRO.BLOC2.F1
PT3016 93.5203 64.2381 101.2314 BLOC2.1
PT3017 93.0850 64.6485 101.2283 BLOC2.1
PT3018 92.6782 65.0204 101.6391 BLOC2.1
PT3019 92.6400 65.0858 101.6512 BLOC2.1
PT3020 92.5569 65.1327 101.2318 BLOC2.1
PT3021 92.2370 65.4302 101.2234 BLOC2.1
PT3022 92.2353 65.4297 100.9687 BLOC2.3
PT3023 91.4908 65.8457 103.8665 ENF.PORTE.F1
PT3024 91.4225 65.7763 103.8574 PORTE.PAR
PT3025 91.4818 65.8595 101.8521 PORTE.1
PT3026 92.1549 65.2186 101.8501 PORTE.1
PT3027 92.1679 65.2175 104.3160 PORTE.3
ST1000 100.0048 100.0006 99.9944
ST2000 103.5005 64.1698 101.3343

Annexe 7. Fichier « input.txt » du levé du B12 pour le programme PERL

OBJ.MEAN;
C.B.F1.F2.F3.F4;
C.F1.B.F2.T2.T1.F4;
C.F2.B.F1.T2.F3;
C.F3.B.F2.T2.T1.F4;

C.F4.B.F1.T1.F3;
C.T1.F3.F4.F1.T2;
C.T2.F1.F2.F3.T1;
PT1001;72.42290461;56.06234602;99.23212671;CT.X;
PT1002;72.41199848;56.05879792;101.2220542;CT.X/F.F2.F3.B;
PT1003;85.99701283;70.8541575;101.231966;CT.X/F.F2.F1.B;
PT1004;85.99765245;70.85429249;99.55912183;CT.X;
PT1005;72.47197685;56.22609568;104.7685738;F.T2.F2.F3;
PT1006;85.98327652;70.88453468;104.7920325;F.T2.F1.F2;
PT1007;87.36875818;69.63344967;104.9544284;F.T2.F1;
PT1008;83.92571637;68.6794774;102.540351;F.F2;
PT1010;89.66441277;67.52281897;105.2408683;F.F1.T2;
PT1011;89.61406162;67.57978518;103.8587421;F.F1;
PT2001;97.6558592;60.21512123;101.1742045;F.F1.F4.B;
PT2002;97.66104439;60.19381047;104.8014186;F.F1.F4.T1;
PT2003;94.44816117;56.702445;101.1722798;F.F4.B;
PT2004;84.20340981;45.46739644;102.1721681;F.F4.F3;
PT2005;84.20202808;45.46665105;103.0171878;F.F4.F3;
PT2006;95.02109766;57.23520377;104.7781108;F.T1.F4;
PT2007;95.89985997;61.84166665;105.0098631;F.T1.F1;
PT2008;94.88155681;57.10036365;102.9354905;F.F4;
PT2009;93.03000006;64.45133597;105.3423342;F.T1.F1;
PT3001;91.8205727;65.55379465;105.5015225;F.T1.T2.F1;
OBJ.MARCHE;
C.B.F1.F2.F3.F4;
C.T.F1.F2.F3.F4;
C.F1.B.T.F2.F4;
C.F2.F1.F3.B.T;
C.F3.F2.F4.B.T;
C.F4.F3.F1.B.T;
PT3002;93.64469111;65.08983591;101.0281293;F.F1.B.F4/INSM;
PT3003;93.64442223;65.08856574;101.1057406;F.F4.F1.T;
PT3004;93.18430189;64.58649456;101.0068144;F.F4.F3.B;
PT3005;93.16565544;64.57680854;101.1023474;F.F3.F4.T;
PT3006;93.46662257;65.24922796;101.0274874;F.F1.B.F2;
PT3007;93.47467666;65.25241583;101.1072626;F.F1.F2.T;
PT3008;92.98957395;64.73940272;101.1034322;F.T.F2.F3;
OBJ.ESCA1;
PT3009;93.46480927;65.25308449;101.2725638;INS.MARCHE.REP.1;
OBJ.ESCA2;
PT3010;93.26307887;65.42803849;101.450023;INS.MARCHE.REP.1;
OBJ.MEAN;
PT3011;93.08758982;65.63430955;101.5106934;PRO.BLOC.F1;
PT3012;93.0841096;65.62827047;101.6471826;BLOC.1;
PT3013;91.95429102;66.67857394;101.6564143;BLOC.1;
PT3014;91.94988337;66.67856498;101.5141982;BLOC.3;
PT3015;93.5264369;64.24107576;101.0110404;PRO.BLOC2.F1;
PT3016;93.52029119;64.23809623;101.231439;BLOC2.1;
PT3017;93.08498347;64.64851327;101.228316;BLOC2.1;
PT3018;92.67816119;65.02035697;101.6391304;BLOC2.1;

PT3019;92.63995077;65.08578165;101.6512182;BLOC2.1;
PT3020;92.5569453;65.13273649;101.2317556;BLOC2.1;
PT3021;92.2369547;65.43016078;101.2234189;BLOC2.1;
PT3022;92.23526823;65.4297172;100.9686785;BLOC2.3;
PT3023;91.49084603;65.84565426;103.866461;ENF.PORTE.F1;
PT3024;91.42252932;65.77634549;103.8574039;PORTE.PAR;
PT3025;91.4818157;65.85951327;101.8521166;PORTE.1;
PT3026;92.15494759;65.21856654;101.8500804;PORTE.1;
PT3027;92.16791079;65.21753672;103.835956;PORTE.3;

Annexe 8. Fichier « geobase.vrml » du levé du B12

```
#VRML V2.0 utf8

NavigationInfo {
  type "EXAMINE"
}
```

```
        Shape {
   appearance  Appearance {
    material    Material {
        diffuseColor 0 .5 .5
        specularColor .87 .25 .25
        ambientIntensity .157
        shininess .5
    }
   }
        geometry        IndexedFaceSet {
        solid FALSE
        coord Coordinate {
                point [
72.42290461 56.06234602 99.23212671
72.41199848 56.05879792 101.2220542
85.99701283 70.8541575 101.231966
85.99765245 70.85429249 99.55912183
]
                }
                coordIndex [
                0, 1, 2, 3, -1,
                ]
                                }
                                }

        Shape {
   appearance  Appearance {
    material    Material {
        diffuseColor 0 .5 .5
        specularColor .87 .25 .25
        ambientIntensity .157
        shininess .5
    }
   }
        geometry        IndexedFaceSet {
        solid FALSE
        coord Coordinate {
                point [
91.49084603 65.84565426 103.866461,
91.4256854942601 65.7735411503794 103.866824905447,
91.4818157 65.85951327 101.8521166,
91.4087637012421 65.7786666849526 101.852524577313,
92.15494759 65.21856654 101.8500804,
92.0981629903838 65.1557230717813 101.850397527919,
92.16791079 65.21753672 103.835956,
92.1107967804543 65.1543286940919 103.836274967592,
]
                }
                coordIndex [
0, 1, 3, 2, -1,
```

```
2, 3, 5, 4, -1,
0, 1, 7, 6, -1,
1, 3, 5, 7, -1,
4, 5, 7, 6, -1,
]
                                        }
                                        }
        Shape {
   appearance  Appearance {
    material    Material {
        diffuseColor 0 .5 .5
        specularColor .87 .25 .25
        ambientIntensity .157
        shininess .5
    }
   }
        geometry        IndexedFaceSet {
        solid FALSE
        coord Coordinate {
                point [
85.99701283 70.8541575 101.231966,
72.41199848 56.05879792 101.2220542,
85.9546966427124 70.9103422075151 104.788682578433,
72.47197685 56.22609568 104.7685738,
91.8790048412037 65.5607129805555 105.483086112992,
78.1058263860848 51.1783477330501 105.426631678409,
97.7078699873232 60.272076988514 101.174243331193,
97.7305344800119 60.2699040334527 104.801921753145,
84.1637658057972 45.4402552309011 101.164131318869,
84.2875876756284 45.5495068447987 104.704586133534,
93.46662257 65.24922796 101.0274874,
93.64469111 65.08983591 101.0281293,
93.4685573728287 65.2459441020515 101.107219776609,
93.64442223 65.08856574 101.1057406,
93.18430189 64.58649456 101.0068144,
93.1715623119488 64.5713582298592 101.102296420975,
93.0001119848877 64.757048840524 101.00627245006,
92.9895267400264 64.7393236660879 101.103867471783,
93.28674073 65.41247654 101.2719219,
93.46480927 65.25308449 101.2725638,
93.2886755328287 65.4091926820515 101.351654276609,
93.46454039 65.25181432 101.3501751,
93.00442005 64.74974314 101.2512489,
92.9916804719488 64.7346068098592 101.346730920975,
92.8202301448877 64.920297420524 101.25070695006,
92.8096449000264 64.9025722460879 101.348301971783,
93.08501033 65.58743054 101.4493811,
93.26307887 65.42803849 101.450023,
93.0869451328287 65.5841466820515 101.529113476609,
93.26280999 65.42676832 101.5276343,
```

```
92.80268965 64.92469714 101.4287081,
92.7899500719488 64.9095608098592 101.524190120975,
92.6184997448877 65.095251420524 101.42816615006,
92.6079145000264 65.0775262460879 101.525761171783,
91.49084603 65.84565426 103.866461,
91.4818157 65.85951327 101.8521166,
92.15494759 65.21856654 101.8500804,
92.16791079 65.21753672 103.835956,
]
                        }
                        coordIndex [
0, 1, 3, 2, -1,
2, 4, 5, 3, -1,
34, 0, 6, 7, 4, 2, 0, 34, 35, 36, 37, -1,
6, 8, 9, 7, -1,
4, 7, 9, 5, -1,
10, 11, 13, 12, -1,
11, 14, 15, 13, -1,
14, 16, 17, 15, -1,
12, 17, 15, 13, -1,
18, 19, 21, 20, -1,
19, 22, 23, 21, -1,
22, 24, 25, 23, -1,
20, 25, 23, 21, -1,
26, 27, 29, 28, -1,
27, 30, 31, 29, -1,
30, 32, 33, 31, -1,
28, 33, 31, 29, -1,
1, 8, 9, 5, 3, -1,
0, 6, 8, 1, -1,
10, 16, 14, 11, -1,
10, 16, 17, 12, -1,
18, 24, 22, 19, -1,
18, 24, 25, 20, -1,
26, 32, 30, 27, -1,
26, 32, 33, 28, -1,
]
                                    }
                                    }
        Shape {
   appearance  Appearance {
    material    Material {
        diffuseColor 0 .5 .5
        specularColor .87 .25 .25
        ambientIntensity .157
        shininess .5
    }
   }
        geometry        IndexedFaceSet {
        solid FALSE
```

```
        coord Coordinate {
                point [
93.08758982 65.63430955 101.5106934,
92.4977791886693 64.9815666247071 101.51398734624,
93.0841096 65.62827047 101.6471826,
92.4992099059822 64.9809624748648 101.650449119872,
91.95429102 66.67857394 101.6564143,
91.3547834443976 66.0150994147085 101.659762401271,
91.94988337 66.67856498 101.5141982,
91.352004390772 66.0168928209344 101.517537205964,
93.5264369 64.24107576 101.0110404,
93.4311633560214 64.1356366097098 101.011572479137,
93.52029119 64.23809623 101.231439,
93.4298153612143 64.1379667070434 101.231944285087,
93.08498347 64.64851327 101.228316,
92.9860044991838 64.5389733269724 101.228868773028,
92.67816119 65.02035697 101.6391304,
92.5781023399404 64.9096219255299 101.639689203886,
92.63995077 65.08578165 101.6512182,
92.5245524320601 64.958070407307 101.651862671125,
92.5569453 65.13273649 101.2317556,
92.4544460138985 65.0193006170329 101.232328033117,
92.2369547 65.43016078 101.2234189,
92.130313419568 65.3121409654214 101.224014465129,
92.23526823 65.4297172 100.9686785,
92.128966176813 65.3120728079255 100.969272170629,
]
                }
                coordIndex [
0, 1, 3, 2, -1,
2, 3, 5, 4, -1,
4, 5, 7, 6, -1,
0, 1, 7, 6, -1,
0, 2, 4, 6, -1,
8, 9, 11, 10, -1,
10, 11, 13, 12, -1,
12, 13, 15, 14, -1,
14, 15, 17, 16, -1,
16, 17, 19, 18, -1,
18, 19, 21, 20, -1,
20, 21, 23, 22, -1,
8, 9, 23, 22, -1,
8, 10, 12, 14, 16, 18, 20, 22, -1,
]
                                }
                                }
```

Annexe 9. Programme PERL (fichier geocod3D.pl)

```
sub problemefichier {

  printf STDOUT "\nIl y a un probleme avec le fichier de donnees ... \n(verifiez le nom de celui-ci)\n";
  die();
}

sub wait {
  printf "\n-- ENTER to Continue --";
  $wait = <STDIN>;
}

open(INPUT,"input.txt");

open(OUTPUT,">geobase.vrml");
printf OUTPUT "#VRML V2.0 utf8

NavigationInfo {
  type \"EXAMINE\"
}\n" ;

open(LOG_FILE,">log.txt");

# on lit le fichier ligne par ligne
 $ligne=0;
 $meanligne=0;

while (<INPUT>) {
# on "split" la ligne en utilisant le ";" comme séparateur
 split(/\;/);

 print STDOUT "$_[0]\n";
 $i=0;
 @pointsep=split(/\./,$_);
 if($pointsep[0] eq  "OBJ"){  #On traite le code "obj"
       $obj = $pointsep[1];         # le nom de l'objet se trouve maintenant dans la variable $obj

       chop($obj);
       chop($obj);
       push(@objliste,$obj);            # @objetliste = liste de tous les objets croisés dans
```

```
                                                        #la lecture du fichier (avec redondance)
      $objligne[$meanligne]=$obj;    # cette variable permet de retrouver l'objet d'un point ou d'un code.
      printf LOG_FILE "
      $obj
      $meanligne
      ";
 }
#########################################################################################################
#isolation de connectivité dans les tableaux @con (liste des valeurs) @countcon(index pour retrouver les lignes)

 elsif($pointsep[0] eq "C"){
      #gestion de la connectivité implicite
      if($pointsep[1] eq "BT"){                    # code "BT"
            for($j=$meanligne;$j>=0;$j--){          # recupération du nom de l'objet
                 if(defined($objligne[$j])){
                       $newcode=$objligne[$j];
                       last;
                 }
           }
           $n=$pointsep[2];
           chop($n);
           chop($n);
           $m=$n-1;
           $p=$n+2;
           printf LOG_FILE"m $m";
           $F1="F1";
           $F2="F2";
           $Fn="F$n";
           $Fm="F$m";
           $B="B";
           $T="T";
           push(@con,"C");
           push(@con,"$newcode$F1");
           push(@con,"$newcode$F2");
           push(@con,"$newcode$Fn");
           push(@con,"$newcode$B");
           push(@con,"$newcode$T");
           push(@countcon,6);
           push(@con,"C");
```

```
      push(@con,"$newcode$Fn");
      push(@con,"$newcode$F1");
      push(@con,"$newcode$Fm");
      push(@con,"$newcode$B");
      push(@con,"$newcode$T");
      push(@countcon,6);
      for($i=2;$i<$n;$i++){
            $Fi="F$i";
            $h=$i-1;
            $Fh="F$h";
            $j=$i+1;
            $Fj="F$j";
            push(@con,"C");
            push(@con,"$newcode$Fi");
            push(@con,"$newcode$T");
            push(@con,"$newcode$B");
            push(@con,"$newcode$Fh");
            push(@con,"$newcode$Fj");
            push(@countcon,6);
      }
      push(@con,"C");
      push(@con,"$newcode$B");
      for($i=1;$i<=$n;$i++){
            $Fi="F$i";
            push(@con,"$newcode$Fi");
      }
      push(@countcon,$p);
      push(@con,"C");
      push(@con,"$newcode$T");
      for($i=1;$i<=$n;$i++){
            $Fi="F$i";
            push(@con,"$newcode$Fi");
      }
      push(@countcon,$p);

      printf LOG_FILE"con @con
      countcon @countcon";
}
else{
```

```
            for($j=$meanligne;$j>=0;$j--){          # boucle servant à introduire le nom de
                  if(defined($objligne[$j])){             # l'objet concerné dans le nom de la face
                        $newcode=$objligne[$j];           # afin de le rendre unique
                        last;
                  }
            }
            @split=split(/\./,$_[0]);
            $count=@split;                #$count contient le nombre d'éléments du tableau @split
            foreach $split(@split) {

                  push(@con,"$newcode$split");
            }
            push(@countcon,$count);
       }

}
#########################################################################################################
# récupération des valeurs des codes et des mesures
 else{
      $id[$ligne] = $_[0];
      $x[$ligne] = $_[1];
      $y[$ligne] = $_[2];
      $z[$ligne] = $_[3];
      $code[$ligne] = $_[4];
      $codeligne[$ligne] = $meanligne;   # ce tableau va permettre de faire le lien entre les codes et les objets
      $ligne++;
 }
 $meanligne++;
 next;
}

printf LOG_FILE "il y avait $ligne mesures dans le fichier\n";
close(INPUT);


#########################################################################################################

# Gestion des différents codes: a chaque code rencontré, l'algorithme effectue une opération particulière
```

```
$mo=0;
$c=0;
$proe=0;
$enfo=0;
$para=0;
$ct=0;
@rep;

for($i=0 ; $i<$ligne ; $i++){
      @difcode=split(/\//,$code[$i]);   # séparation des différents codes pour un même point
      foreach $difcode(@difcode){          # les codes séparés par des "/" sont donc traités indépendement.
            @list = split(/\./,$difcode);

#########################################################################################################
# Répétition d'objets
# Gestion du code "INSM" (objet modele de la répétition d'objet)

            if($list[0] eq "INSM"){
                  for($j=$codeligne[$i];$j>=0;$j--){    # recherche de l'objet correspondant
                        if(defined($objligne[$j])){
                              $modele[$mo]=$objligne[$j];
                              $insmx[$mo]=$x[$i];                  # on retient les coordonnées du point d'insertion
                              $insmy[$mo]=$y[$i];
                              $insmz[$mo]=$z[$i];
                              last;
                        }
                  }

                  $mo++;
            }

# Gestion du code "INS" (objet copie de la répétition d'objet)

            elsif($list[O] eq "INS"){
                  for($j=$codeligne[$i];$j>=0;$j--){    # recherche du nom de l'objet copié (@copie)
                        if(defined($objligne[$j])){
                              $copie[$c]=$objligne[$j];
                              $inscx[$c]=$x[$i];                        # on retient les coordonnées du point
d'insertion
```

```
                              $inscy[$c]=$y[$i];
                              $inscz[$c]=$z[$i];
                              $e=$i;
                              if($list[2] eq "REP"){              # gestion de la repetition multiple d'une copie
                                    $rep[$c]=$list[3];
                              }
                              else{
                                    $rep[$c]=1;
                              }

                              for ($u=0;$u<$mo;$u++){           # recherche du nom du modele correspondant
                                    if($modele[$u] == $list[1]){
                                          $deltax[$c]=$inscx[$c]-$insmx[$u];   #calcul des paramètes delta de
translation
                                          $deltay[$c]=$inscy[$c]-$insmy[$u];
                                          $deltaz[$c]=$inscz[$c]-$insmz[$u];
                                          $ref=$modele[$u];
                                    }
                              }
                              $h=0;
                              $y=0;
                              $end=0;
                              while(defined($objliste[$h])){      # cette partie du code sert à trouver entre quelle
                                    if($objliste[$h] eq $ref){        # et quelle ligne on va rechercher les points
correspondant
                                          if(defined($objliste[$h+1])){# à l'objet modele
                                                $objnext[$y]=$objliste[$h+1];
                                                $y++;
                                          }
                                    }
                                    $h++;
                              }

#réouverture du fichier INPUT pour introduire la connectivité de l'objet copié (à partir de celle de l'objet modèle)

                              open(INPUT,"input.txt");
                              $obj=0;
                              while (<INPUT>) {
                              split(/\;/);
```

```
                              @pointsep=split(/\./,$_);           # Il faut d'abord passer sur l'instruction OBJ
concernant l'objet modèle (if)
                                    if($pointsep[0] eq  "OBJ"){         # et seulement après traiter la connectivité
(elsif)
                                          $obj = $pointsep[1];
                                          chop($obj);
                                          chop($obj);
                                    }
                                    elsif($pointsep[0] eq "C" and $obj eq $ref){
                                          for($f=1;$f<=$rep[$c];$f++){
                                                $newcode="$copie[$c]$f";          # rappel: $copie[$c] est le nom de
l'objet copié
                                                @split=split(/\./,$_[0]);
                                                $count=@split;
                                                foreach $split(@split) {
                                                      push(@con,"$newcode$split");
                                                }
                                          push(@countcon,$count);
                                          }
                                    }
                              }
                              close(INPUT);
                              printf LOG_FILE "con @con";

# fin de la connectivité de l'objet copié
                              $dxoriginel[$c]=$deltax[$c];
                              $dyoriginel[$c]=$deltay[$c];
                              $dzoriginel[$c]=$deltaz[$c];

                              for($f=1;$f<=$rep[$c];$f++){
                                    $objligne[$meanligne]="$copie[$c]$f"; #introduction du nouvel objet dans le
programme
                                    $meanligne++;
                                    $deltax[$c]=$f*$dxoriginel[$c];     #calcul des paramètes delta de translation
                                    $deltay[$c]=$f*$dyoriginel[$c]; #pour chaque nouvelle répétition
                                    $deltaz[$c]=$f*$dzoriginel[$c];
                                    $l=0;                                          #recherche des points correspondants à
l'objet modèle
                                    $t=0;
```

```
                                    for($k=0;$k<$meanligne;$k++){
                                          if($objligne[$k] eq $ref){
                                                for($g=$k;$g<=$meanligne;$g++){
                                                      if($objligne[$g] eq $objnext[$l]){
                                                            for($r=$k;$r<$g;$r++){      # on en a trouvé entre quelle
et quelle ligne
                                                                  $u=0;                         # on devait traiter
les points
                                                                  while(defined($codeligne[$u])){

                                                                        if($codeligne[$u]==$r){
                                                                              if(defined($id[$u])){
      #application de la transformation de coordonnées entre modele et copie
                                                                                    $newx=$x[$u]+$deltax[$c];
                                                                                    $newy=$y[$u]+$deltay[$c];
                                                                                    $newz=$z[$u]+$deltaz[$c];

       #on enleve le code d'insertion sur l'objet copié
                                                                                    @codesplit=split(/\//,$code[$u]);
                                                                                          foreach $codesplit
(@codesplit) {

      @codesplit2=split(/\./,$codesplit);
                                                                                                foreach $codesplit2
(@codesplit2) {
                                                                                                      if($codesplit2
eq "INSM" or $codesplit2 eq "OR1M" or $codesplit2 eq "OR2M"){

      $codesplit="";
                                                                                                      }
                                                                                                }
                                                                                          }
                                                                                    $newcode=join(/\//,@codesplit);
      #On insère toutes les coordonnées de l'objet copié dans l'algorithme général

                                                                                    push(@x,$newx);
                                                                                    push(@y,$newy);
                                                                                    push(@z,$newz);
                                                                                    push(@code,$newcode);
                                                                                    push(@codeligne,$meanligne);
```

```
                                                                        printf LOG_FILE"newx $newx  ";
                                                                        $ligne++;
                                                                        $meanligne++;
                                                                        $t++;
                                                                  }
                                                                  last;
                                                            }
                                                            $u++;
                                                      }
                                                }
                                          }
                                    }
                                    if(defined($objnext[$l+1])){
                                          $l++;
                                    }

                              }
                        }
                        $c++;
                        last;
                  }
            }
      }

#########################################################################################################
# cas général de la méthode par construction de plan

      elsif($list[0] eq "F"){
            for($j=$codeligne[$i];$j>=0;$j--){     # boucle servant à introduire le nom de l'objet concerné
                                                   # dans le nom de la face
                  if(defined($objligne[$j])){           # ===> pas de confusion entre les faces
                        $newcode=$objligne[$j];
                        foreach $list (@list) {
                              $list="$newcode$list";
                        }
                        last;
                  }
            }
```

```
                  $code0[$i] = $list[1];

                  if(defined($list[2])){
                        $code1[$i] = $list[2];
                  }
                  if(defined($list[3])){
                        $code2[$i] = $list[3];
                  }
            }
#########################################################################################################################
# gestion du code "lipr"

            elsif($list[0] eq "lipr"){
                  if($list[1] eq "0"){ #ouverture d'une liaison
                        $start=$i;
                        $f=$i;
                        $p=1;
                        $t=0;
                        printf OUTPUT"
Shape {
    appearance    Appearance {
      material    Material {
      diffuseColor 0 1 1
      specularColor .87 .25 .25
      ambientIntensity .157
      shininess .5
      }
    }
geometry    IndexedFaceSet {
solid FALSE
coord Coordinate {
point [
$x[$i] $y[$i] $z[$i],\n"
                  }
                  elsif($list[1] eq "2"){ #point donnant la hauteur
                        $hauteur=$i;
                        $deltahx=$x[$hauteur]-$x[$start];
                        $deltahy=$y[$hauteur]-$y[$start];
```

```
                        $deltahz=$z[$hauteur]-$z[$start];
# les lignes de codes mises en commentaire correspondent à un essai de construction géométriquement correcte
# au moyen de calcul d'angle d'orientation, mais l'utilisation de l'Arctg et ses contraintes de changement
# de signe rendait le problème assez complexe

#                       $dha=sqrt(($x[$start]-$x[$hauteur])**2+($y[$start]-$y[$hauteur])**2);
#                       $dhb=sqrt(($x[$start]-$x[$hauteur])**2+($y[$start]-$z[$hauteur])**2);
                        $p++;
                        $t++;
                }
                elsif($list[1] eq "5"){ # point donnant la largeur
                        $largeur=$i;
                        $deltalx=$x[$largeur]-$x[$start];
                        $deltaly=$y[$largeur]-$y[$start];
                        $deltalz=$z[$largeur]-$z[$start];
#                       $dla=sqrt(($x[$start]-$x[$largeur])**2+($y[$start]-$y[$largeur])**2);
#                       $dlb=sqrt(($x[$start]-$x[$largeur])**2+($y[$start]-$z[$largeur])**2);
                        $p++;
                        $t++;
                }
                if($t==2){    #une fois qu'on a la largeur et la hauteur, on peut écrire les trois premiers points
                        $point3=$i;
                        $x3=$x[$start]+$deltalx+$deltahx;
                        $y3=$y[$start]+$deltaly+$deltahy;
                        $z3=$z[$start]+$deltalz+$deltahz;
                        printf OUTPUT"$x[$hauteur] $y[$hauteur] $z[$hauteur],
                        $x3 $y3 $z3,
                        $x[$largeur] $y[$largeur] $z[$largeur],\n";
                        $t++;
                }

                elsif($list[1] eq "1"){
#                       $anglea=-atan2($y[$i]-$y[$f],$x[$i]-$x[$f]);
#                       $angleb=atan2($x[$i]-$x[$f],$z[$i]-$z[$f]);
#                       $dhx=$dha*sin($anglea);
#                       $dhy=$dha*cos($anglea);
#                       $dhz=$dhb*sin($angleb);
#                       $dlx=$dla*sin($anglea);
#                       $dly=$dla*cos($anglea);
```

```
#                       $dlz=$dlb*sin($angleb);
                        $x01=$x[$f];
                        $y01=$y[$f];
                        $z01=$z[$f];
                        $p++;
                        $x02=$x[$f]+$deltahx;
                        $y02=$y[$f]+$deltahy;
                        $z02=$z[$f]+$deltahz;
                        $p++;
                        $x03=$x[$f]+$deltahx+$deltalx;
                        $y03=$y[$f]+$deltahy+$deltaly;
                        $z03=$z[$f]+$deltahz+$deltalz;
                        $p++;
                        $x04=$x[$f]+$deltalx;
                        $y04=$y[$f]+$deltaly;
                        $z04=$z[$f]+$deltalz;
                        $p++;

                        $x1=$x[$i];
                        $y1=$y[$i];
                        $z1=$z[$i];
                        $p++;
                        $x2=$x1+$deltahx;
                        $y2=$y1+$deltahy;
                        $z2=$z1+$deltahz;
                        $p++;
                        $x3=$x1+$deltahx+$deltalx;
                        $y3=$y1+$deltahy+$deltaly;
                        $z3=$z1+$deltahz+$deltalz;
                        $p++;
                        $x4=$x1+$deltalx;
                        $y4=$y1+$deltaly;
                        $z4=$z1+$deltalz;
                        $p++;

                        printf OUTPUT"$x01 $y01 $z01,
                        $x02 $y02 $z02,
                        $x03 $y03 $z03,
                        $x04 $y04 $z04,
```

```
                $x1 $y1 $z1,
                $x2 $y2 $z2,
                $x3 $y3 $z3,
                $x4 $y4 $z4,\n";
                $f=$i;                          #$f = point précédent
            }
            elsif($list[1] eq "4"){         #point cloturant une liaison
#               $anglea=-atan2($y[$i]-$y[$f],$x[$i]-$x[$f]);
#               $angleb=atan2($x[$i]-$x[$f],$z[$i]-$z[$f]);
#               $dhx=$dha*sin($anglea);
#               $dhy=$dha*cos($anglea);
#               $dhz=$dhb*sin($angleb);
#               $dlx=$dla*sin($anglea);
#               $dly=$dla*cos($anglea);
#               $dlz=$dlb*sin($angleb);

                $x01=$x[$f];
                $y01=$y[$f];
                $z01=$z[$f];
                $p++;
                $x02=$x[$f]+$deltahx;
                $y02=$y[$f]+$deltahy;
                $z02=$z[$f]+$deltahz;
                $p++;
                $x03=$x[$f]+$deltahx+$deltalx;
                $y03=$y[$f]+$deltahy+$deltaly;
                $z03=$z[$f]+$deltahz+$deltalz;
                $p++;
                $x04=$x[$f]+$deltalx;
                $y04=$y[$f]+$deltaly;
                $z04=$z[$f]+$deltalz;
                $p++;

                $x1=$x[$i];
                $y1=$y[$i];
                $z1=$z[$i];
                $p++;
                $x2=$x1+$deltahx;
                $y2=$y1+$deltahy;
```

```
            $z2=$z1+$deltahz;
            $p++;
            $x3=$x1+$deltahx+$deltalx;
            $y3=$y1+$deltahy+$deltaly;
            $z3=$z1+$deltahz+$deltalz;
            $p++;
            $x4=$x1+$deltalx;
            $y4=$y1+$deltaly;
            $z4=$z1+$deltalz;
            $p++;

            printf OUTPUT"$x01 $y01 $z01,
            $x02 $y02 $z02,
            $x03 $y03 $z03,
            $x04 $y04 $z04,
            $x1 $y1 $z1,
            $x2 $y2 $z2,
            $x3 $y3 $z3,
            $x4 $y4 $z4,";
            $f=$i;
            printf OUTPUT"]
}
coordIndex [\n";
            printf OUTPUT"0, 1, 2, 3, -1
            ";
            for($k=4;$k<$p-6;$k=$k+8){
                  $a1=$k;
                  $b1=$k+1;
                  $c1=$k+2;
                  $d1=$k+3;
                  $a2=$k+4;
                  $b2=$k+5;
                  $c2=$k+6;
                  $d2=$k+7;
                  printf OUTPUT"$a1, $b1, $c1, $d1, -1,
                  $a1, $a2, $b2, $b1, -1,
                  $d1, $d2, $c2, $c1, -1,
                  $d1, $a1, $a2, $d2, -1,
                  $b1, $c1, $c2, $b2, -1,
```

```
                              $a2, $b2, $c2, $d2, -1\n";
                        }
                        printf OUTPUT"]
                    }
                    }";
                }
            }

      ##########################################################################################################
      #gestion du code "lip"

            elsif($list[0] eq "lip"){
                  if($list[1] eq "0"){    #ouverture d'une liaison
                        $b=0;
                        $start=$i;
                        $sol[$b]=0;
                        $b++;
                        printf OUTPUT"
Shape {
    appearance    Appearance {
      material    Material {
      diffuseColor 0 1 .5
      specularColor .87 .25 .25
      ambientIntensity .157
      shininess .5
      }
    }
geometry    IndexedFaceSet {
solid FALSE
coord Coordinate {
point [
$x[$i] $y[$i] $z[$i],\n"
                  }
                  elsif($list[1] eq "2"){ # poing donnant la hauteur
                        $t=O;
                        $hauteur=$i;
                        $deltax=$x[$hauteur]-$x[$start];
                        $deltay=$y[$hauteur]-$y[$start];
                        $deltaz=$z[$hauteur]-$z[$start];
```

```
            $toit[$t]=1;
            $t++;
            printf OUTPUT"$x[$i] $y[$i] $z[$i],\n";
            $p=2;
      }
      elsif($list[1] eq "1"){  # liaison
            $xb=$x[$i];
            $yb=$y[$i];
            $zb=$z[$i];
            $sol[$b]=$p;
            $b++;
            $p++;
            $xt=$xb+$deltax;
            $yt=$yb+$deltay;
            $zt=$zb+$deltaz;
            $toit[$t]=$p;
            $t++;
            $p++;
            printf OUTPUT"$xb $yb $zb,\n$xt $yt $zt,\n";
      }
      elsif($list[1] eq "3"){  # fermer sur le premier point
            $xb=$x[$i];
            $yb=$y[$i];
            $zb=$z[$i];
            $sol[$b]=$p;
            $b++;
            $p++;
            $xt=$xb+$deltax;
            $yt=$yb+$deltay;
            $zt=$zb+$deltaz;
            $toit[$t]=$p;
            $t++;
            $p++;
            printf OUTPUT"$xb $yb $zb,\n$xt $yt $zt,\n";
            printf OUTPUT"]
}
coordIndex [\n";
            for($k=0;$k<$p-3;$k=$k+2){
                  $a=$k;
```

```
                        $b=$k+1;
                        $c=$k+3;
                        $d=$k+2;
                        printf OUTPUT"$a, $b, $c, $d, -1,\n";
                  }
                  $q=$p-1;
                  $r=$p-2;
                  printf OUTPUT"$r, $q, 1, 0, -1,\n";
                  foreach $toit (@toit) {
                        printf OUTPUT"$toit, ";
                        $toit="";
                  }
                  printf OUTPUT "-1,\n";
                  foreach $sol (@sol) {
                        printf OUTPUT"$sol, ";
                        $sol="";
                  }
                  printf OUTPUT "-1,\n";
                  printf OUTPUT"]
            }
            }";
      }
}
#############################################################################################################
# gestion du code PRO
      elsif($list[0] eq "PRO"){
            $nompro[$proe]=$list[1];
            for($j=$codeligne[$i];$j>=0;$j--){     # recherche du nom de l'objet
                  if(defined($objligne[$j])){
                        $nomobj=$objligne[$j];
                        last;
                  }
            }
            if(defined($list[2])){
                  $nomface[$proe]="$nomobj$list[2]";
            }
            else{
                  printf "\nProbleme dans le fichier input: code PRO incomplet\n";
            }
```

```
                  $xpro[$proe]=$x[$i];
                  $ypro[$proe]=$y[$i];
                  $zpro[$proe]=$z[$i];
                  $proe++;
            }
            for($d=0;$d<@nompro;$d++){          # gestion des autres points se rapportant à une certaine proeminence
                  if($nompro[$d] eq $list[0]){
                        if($list[1] eq "1"){
                              $xpro[$proe]=$x[$i];       # ces tableaux contiennent tous les points de proéminence
                              $ypro[$proe]=$y[$i];
                              $zpro[$proe]=$z[$i];
                              $nomface[$proe]=$nomface[$d];# ce tableau permet de voir à quel face se rapporte la
proéminence
                              $proe++;
                        }
                        elsif($list[1] eq "3"){
                              $xpro[$proe]=$x[$i];
                              $ypro[$proe]=$y[$i];
                              $zpro[$proe]=$z[$i];
                              $nomface[$proe]="end";     # le code "end" permet de différencier les différentes
proéminences
                              $proe++;
                        }
                  }
            }
#############################################################################################################
 # gestion du code ENF

            if($list[0] eq "ENF"){
                  $nomenf[$enfo]=$list[1];
                  for($j=$codeligne[$i];$j>=0;$j--){     # recherche du nom de l'objet
                        if(defined($objligne[$j])){
                              $nomobj=$objligne[$j];
                              last;
                        }
                  }
                  if(defined($list[2])){
                        $nomfaceenf[$enfo]="$nomobj$list[2]";
                  }
```

```
                else{
                        printf "\nProbleme dans le fichier input: code ENF incomplet\n";
                }
                $xenf[$enfo]=$x[$i];
                $yenf[$enfo]=$y[$i];
                $zenf[$enfo]=$z[$i];
                $enfo++;
        }
        for($d=0;$d<@nomenf;$d++){          #gestion du point paramétrique donnant la distance d'enfoncement
                if($nomenf[$d] eq $list[0]){
                        if($list[1] eq "PAR"){
                                $parenf[$para][0]=$x[$i];
                                $parenf[$para][1]=$y[$i];
                                $parenf[$para][2]=$z[$i];
                                $para++;
                        }
                        if($list[1] eq "1"){            #gestion des autres points se rapportant à un certain
enfoncement
                                $xenf[$enfo]=$x[$i];       # ces tableaux contiennent tous les points d'enfoncement
                                $yenf[$enfo]=$y[$i];
                                $zenf[$enfo]=$z[$i];
                                $nomfaceenf[$enfo]=$nomfaceenf[$d]; #ce tableau permet de voir à quel face
                                $enfo++;                                                #se rapporte l'enfoncement
                        }
                        elsif($list[1] eq "3"){
                                $xenf[$enfo]=$x[$i];
                                $yenf[$enfo]=$y[$i];
                                $zenf[$enfo]=$z[$i];
                                $nomfaceenf[$enfo]="end";   # le code "end" permet de différencier les différentes
enfoncements
                                $enfo++;
                        }
                }
        }

#########################################################################################################

        if($list[0] eq "CT"){               # gestion du code "CT"
                $numct[$ct]=$list[1];
```

```
                  for($j=$codeligne[$i];$j>=0;$j--){   # recherche du nom de l'objet
                        if(defined($objligne[$j])){
                              $nomobj=$objligne[$j];
                              last;
                        }
                  }
                  if(defined($list[1])){
                        $nomct[$ct]="$nomobj$list[1]";
                  }
                  else{
                        printf "\nProbleme dans le fichier input: code CT incomplet\n";
                  }
                  $xct[$ct]=$x[$i];
                  $yct[$ct]=$y[$i];
                  $zct[$ct]=$z[$i];
                  $ct++;
            }
      }
}
# Ecriture des faces issues du code CT en VRML

M:for($i=0;$i<$ct;$i++){
      foreach $fait (@fait) {
            if($fait eq $nomct[$i]){   # si la face a déjà été faite, on passe à la ligne suivante
                  next M;
            }
      }
      $s=0;
      printf OUTPUT"
      Shape {
    appearance    Appearance {
      material    Material {
      diffuseColor 0 .5 .5
      specularColor .87 .25 .25
      ambientIntensity .157
      shininess .5
      }
    }
      geometry    IndexedFaceSet {
```

```
	solid FALSE
	coord Coordinate {
		point [\n";
	$currentface=$nomct[$i];
	push(@fait,$nomct[$i]);
	for($j=$i;$j<$ct;$j++){
		if($nomct[$j] eq $currentface){
			printf OUTPUT "$xct[$j] $yct[$j] $zct[$j]\n";
			$s++;
		}
	}
	print OUTPUT "]
		}
		coordIndex [
		";
	for($u=0;$u<$s;$u++){
		printf OUTPUT "$u, ";
	}
	printf OUTPUT"-1,
		]
				}
				}
		";

}

# Fin de la gestion des codes
#############################################################################################################
#############################################################################################################
# On revient à la codification par contruction de plan

# creation d'un tableau @repface répertoriant les différentes faces = liste des différentes faces sans redondance

@repface;

for($i=0 ; $i<$ligne ; $i++){
	$flag=0;
	foreach $repface (@repface) {
		if($repface eq $code0[$i]){
```

```
                  $flag=1;
            }
      }
      if($flag==0){
            push(@repface,$code0[$i]);
      }
}

for($i=0 ; $i<$ligne ; $i++){
      $flag=0;
      foreach $repface (@repface) {
            if($repface eq $code1[$i]){
                  $flag=1;
            }
      }
      if($flag==0){
            push(@repface,$code1[$i]);
      }
}

for($i=0 ; $i<$ligne ; $i++){
      $flag=0;
      foreach $repface (@repface) {
            if($repface eq $code2[$i]){
                  $flag=1;
            }
      }
      if($flag==0){
            push(@repface,$code2[$i]);
      }
}
#On nettoie les blancs dans les tableaux repface, x, y, z, code0, code1, code2, on supprime les mesures sans code

for($i=0;$i<@repface;$i++){
      if (not defined($repface[$i])) {
            for($j=$i;$j<@repface;$j++){
                  $repface[$j]=$repface[$j+1];
            }
      }
```

```
}

for($i=0;$i<@code0;$i++){
      if (not defined($code0[$i])) {
            for($j=$i;$j<@code0;$j++){
                  $code0[$j]=$code0[$j+1];
                  $code1[$j]=$code1[$j+1];
                  $code2[$j]=$code2[$j+1];
                  $y[$j]=$y[$j+1];
                  $x[$j]=$x[$j+1];
                  $z[$j]=$z[$j+1];
            }
      }
}


#########################################################################################################################

# creation d'un tableau bidimensionel (@face) contenant tous les points pour chaque face.
# nb: les faces sont renommées numériquement par le programme dans le tableau @face. Le tableau @repface permet
# donc de retrouver les anciens noms.

@face;
@count;     # tableau @count va mémoriser le nombre de points par face
$n=0;

foreach $repface (@repface) {
      $n++;
}
# $n compte le nombre d'éléments du tableau @repface

for($i=0; $i<$n; $i++){    # initialise le compteur à zero pour chaque element du tableau
      $count[$i] = 0;
}

for($i=0; $i<$ligne; $i++){
      for($j=0; $j<$n; $j++){
            if($code0[$i] eq $repface[$j]){
                  $face[$j][$count[$j]]=$i;
```

```
                $count[$j]++;
            }
            if($code1[$i] eq $repface[$j]){
                $face[$j][$count[$j]]=$i;
                $count[$j]++;
            }
            if($code2[$i] eq $repface[$j]){
                $face[$j][$count[$j]]=$i;
                $count[$j]++;
            }
      }
}
$d=@repface;

# Attention, les id des points débutent mtnt à zéro.
#########################################################################################################
# On va calculer les équations de plan des faces

# Verification du nombre de points par plan (au moins 3)

foreach $count (@count) {
      if ($count<3) {
            printf "Erreur: il y a moins de 3 points pour une face";
      }
}

# Recherche des paramètres de l'équation du plan de chaque face (résolution matricielle à partir de 3 points)
@plan;

for($i=0; $i<$ligne; $i++){
      $xa = $x[$face[$i][0]];
      $xb = $x[$face[$i][1]];
      $xc = $x[$face[$i][2]];
      $ya = $y[$face[$i][0]];
      $yb = $y[$face[$i][1]];
      $yc = $y[$face[$i][2]];
      $za = $z[$face[$i][0]];
      $zb = $z[$face[$i][1]];
      $zc = $z[$face[$i][2]];
```

```
# Les coordonnées de trois points sont récupérées dans la boucle plan par plan.

      $xab=$xb-$xa;
      $yab=$yb-$ya;
      $zab=$zb-$za;
      $xac=$xc-$xa;
      $yac=$yc-$ya;
      $zac=$zc-$za;

      $plan[$i][0] = ($yab*$zac)-($zab*$yac);                                                    #a
      $plan[$i][1] = (-1)*(($xab*$zac)-($zab*$xac));                                        #b
      $plan[$i][2] = ($xab*$yac)-($yab*$xac);                                                    #c
      $plan[$i][3] = ((-1*$xa*$plan[$i][0])+(-1*$ya*$plan[$i][1])+(-1*$za*$plan[$i][2]));      #d

# les 4 paramètres du plan sont trouvés: ax+by+cz+d=0
# et stockés dans un tableau
}

#############################################################################################

# on revient à la connectivité: obtention d'un tableau @connec[][] où pour chaque face, on a une liste des faces
# qui lui sont connectées.

@connec;
$u=0;
$c = @countcon;
for($i=0; $i<$c; $i++){
      for($j=0; $j<$countcon[$i]; $j++){
            $connec[$i][$j]=$con[$j+$u];
      }
      $u=$u+$j;
}

# @connec contient maintenant les mêmes infos que @countcon et @con (connectivité)
# dans un seul tableau bidimensionel

# création d'un tableau avec les informations de connectivité selon
# la première méthode (face principal + faces secondaires)
```

```
# et mise dans le même ordre que le tableau @repface

$k=0;

foreach $repface (@repface){
      $connecti[$k][0]= $repface ;
      $m=1;
OUF:for($i=0; $i<$c; $i++){
            if($connec[$i][1] eq $repface){
BILLEN:           for($j=2; $j<$countcon[$i]; $j++){  # BILLEN = étiquette servant à réitérer la boucle via
l'instruction next
                        for($mm=0;$mm<=$m;$mm++){                   # si la face ne figure pas déjà dans le tableau,
on l'ajoute
                              if($connecti[$k][$mm] eq $connec[$i][$j]){
                                    next BILLEN;
                              }
                        }
                        $connecti[$k][$m] = $connec[$i][$j];
                        $m++;
                  }

            }
            else{

DONNAY:           for($j=2;$j<$countcon[$i];$j++){
                        if($connec[$i][$j] eq $repface){
                              for($mm=0;$mm<=$m;$mm++){      #si la face ne figure pas déjà dans le tableau, on
l'ajoute
                                    if($connecti[$k][$mm] eq $connec[$i][1]){
                                          next DONNAY;
                                    }
                              }
                              $connecti[$k][$m]=$connec[$i][1];
                              $m++;
                              next OUF;
                        }
                  }
            }
      }
```

```
      $k++;
}

##############################################################################################
# on va maintenant rechercher les sommets formés par des triplets de plans
# (intersection de 3 plans forment un point)

@sommet;
$i=0;
$s=0;
while (defined($connecti[$i][0])){
      $j=1;
      while (defined($connecti[$i][$j])){
            $u=$j+1;
          while(defined($connecti[$i][$u])){ #on a deux faces qui sont connectées à une première.
                  $i2=0;                                        #Il faut que les deux faces en question
OZER:       while (defined ($connecti[$i2][0])) {    # soient connectées ensemble pour satisfaire la condition
                        if($connecti[$i2][0] eq $connecti[$i][$u]){
                              $j2=0;
                              while (defined($connecti[$i2][$j2])) {     # la troisieme face se trouve-t'elle dans la
ligne
                                                                                    #  de la deuxieme, autrement dit:
lui est elle connectée?
# exclusion du cas d'une face triangulaire (la condition est remplie sans pour autant
# que les faces s'intersectent en un même point)
                                    $a=0;
                                    while(defined($connecti[$a][0])){
                                          $b1=1;
                                          while(defined($connecti[$a][$b1])){
                                                if($connecti[$a][$b1] eq $connecti[$i][0] and not
defined($connecti[$a][4])){
                                                      $b2=1;
                                                      while(defined($connecti[$a][$b2])){
                                                            if($connecti[$a][$b2] eq $connecti[$i][$j]){
                                                                  $b3=1;
                                                                  while(defined($connecti[$a][$b3])){
                                                                        if($connecti[$a][$b3] eq $connecti[$i][$u]){
                                                                              last OZER;
                                                                        }
```

```
                                                $b3++;
                                          }
                                    }
                                    $b2++;
                              }
                        }
                        $b1++;
                  }
                  $a++;
            }
            if($connecti[$i2][$j2] eq $connecti[$i][$j]){  #supression des redondances
                  $v=0;
                  $base[0] = $connecti[$i][0];
                  $base[1] = $connecti[$i][$j];
                  $base[2] = $connecti[$i][$u];
                  @base=sort(@base);              #sort = classement alphabétique
                  $base=join(":",@base);          #join = regrouper le tableau dans une seule
variable

                  while (defined($sommet[$v][0])) {
                        $test[0] = $sommet[$v][0];
                        $test[1] = $sommet[$v][1];
                        $test[2] = $sommet[$v][2];
                        @test=sort(@test);
                        $test=join(":",@test);

                        if($test eq $base){
                              last OZER;
                        }
                        $v++
                  }

                  $sommet[$s][0] = $connecti[$i][0];
                  $sommet[$s][1] = $connecti[$i][$j];
                  $sommet[$s][2] = $connecti[$i][$u];
                  printf LOG_FILE "s $i";
                  $s++;
#                 last;
            }
```

```
                                    $j2++;
                                }
#                               last;
                            }
                            $i2++;
                        }
                        $u++;
                }
                $j++;
        }
        $i++;
}

#############################################################################################################
# calcul des coordonnées des sommets: résolution de systèmes de 3 équations (équations de plan)
# à 3 inconnues (coordonnées):
# a1x + b1y + c1z + d1 = 0
# a2x + b2y + c2z + d2 = 0
# a3x + b3y + c3z + d3 = 0
#
#@sommet contient les 3 plans (= un point)
#@plan contient les 4 paramètes d'équation de chaque plan
#@repface contient le nom des plans se rapportant à @plan

$i=0;
while (defined($sommet[$i][0])) {
        $i2=0;
        while (defined($repface[$i2])) {
                if($repface[$i2] eq $sommet[$i][0]){
                        $a1[$i]=$plan[$i2][0];
                        $b1[$i]=$plan[$i2][1];
                        $c1[$i]=$plan[$i2][2];
                        $d1[$i]=$plan[$i2][3];
                }
                elsif($repface[$i2] eq $sommet[$i][1]){
                        $a2[$i]=$plan[$i2][0];
                        $b2[$i]=$plan[$i2][1];
                        $c2[$i]=$plan[$i2][2];
```

```
                  $d2[$i]=$plan[$i2][3];
            }
            elsif($repface[$i2] eq $sommet[$i][2]){
                  $a3[$i]=$plan[$i2][0];
                  $b3[$i]=$plan[$i2][1];
                  $c3[$i]=$plan[$i2][2];
                  $d3[$i]=$plan[$i2][3];
            }

            $i2++;
      }
      $i++;
}

# résolution matricielle des équations par l'algoritme de Gauss-Jordan
# on va d'abord mettre les variable en place afin qu'elles soient compatibles avec l'algoritme (faux tableaux 1D)
#(ancien algoritme programmé en C++ en deuxieme candi)
# N= dimension de la matrice a (carrée)
# P= nombre de colonnes de la matrice B
# AX=B
#      a1 b1 c1    x     -d1
#      a2 b2 c2 *  y  =  -d2
#      a3 b3 c3    z     -d3

$m=0;
@a;
@b;
@x;
while (defined($a1[$m])) {
      $a[0] = $a1[$m];
      $a[1] = $b1[$m];
      $a[2] = $c1[$m];
      $a[3] = $a2[$m];
      $a[4] = $b2[$m];
      $a[5] = $c2[$m];
      $a[6] = $a3[$m];
      $a[7] = $b3[$m];
      $a[8] = $c3[$m];
      $b[0] = -$d1[$m];
```

```
	$b[1] = -$d2[$m];
	$b[2] = -$d3[$m];
	$n=3;
	$p=1;
	# Gauss-Jordan
	# attention: @a=@z
	$sing=0;

	for ($j=0;$j<$n;$j++){
		$max=0;
		for($i=$j;$i<$n;$i++){
			if($max<abs($a[$i*$n+$j])){
				$max=abs($a[$i*$n+$j]);
				$pivot=$i;
			}
		}
# la valeur du pivot est trouvée
		if($max==0){
			$sing=1;
			printf "Problème: impossibilité de résoudre la face $repface[$m] (matrice singulière)";
			last;
		}
		else{

			for ($jam=0;$jam<$n;$jam++){
				$tempo=$a[$pivot*$n+$jam];
				$a[$pivot*$n+$jam]=$a[$j*$n+$jam];
				$a[$j*$n+$jam]=$tempo;
			}

			for ($jam=0;$jam<$p;$jam++){
				$tempo=$b[$pivot*$p+$jam];
				$b[$pivot*$p+$jam]=$b[$j*$p+$jam];
				$b[$j*$p+$jam]=$tempo;
			}

# les lignes sont interverties pour la matrice a et b

			$ajj=$a[$j*$n+$j];
```

```
                  for ($jam=0;$jam<$n;$jam++){
                        $a[$j*$n+$jam]=$a[$j*$n+$jam]/$ajj;
                  }

                  for ($jam=0;$jam<$p;$jam++){

                        $b[$j*$p+$jam]=$b[$j*$p+$jam]/$ajj;
                  }

                  for($i=0;$i<$n;$i++){

                        if($i!=$j){
                              $aij=$a[$i*$n+$j];
                              for($jam=0;$jam<$n;$jam++){
                                    $a[$i*$n+$jam]=$a[$i*$n+$jam]-$aij*$a[$j*$n+$jam];
                              }
                              for($jam=0;$jam<$p;$jam++){
                                  $b[$i*$p+$jam]=$b[$i*$p+$jam]-$aij*$b[$j*$p+$jam];
                              }
                        }
                  }
            }
            $sing=0;

      }
# Les coordonnées des sommets sont stockées dans les tableaux @sx, @sy et @sz.
      $sx[$m]=$b[0];
      $sy[$m]=$b[1];
      $sz[$m]=$b[2];
      $m++;
}
# fin de Gauss Jordan
#############################################################################################
#Gestion des enfoncements

$flag=0;
$para=0;
for($i1=0;$i1<@nomfaceenf;$i1++){
      if($flag==0){
```

```
            for($j1=0;$j1<@repface;$j1++){
                  if($repface[$j1] eq $nomfaceenf[$i1]){

#Recherche de l'équation du plan parallèle au plan de base de l'enfoncement et comprenant le point paramétrique
#==> recherche de l'inconnue d dans l'équation ax+by+cz=d, (a, b, c) etant le même vecteur directeur puisque
#les plans sont parallèles.

                        $flag=1;
                        $a=$plan[$j1][0];
                        $b=$plan[$j1][1];
                        $c=$plan[$j1][2];
                        $d=$a*$parenf[$para][0]+$b*$parenf[$para][1]+$c*$parenf[$para][2];
                        $para++;
                  }
            }
      }
      if($nomfaceenf[$i1] eq "end"){
            $flag=0;
      }
      # Gauss Jordan pour retrouver les points d'intersection avec la face de base de l'enfoncement
      # Recherche d'un point appartenant à la fois à une droite perpendiculaire à un plan et au dit-plan.
      # On connait l'équation cartésienne du plan et équation paramétrique de la droite (=vecteur normal directeur
du plan)
      # système de 4 équation à 4 inconnues
      # x = x0 + r*a
      # y = y0 + r*b
      # z = z0 + r*c
      # ax + by + cz = d

      $a[0] = $a;
      $a[1] = $b;
      $a[2] = $c;
      $a[3] = 0;
      $a[4] = 1;
      $a[5] = 0;
      $a[6] = 0;
      $a[7] = -$a;
      $a[8] = 0;
      $a[9] = 1;
```

```
	$a[10] = 0;
	$a[11] = -$b;
	$a[12] = 0;
	$a[13] = 0;
	$a[14] = 1;
	$a[15] = -$c;

	$b[0] = $d;
	$b[1] = $xenf[$i1];
	$b[2] = $yenf[$i1];
	$b[3] = $zenf[$i1];
	$n=4;
	$p=1;
	# Gauss-Jordan
	# attention: @a=@z
	$sing=0;

	for ($j=0;$j<$n;$j++){
		$max=0;
		for($i=$j;$i<$n;$i++){
			if($max<abs($a[$i*$n+$j])){
				$max=abs($a[$i*$n+$j]);
				$pivot=$i;
			}
		}
# la valeur du pivot est trouvée
		if($max==0){
			$sing=1;
			printf "Problème: impossibilité de résoudre la proeminence concernant $nomface[$i1] (matrice
singulière)";
			last;
		}
		else{

			for ($jam=0;$jam<$n;$jam++){
				$tempo=$a[$pivot*$n+$jam];
				$a[$pivot*$n+$jam]=$a[$j*$n+$jam];
				$a[$j*$n+$jam]=$tempo;
			}
```

```
                  for ($jam=0;$jam<$p;$jam++){
                        $tempo=$b[$pivot*$p+$jam];
                        $b[$pivot*$p+$jam]=$b[$j*$p+$jam];
                        $b[$j*$p+$jam]=$tempo;
                  }

# les lignes sont interverties pour la matrice a et b

                  $ajj=$a[$j*$n+$j];
                  for ($jam=0;$jam<$n;$jam++){
                        $a[$j*$n+$jam]=$a[$j*$n+$jam]/$ajj;
                  }

                  for ($jam=0;$jam<$p;$jam++){

                        $b[$j*$p+$jam]=$b[$j*$p+$jam]/$ajj;
                  }

                  for($i=0;$i<$n;$i++){

                        if($i!=$j){
                              $aij=$a[$i*$n+$j];
                              for($jam=0;$jam<$n;$jam++){
                                    $a[$i*$n+$jam]=$a[$i*$n+$jam]-$aij*$a[$j*$n+$jam];
                              }
                              for($jam=0;$jam<$p;$jam++){
                                    $b[$i*$p+$jam]=$b[$i*$p+$jam]-$aij*$b[$j*$p+$jam];
                              }
                        }
                  }
            }
            $sing=0;

      }
      $xbenf[$i1]=$b[0];
      $ybenf[$i1]=$b[1];
      $zbenf[$i1]=$b[2];
# fin de Gauss Jordan
```

```
}
# on va maintenant tracer les enfoncements en VRML

printf OUTPUT"
      Shape {
    appearance    Appearance {
      material    Material {
      diffuseColor 0 .5 .5
      specularColor .87 .25 .25
      ambientIntensity .157
      shininess .5
      }
    }
      geometry    IndexedFaceSet {
      solid FALSE
      coord Coordinate {
            point [\n";
for($i=0;$i<@nomfaceenf;$i++){
      printf OUTPUT"$xenf[$i] $yenf[$i] $zenf[$i],\n$xbenf[$i] $ybenf[$i] $zbenf[$i],\n";
}
printf OUTPUT"]
            }
            coordIndex [\n";
$p=0;
for($i=2;$i<@nomfaceenf*2;$i=$i=$i+2){
      if($nomfaceenf[$i/2] eq "end"){
            $g=$p;
            $h=$p+1;
            $j=$i+1;
            printf OUTPUT "$g, $h, $j, $i, -1,\n";
            for($u=$p+1;$u<=$i+1;$u=$u+2){
                  printf OUTPUT"$u, ";
            }
            printf OUTPUT"-1, \n";
            $p=$i+2;
      }
      $h=$i-1;
      $g=$i-2;
      $j=$i+1;
```

```
      printf OUTPUT "$g, $h, $j, $i, -1,\n";
}
printf OUTPUT"]
                           }
                           }";
#############################################################################################################
# ecriture des coordonnées des faces en VRML (code F)
$n=0;
if(defined($sx[0])){
      printf OUTPUT"
      Shape {
    appearance    Appearance {
      material    Material {
      diffuseColor 0 .5 .5
      specularColor .87 .25 .25
      ambientIntensity .157
      shininess .5
      }
    }
      geometry    IndexedFaceSet {
      solid FALSE
      coord Coordinate {
            point [\n"
}
while(defined($sx[$n])){
      printf OUTPUT "$sx[$n] $sy[$n] $sz[$n],\n";
      $n++;
}
$nn=$n;                  # ecriture des points d'enfoncement
for($i=0;$i<@xenf;$i++){
      printf OUTPUT "$xenf[$i] $yenf[$i] $zenf[$i],\n";
}

if(defined($repface[$0])){
      printf OUTPUT"]
                  }
                  coordIndex [\n";

#écriture des faces en VRML (tourner autour le long des arêtes)
```

```
$flag=0;
$r=0;
BB:while(defined($repface[$r])){
      $enflag=0;
      for($a=0;$a<@nomfaceenf;$a++){    #une face avec un enfoncement doit être dessinée différement
            if($nomfaceenf[$a] eq $repface[$r]){
                  $nn=$n+$a;
                  printf OUTPUT"$nn, ";
                  $enflag=1;
                  last;
            }
      }
      foreach $done (@done) {
            $done=0;
      }
      $i=0;
YEW:while(defined($sommet[$i][0])){
            $j=0;
            while(defined($sommet[$i][$j])){
                  if($sommet[$i][$j] eq $repface[$r]){
                        printf OUTPUT "$i, ";
                        $again=$i;
                        push(@done,$sommet[$i][0]);
                        push(@done,$sommet[$i][1]);
                        push(@done,$sommet[$i][2]);

                        if($j==2){
                              $jam=1;
                        }
                        else{
                              $jam=$j+1;
                        }
                        $imean=$i;
                        $jmean=$j;
                        last YEW;
                  }
                  $j++;
            }
```

```
                $i++;
        }

# on a le point de départ et le sens de rotation

        $i=0;
    KID:while(defined($sommet[$i][0])){
                $j=0;
                while(defined($sommet[$i][$j])){
                        if($sommet[$i][$j] eq $sommet[$imean][$jmean] and $i != $imean){
                                $v=0;
                                while(defined($sommet[$i][$v])){
                                        if($sommet[$i][$v] eq $sommet[$imean][$jam]){
                                                printf OUTPUT "$i, ";
                                                $iout=$i;
                                                $jout=$v;
                                                last KID;
                                        }
                                        $v++;
                                }
                        }
                        $j++;
                }
                $i++;
        }

        $j=0;
    SPAIN:while(defined($sommet[$iout][$j])){
                $flag=0;
                if($sommet[$iout][$j] eq  $sommet[$imean][$jmean]){
                        $flag=1;
                }
                if($sommet[$iout][$j] eq $sommet[$iout][$jout]){
                        $flag=1;
                }
                if($flag==0){
                        $icom=$iout;
                        $jcom=$j;
                        last;
```

```
	}
	$j++;
}
foreach $done (@done) {
	if($sommet[$icom][$jcom] eq $done){
		if($enflag==1){	#cas d'une face avec enfoncement
			printf OUTPUT"$again, ";
			while($nomfaceenf[$a] eq $repface[$r]){	# à la fin de la boucle, nomfaceenf est "end"
				printf OUTPUT "$nn, ";
				$a++;
				$nn++;
			}
			print OUTPUT"$nn, ";
		}
		printf OUTPUT "-1,\n";
		$r++;
		next BB;
	}
}
$i=0;
while(defined($sommet[$i][$0])){
	$j=0;
	while(defined($sommet[$i][$j])){
		if($sommet[$i][$j] eq $sommet[$imean][$jmean] and $i!=$icom){
			$v=0;
			while(defined($sommet[$i][$v])){
				if($sommet[$i][$v] eq $sommet[$icom][$jcom]){
					printf OUTPUT "$i, ";
					push(@done,$sommet[$icom][$jcom]);
					$iout=$i;
					$jout=$v;
					$j=0;
					goto SPAIN;
				}
				$v++;
			}
		}
		$j++;
	}
```

```
                  $i++;
            }
            $r++;
      }
      $p=1;

      printf OUTPUT"]
                        }
                        }";
}
#########################################################################################################
# on va maintenant dessiner les proéminence (code "PRO")

$flag=0;
for($i1=0;$i1<@nomface;$i1++){
      if($flag==0){
            for($j1=0;$j1<@repface;$j1++){
                  if($repface[$j1] eq $nomface[$i1]){
                        $flag=1;
                        $a=$plan[$j1][0];
                        $b=$plan[$j1][1];
                        $c=$plan[$j1][2];
                        $d=-$plan[$j1][3];
                  }
            }
      }
      if($nomface[$i1] eq "end"){
            $flag=0;
      }
      # Gauss Jordan pour retrouver les points d'intersection avec la face de base de la proéminence
      # Recherche d'un point appartenant à une droite perpendiculaire à un plan et au dit-plan.
      # on connait équation cartésienne du plan et équation paramétrique de la droite (=vecteur normal directeur du
plan)
      # système de 4 équation à 4 inconnues
      # x = x0 + r*a
      # y = y0 + r*b
      # z = z0 + r*c
      # ax + by + cz = d
```

```
	$a[0] = $a;
	$a[1] = $b;
	$a[2] = $c;
	$a[3] = 0;
	$a[4] = 1;
	$a[5] = 0;
	$a[6] = 0;
	$a[7] = -$a;
	$a[8] = 0;
	$a[9] = 1;
	$a[10] = 0;
	$a[11] = -$b;
	$a[12] = 0;
	$a[13] = 0;
	$a[14] = 1;
	$a[15] = -$c;

	$b[0] = $d;
	$b[1] = $xpro[$i1];
	$b[2] = $ypro[$i1];
	$b[3] = $zpro[$i1];
	$n=4;
	$p=1;
	# Gauss-Jordan
	# attention: @a=@z
	$sing=0;

	for ($j=0;$j<$n;$j++){
		$max=0;
		for($i=$j;$i<$n;$i++){
			if($max<abs($a[$i*$n+$j])){
				$max=abs($a[$i*$n+$j]);
				$pivot=$i;
			}
		}
# la valeur du pivot est trouvée
		if($max==0){
			$sing=1;
```

```
                  printf "Problème: impossibilité de résoudre la proeminence concernant $nomface[$i1] (matrice
singulière)";
                  last;
            }
            else{

                  for ($jam=0;$jam<$n;$jam++){
                        $tempo=$a[$pivot*$n+$jam];
                        $a[$pivot*$n+$jam]=$a[$j*$n+$jam];
                        $a[$j*$n+$jam]=$tempo;
                  }

                  for ($jam=0;$jam<$p;$jam++){
                        $tempo=$b[$pivot*$p+$jam];
                        $b[$pivot*$p+$jam]=$b[$j*$p+$jam];
                        $b[$j*$p+$jam]=$tempo;
                  }

# les lignes sont interverties pour la matrice a et b

                  $ajj=$a[$j*$n+$j];
                  for ($jam=0;$jam<$n;$jam++){
                        $a[$j*$n+$jam]=$a[$j*$n+$jam]/$ajj;
                  }

                  for ($jam=0;$jam<$p;$jam++){

                        $b[$j*$p+$jam]=$b[$j*$p+$jam]/$ajj;
                  }

                  for($i=0;$i<$n;$i++){

                        if($i!=$j){
                              $aij=$a[$i*$n+$j];
                              for($jam=0;$jam<$n;$jam++){
                                    $a[$i*$n+$jam]=$a[$i*$n+$jam]-$aij*$a[$j*$n+$jam];
                              }
                              for($jam=0;$jam<$p;$jam++){
                                    $b[$i*$p+$jam]=$b[$i*$p+$jam]-$aij*$b[$j*$p+$jam];
```

```
                                }
                            }
                        }
                    }
                    $sing=0;

            }
            $xbpro[$i1]=$b[0];
            $ybpro[$i1]=$b[1];
            $zbpro[$i1]=$b[2];
# fin de Gauss Jordan
}

printf OUTPUT"
            Shape {
          appearance    Appearance {
            material    Material {
            diffuseColor 0 .5 .5
            specularColor .87 .25 .25
            ambientIntensity .157
            shininess .5
            }
          }
            geometry    IndexedFaceSet {
            solid FALSE
            coord Coordinate {
                  point [\n";
for($i=0;$i<@nomface;$i++){
            printf OUTPUT"$xpro[$i] $ypro[$i] $zpro[$i],\n$xbpro[$i] $ybpro[$i] $zbpro[$i],\n";
}
printf OUTPUT"]
                  }
                  coordIndex [\n";
$p=0;
for($i=2;$i<@nomface*2;$i=$i=$i+2){
            $h=$i-1;            # on établit les faces latérales
            $g=$i-2;
            $j=$i+1;
            printf OUTPUT "$g, $h, $j, $i, -1,\n";
```

```
      if($nomface[$i/2] eq "end"){  # on établit la face "toit"
            $g=$p;
            $h=$p+1;
            $j=$i+1;
            printf OUTPUT "$g, $h, $j, $i, -1,\n";
            for($u=$p;$u<=$i;$u=$u+2){
                  printf OUTPUT"$u, ";
            }
            printf OUTPUT"-1, \n";
            $p=$i+2;
            $i=$i+2;
      }
}

printf OUTPUT"]
                        }
                        }";

close(OUTPUT);
close(LOG_FILE);
```