Implémentation d'un algorithme de généralisation automatique de bâtiments 3D

POUR L'AFFAIRE

Terra Data-1

Préparé par :
Laboratoire COGIT - IGN
2/4 avenue Pasteur
94165 Saint-Mandé Cedex
http://recherche.ign.fr/cogit/

Document préparé grâce aux contributions de :

Aurélien VELTEN

Guillaume TOUYA

Laboratoire COGIT - IGN

_____

EVOLUTIONS

INDICE DE REVISION	DESCRIPTION
A	
B	
C	
D	
E	
F	

Ind. + Date	20/06/08	A	B	C	D	E	F
Etabli par :							
IGN	M. Brasebin						
Vérifié par :							
IGN	A. Ruas						
Approuvé par :							
IGN	A. Ruas						
Validé par :							
THALES	P. Peyronnet						

TABLE DES MATIERES

TABLE DES ILLUSTRATIONS

1. GENERALITES

1.1. OBJET DU DOCUMENT

Ce document a été rédigé dans le cadre du WP 4.1.1 du projet Terra Numerica. Il décrit l'état d'avancement du développement de l'algorithme de généralisation 3D. Ce dossier sera accompagné d'un livrable alpha de cet algorithme développé par le COGIT. Ce document reprendra, tout d'abord, l'état de l'art qui a permis de choisir un algorithme de simplification. Cet algorithme sera ensuite présenté étape par étape. Les premiers résultats obtenus grâce à cet algorithme seront présentés, ce qui permettra d'aborder le sujet des améliorations possibles. Enfin, un descriptif du livrable accompagnant ce document, conclura ce document.

Ce document pourra être consultable par les financeurs ou l'association (Cap Digital) pour information.

2. DOCUMENTS DE REFERENCES

2.1. DOCUMENTS EN REFERENCE

[1] **Badard T., Braun A. (2004),** *OXYGENE: A Platform for the Development of Interoperable Geographic Applications and Web Services*. In proceedings of the 15th International Workshop on Database and Expert Systems Applications, IEEE Press, August 30 – September 03, 2004, Zaragoza, Spain, pp. 888-892

[2] **Barrault M., Regnauld N., Duchêne C., Haire K., Baeijs C., Demazeau Y., Hardy P., Mackaness W., Ruas A., Weibel R. (2001)**, *Integrating multi-agent, object-oriented, and algorithmic techniques for improved automated map generalisation*, /20th international conference of cartography, ICA, Beijing, vol. 3, 2001, p. 2110-2116.

[3] **CityGML (2007)** *Exchange and Storage of Virtual 3D City Models.* Homepage of the project available at http://www.citygml.org/.

[4] **Forberg A. (2004),** *Simplification of 3D building data*. In proceedings of the 6th CA Workshop on progress in automated map generalisation, Leicester, 2004.

[5] **Kada M. (2005),** *3D Building Generalization.* in proceedings of ICA2005, the International Cartography Association Conference, La Coruña (Spain), 2005.

[6] **Kada M. (2007),** *3D Building Generalisation by Roof Simplification and Typification.* in proceedings of ICC2007, the International Cartography Association Conference, Moscow (Russia), 2007.

[7] **Lal J., Meng L. (2004),** *3D building recognition using artificial neural network.* In proceedings of the 6th CA Workshop on progress in automated map generalisation, Leicester, 2004.

[8] **Meng L., Forberg A. (2007),** *3D Building Generalisation*. Mackaness W., Ruas A., Sarjakoski T. (eds) : The Generalisation of Geographic Information : Models and Applications. Elsevier (2007). Chapter 11.

[9] **Poupeau B., Bonin O. (2006),** Cristage: a 3D GIS with a logical crystallographic layer to enable complex analyses. In: Proceedings of 3D GeoInfo'06, Kuala-Lumpur, Malaysia.

[10] **Poupeau B. (2007),** *A crystallographic approach to simplify 3D building*. in proceedings of ICC2007, the International Cartography Association Conference, Moscow (Russia), 2007.

[11] **Thiemann F., Sester M. (2004),** *Segmentation of Buildings for 3D-Generalisation*. In proceedings of the 6th CA Workshop on progress in automated map generalisation, Leicester, 2004.

Mise en forme : Puces et numéros

~~2.1~~2.2. NORMES APPLICABLES

?

2.3. GLOSSAIRE

Glossaire des différentes abréviations utilisées dans ce document.

COGIT	Conception Objet et Généralisation de l'Information Topographique
IGN	Institut Géographique National

3. RAPPEL DE L'ETAT DE L'ART

Dans le domaine des données géographiques et de la cartographie, la généralisation est une opération qui vise à réduire le niveau de détail d'une base de données géographiques ou d'une carte tout en maintenant les caractéristiques principales et en assurant le respect des règles de lisibilité dans le cas d'une carte (figure 1).

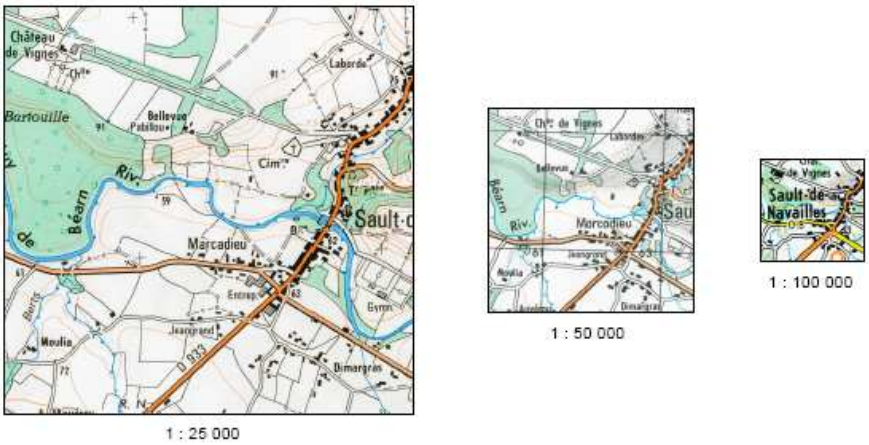

Figure 1 : Extraits de cartes IGN à différentes échelles sur la région de Sault-de-Navailles (64). La généralisation a été indispensable à chaque changement d'échelle.

Les bâtiments étant un élément fondamental de l'information topographique, de nombreux travaux ont traité de l'automatisation de la généralisation de ces objets géographiques. Les travaux présentés en figure 2 en sont un exemple.

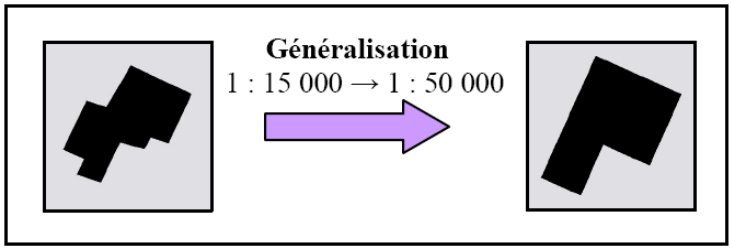

Figure 2 : Exemple de généralisation de bâtiment avec le processus AGENT (Barrault et al, 2001) [2].

Avec l'apparition des données géographiques en 3D, le problème de la généralisation est bien évidemment réapparu notamment pour des questions de volumes de données ou de vitesse de visualisation. Les premiers travaux traitant de la généralisation des bâtiments en 3 dimensions sont apparus en 2004 avec trois travaux et surtout approches différentes. La première méthode traite de la segmentation des bâtiments 3D (Thiemann & Sester, 2004)[11]. Elle part du principe que pour généraliser un bâtiment, il faut modéliser précisément ses caractéristiques. Ainsi, un bâtiment est modélisé en parties "atomiques" qui sont assemblées par opérations booléennes : des murs, des toits, des fenêtres, des porches ou des cheminées (figure 3). Le bâtiment est ensuite généralisé en simplifiant ou supprimant ces parties atomiques : par exemple, on enlève les fenêtres et la cheminée et on simplifie la forme du toit. Cette méthode semble appropriée dans le cas de données initiales très détaillées comme dans la figure 3.

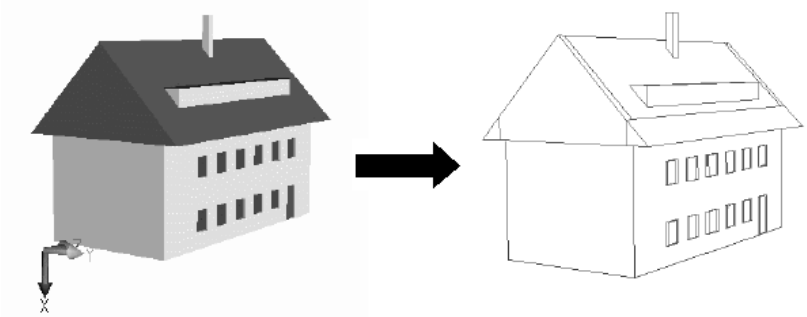

Figure 3 : Algorithme de segmentation d'un bâtiment 3D en parties atomiques (Thiemann & Sester, 2004) [11].

Le travail de (Forberg, 2004) [4] est une autre approche de la généralisation de bâtiments 3D. Dans ce cas, il s'agit plus simplement de simplifier la forme du bâtiment. La méthode conçue appelée "Parallel shift" est basée sur la réunion de deux concepts mathématiques : la morphologie mathématique et la courbure de l'espace. Il s'agit d'une méthode de recherche incrémentale de faces parallèles. Afin de donner de meilleurs résultats, un prétraitement d'équarrissage du bâtiment doit être effectué. Cet algorithme ne traite pas les toitures comme on peut le voir sur les résultats en figure 4.

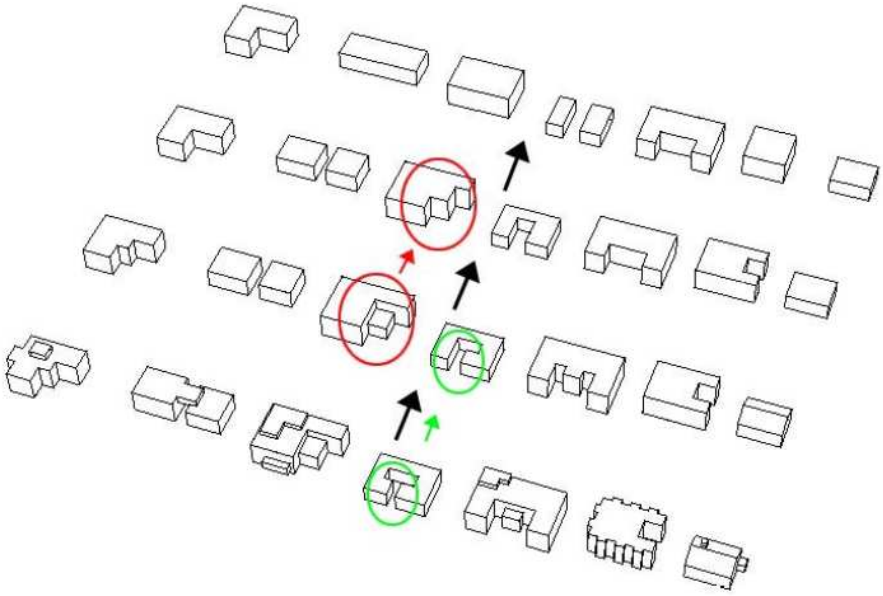

Figure 4 : Quelques résultats de l'algorithme de simplification de Forberg (Forberg, 2004) [4].

Une troisième approche est apparue la même année mais elle était plus à l'état théorique qu'expérimental. (Lal & Meng, 2004) [7] propose d'aller plus loin et de généraliser les bâtiments en 3D en prenant en compte le contexte comme cela se fait en 2D. Cette méthode cherche à détecter les caractéristiques spatiales des bâtiments individuels mais aussi les relations spatiales entre les bâtiments voisins ainsi que l'existence de groupes (bâtiments de même forme ou alignés par exemple). Une classification des formes courantes de bâtiments a été mise au point (figure 5) ainsi que des algorithmes pour reconnaître de quelle forme un bâtiment quelconque est proche. La simplification est alors effectuée en donnant au bâtiment la forme idéale simplifiée à laquelle il ressemble.

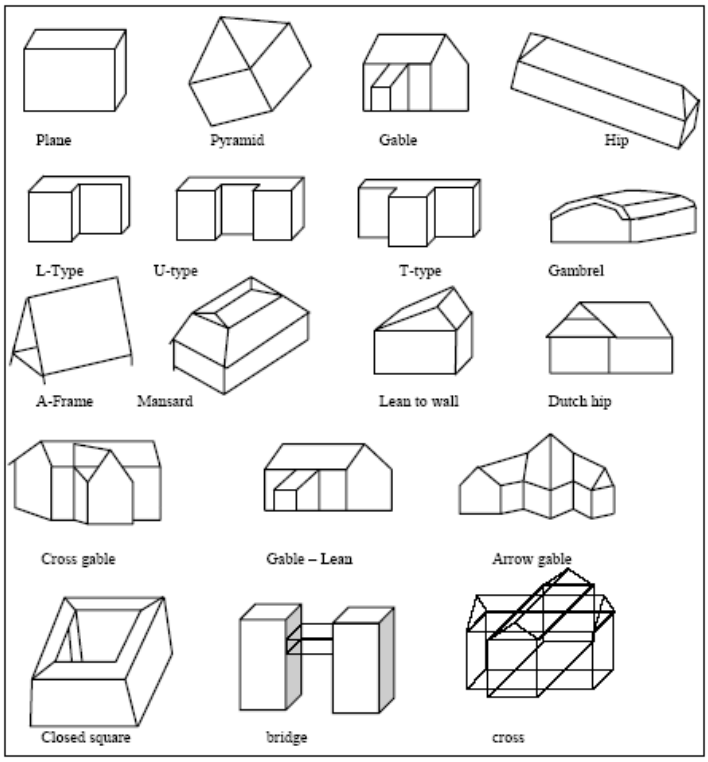

Figure 5 : Classification de différentes formes typiques de bâtiments.

Une autre approche a été développée au laboratoire COGIT par Benoît Poupeau (Poupeau, 2007) [10] pour simplifier un bâtiment. Elle s'appuie sur une modélisation cristallographique du bâtiment. Cette modélisation a pour but de gérer des relations spatiales complexes des données géographiques 3D comme le fait qu'un bâtiment se trouve au-dessus d'une faille ou de certaines couches géologiques par exemple. Mais il est apparu que cette modélisation pouvait aussi permettre de simplifier la forme des bâtiments. En effet, la modélisation est basée sur la détection des diverses symétries d'un polyèdre. Par exemple, quand des façades sont presque symétriques, la simplification consiste à les rendre vraiment symétriques. Des résultats obtenus avec cette méthode sont présentés en figure 6. Cette méthode n'a pas été retenue car elle n'est pas encore robuste et reste lente en temps de calcul (la simplification n'étant pas l'objectif de ce travail de modélisation, l'algorithme n'a pas fait l'objet d'assez de mise au point).

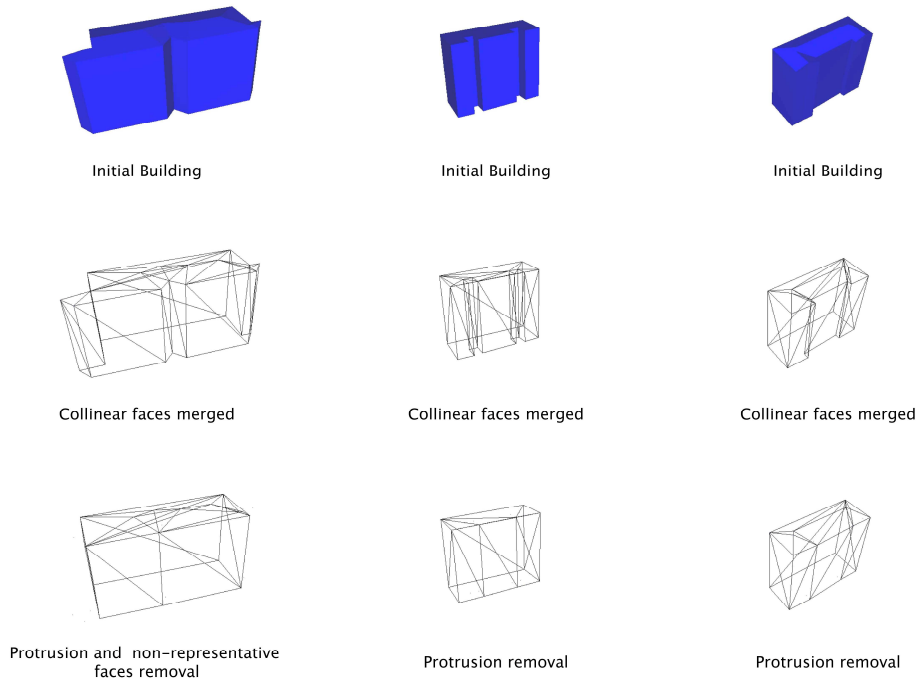

Figure 6 : Exemples de simplification avec l'algorithme de Poupeau (Poupeau, 2007) [10].

Une dernière approche a été développée, assez proche de celle Forberg, qui consiste en une simplification importante de la forme du bâtiment, y compris le toit (Kada, 2005, 2007) [5] [6]. Cette approche, dont les différentes étapes sont présentées dans la figure ci-dessous, consiste à rechercher des plans approximant plus ou moins (selon le degré de généralisation) proche des différentes façades d'un bâtiment en 3D. Ensuite l'espace est découpé par ces plans et les parties de l'espace contenant principalement le bâtiment initial sont gardées comme approximation du bâtiment. Les toitures sont ensuite simplifiées de la même manière en recherchant des toitures approximantes. Toutes les parties sont finalement recollées pour donner le bâtiment généralisé. Cet algorithme a l'avantage d'être rapide et de simplifier fortement les bâtiments.

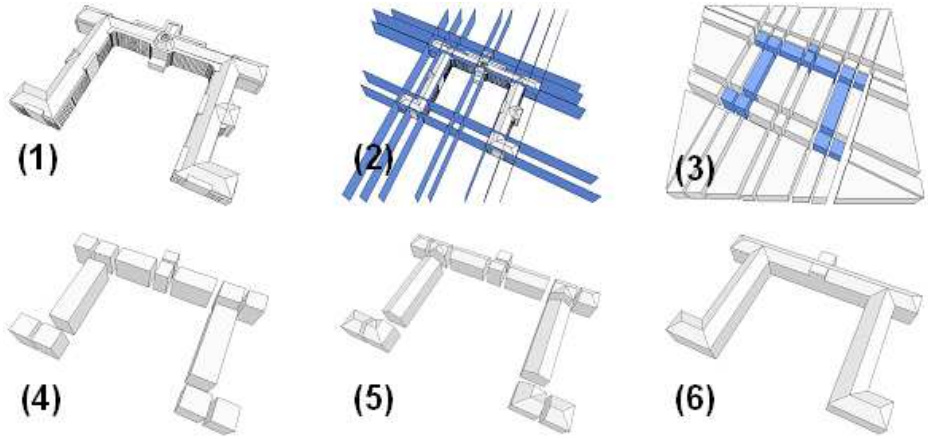

Figure 7 : Différentes étapes de la simplification de Kada (Kada 2007) [10]

Enfin, il est intéressant de noter l'existence du projet Open Source CityGML qui est un projet d'extension de GML pour modéliser les villes en 3 dimensions. Une partie de ce modèle concerne les bâtiments avec une modélisation qui autorise plusieurs niveaux de détails ce qui est très intéressant pour la généralisation. La figure 8 montre des exemples de données au format CityGML à différents niveaux de détails.

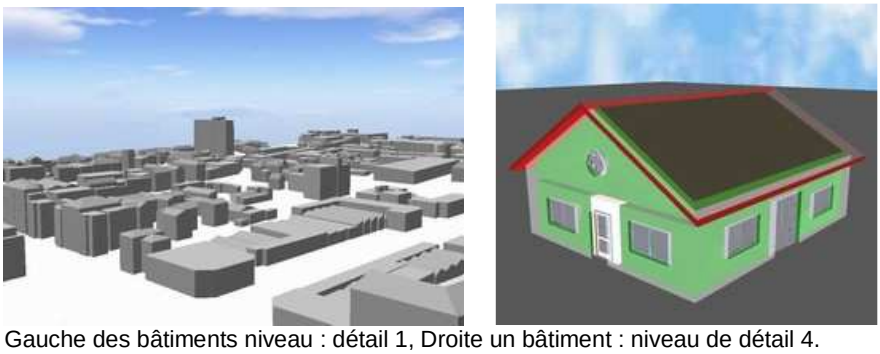

Figure 8 : Gauche des bâtiments niveau : détail 1, Droite un bâtiment : niveau de détail 4.

4. DESCRIPTION DE L'ALGORITHME CHOISI

Dans le cadre du projet Terra Numerica, il semblait plus intéressant de tester un algorithme permettant une forte simplification des bâtiments en 3D. Le choix s'est porté sur l'algorithme de Martin Kada car il paraissait simple, rapide et efficace.

Aurélien Velten, a implémenté un algorithme s'inspirant de celui de Martin Kada. Le code a ensuite été repris et amélioré par Mickael Brasebin.

Cet algorithme est une simplification de l'algorithme de Kada. L'algorithme reprend le principe des zones tampons proposé par Martin Kada, mais la forme simplifiée est calculée essentiellement à partir de projections en deux dimensions du bâtiment. Cela a été fait dans un souci d'efficacité de calcul : les opérations sur les projections sont beaucoup plus rapides à effectuer que les opérations sur les objets 3D (notamment l'intersection, l'union, le calcul de zones tampons …).

Voici les différentes étapes de l'algorithme :

Etape 1 : *Elimination des faces inclinées*

La 1ère étape consiste à ne garder que les murs. Pour cela on détecte les faces ayant un angle quasi-droit avec l'horizontale.

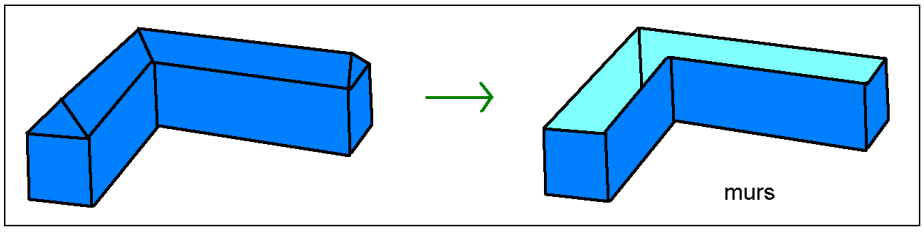

Figure 9 : Elimination des faces non verticales

Etape 2 : *Détection des différentes « coupes en z »*

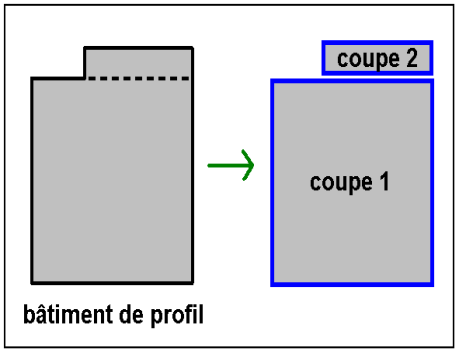

Figure 10 : Détection des coupes en Z

Ensuite une fois que les plans inclinés ont été éliminés, on détecte les différentes « coupes en z » du bâtiment, afin d'isoler les différents blocs. Pour cela on regroupe les murs par leur point d'altitude maximale. Pour décider à quelle coupe une face appartient un seuil d'altitude a été déterminé, par exemple 3 mètres. Ainsi, si deux faces ont une différence d'altitudes maximales inférieure à 3 mètres, elles feront partie de la même coupe et seront au même niveau d'altitude dans le bâtiment obtenu en sortie de l'algorithme.

Etape 3 : *Projection des murs pour chaque coupe*

Pour chaque « coupe en z », l'ensemble des murs est projeté sur le plancher (plan de z minimal) de la coupe en question. On obtient donc un ensemble de segments, qui ne forme pas obligatoirement un cycle fermé.

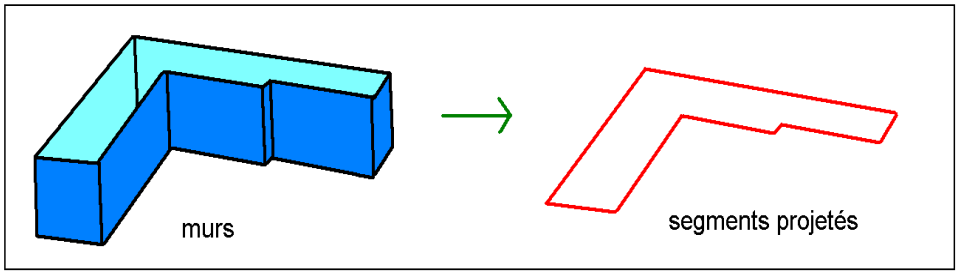

Figure 11 : Projection des murs

Etape 4 : *Détection des cycles pour chaque coupe*

Pour chaque « coupe en z » on détecte les différents cycles à partir des segments projetés. Un cycle est un ensemble de segments formant un circuit fermé. Par exemple, si deux tours ont la même altitude, elles feront partie de la même « coupe en z » mais formeront deux cycles distincts :

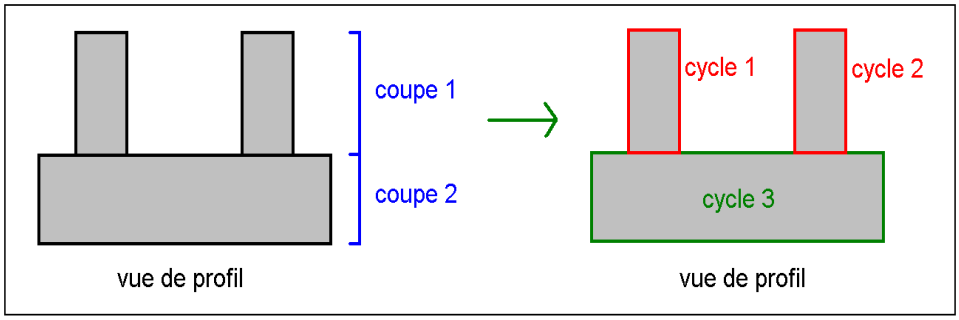

Figure 12 : Détection des cycles

Etape 5 : *Construction des buffers initiaux*

Pour chaque cycle de chaque « coupe en z », un « buffer initial » est créé englobant chaque segment projeté. Les buffers sont créés sous la forme de rectangles ayant pour longueur la longueur du segment, et pour largeur une distance fixée par l'utilisateur (on propose 0,6 mètres).

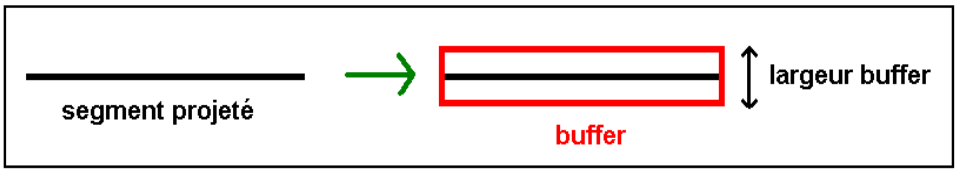

Figure 13 : Création d'un buffer

Etape 6 : *Fusion des buffers*

Il s'agit de la phase la plus importante, les « façades-approximantes » y sont détectées. Les façades-approximantes sont les façades qui remplaceront, dans le bâtiment simplifié, les façades du bâtiment initial.
Pour les déterminer, on essaye de fusionner les buffers initiaux par paire : on construit une boîte englobante rectangulaire d'aire minimale contenant les deux buffers, et si la largeur est inférieure à un seuil de fusion (on propose 1.3 m), la fusion est validée, les deux buffers sont effacés, puis remplacés par le nouveau buffer : il s'agit de la boîte englobante. Ce nouveau buffer sera aussi testé avec les autres buffers. Cette étape s'arrête lorsqu'il n'y a plus de fusion possible.
On obtient alors une liste de « buffers finaux », qui représentent la trace au sol des façades-approximantes du bâtiment.

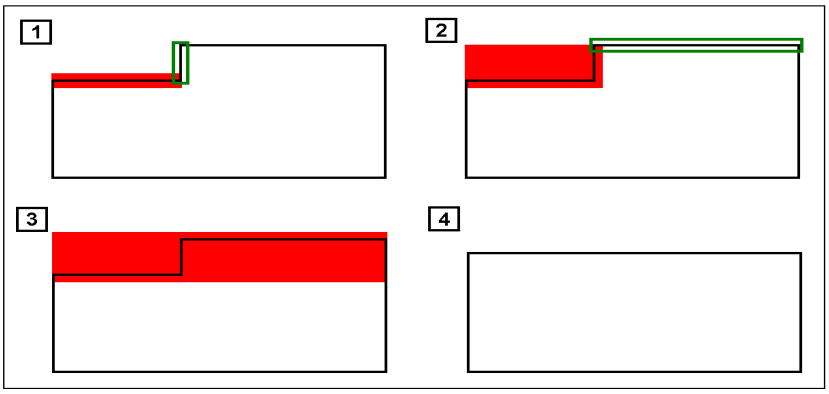

Figure 14 : Exemple de fusions de buffers

Dans cette figure, en 1, les buffers rouges et verts sont testés. La largeur de la boite englobante est inférieure au seuil : la fusion est validée, nous allons tester le nouveau buffer rouge sur un autre buffer vert (étape 2). La fusion est également validée, il en résulte donc l'étape 3. L'étape 4 présente le résultat après calcul des segments-approximants.

Après avoir testé l'algorithme, il s'est avéré que certaines fusions ne se faisaient pas « logiquement ». Une étape de « pré-fusion » a été ajoutée pour résoudre un type d'erreur (présenté dans la figure suivante). Cette « pré-fusion » consiste à effectuer des fusions avec un coefficient de valeur inférieure à celle utilisée lors de la fusion des buffers (Par exemple ½ du seuil de fusion). Cela permet de fusionner en priorité les murs alignés entre eux.

Par exemple, dans les données-tests, certains murs étaient composés de deux faces parfaitement adjacentes et alignées. Grâce à cette « pré-fusion » ces deux faces se retrouvaient fusionnées ensemble, sinon il aurait été possible que l'un de ces deux murs fusionne d'abord avec un autre mur non aligné avec les deux autres.

Voici un exemple où la pré-fusion des buffers a été utile :

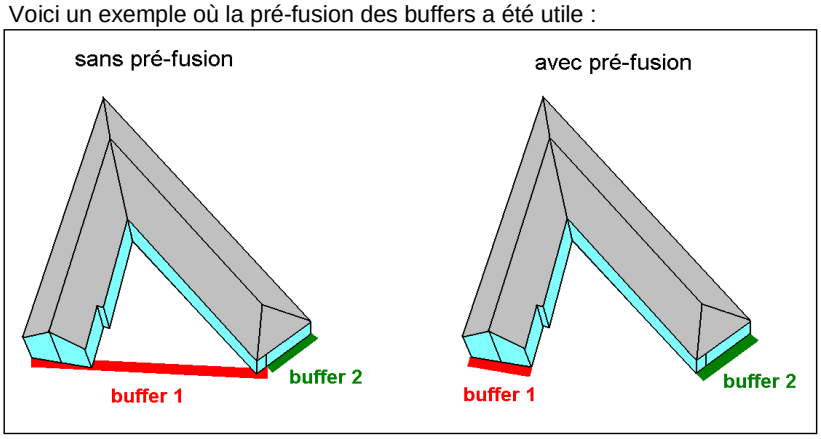

Figure 15 : Exemple de l'utilité de la pré-fusion

Etape 7 : *Calcul des segments-approximants*

Une fois les buffers finaux obtenus, il est nécessaire de calculer les « segments-approximants » ayant pour extrémités les milieux des largeurs des buffers finaux. En quelque sorte c'est le raisonnement inverse de l'étape 5 qui consiste à créer des buffers à partir de segments.

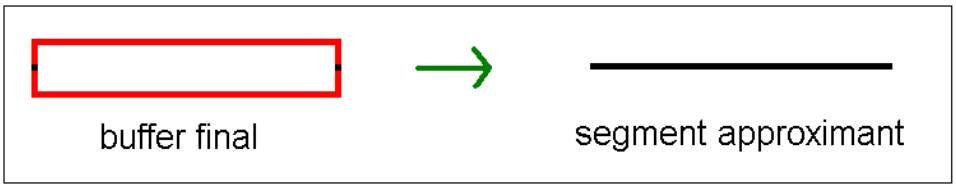

Figure 16 : Transformation d'un buffer en segment

Etape 8 : *Recollage des segments-approximants*

Une fois les segments-approximants obtenus, il se peut que certains segments ne soient pas adjacents du fait de la fusion des buffers qui engendre des déplacements. Il est donc important de recoller ces segments entre eux afin d'éviter les trous dans le bâtiment final. Il faut au préalable ordonner les segments-approximants (dans le sens trigonométrique par exemple), puis réajuster les extrémités afin qu'ils soient tous adjacents deux à deux.

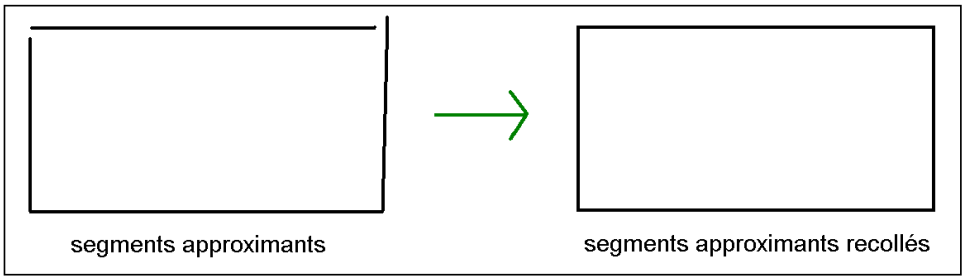

Figure 17: Recollement des segments

Etape 9 : *Construction des murs-approximants*

Toujours pour chaque cycle de chaque « coupe en z », on construit les murs à partir des segments-approximants recollés : pour chaque segment-approximant, on reconstruit une façade qui a pour base ce segment et pour hauteur l'altitude maximale de chaque coupe.

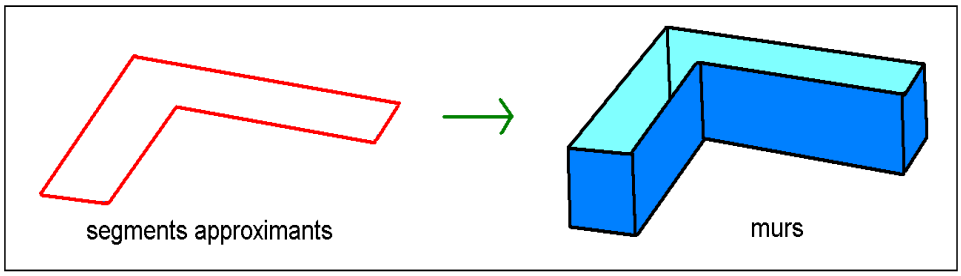

Figure 18 : Reconstruction d'un mur à partir de segments

Etape 10 : *Construction du toit et du plancher*

La construction du toit se fait à partir de la liste des points des segments finaux à l'altitude maximale de la coupe (et à altitude minimale pour le plancher). La liste de point permet de former un polygone couvrant le bâtiment.

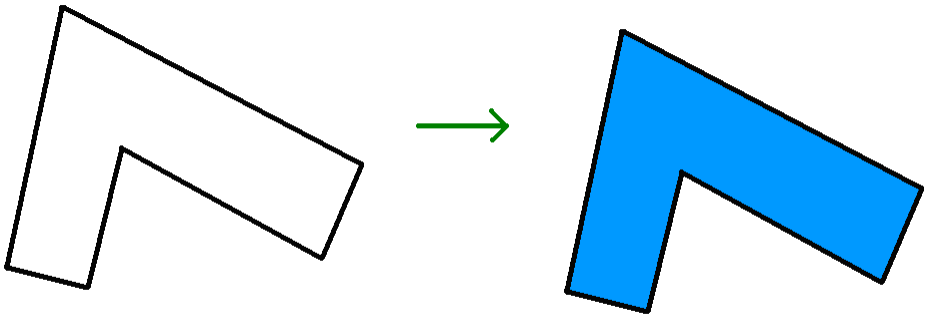

Figure 19 : Construction d'un toit et d'un plancher

Etape 11 : *Fusion des cycles*

La dernière étape consiste à fusionner tous les cycles de toutes les « coupes en z » en un seul objet 3D. Le bâtiment simplifié est alors obtenu, prêt à être affiché

5. IMPLEMENTATION ET RESULTATS

1. MODELE DE DONNEES ET IMPLANTATION

La géométrie des solides à été modélisée par le modèle BRep (Boundary Representation). Ce modèle propose de décomposer un objet (corps) en différentes surfaces qui elles-mêmes sont définies par des arcs composés de sommets.
Puisque la notion d'arc n'est jamais utilisée lors de la phase de simplification, le modèle de données suivant a été utilisé lors de l'implémentation :

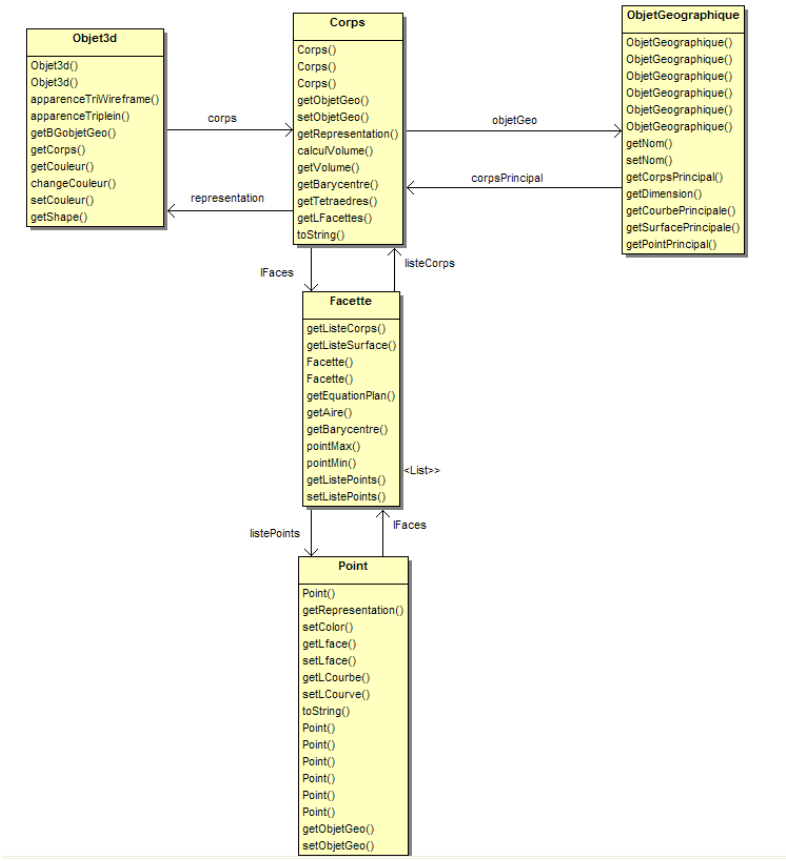

Figure 20 : Modélisation des données de 3D

Un bâtiment est considéré comme un objet de la classe « ObjetGéographique » et possède ainsi un corps.
Cet algorithme a été implanté dans la surcouche 3D Cristage de la plateforme SIG GéOxygène développée au COGIT (Badard & Braun, 2004) [1]. Cettte plateforme a été développée en Java et nécessite une machine virtuelle Java de version 1.5 ou supérieure.

Pour pouvoir utiliser la surcouche Cristage, il est nécessaire, outre l'installation de GéOxygène, d'installer Java3D Version 1.51 (pour l'affichage) et d'utiliser 2 dll : tegendll.dll et trianguledll.dll (elles permettent des opérations de tétraèdreisation et de triangulation).

2. LES PARAMETRES

Cet algorithme est paramétrable à travers 4 paramètres (entre parenthèse est indiqué le paramètre par défaut utilisé pour les bâtiments de Bati3D):

- Seuil de coupe : ce seuil permet de regrouper les faces du bâtiment en coupe selon leur altitude maximale (3m)
- Seuil de fusion : la fusion à lieu si la boîte englobante de 2 buffers à une largeur inférieure à ce seuil (1.3m)
- Taille du buffer : il s'agit de la largeur des buffers initiaux (0.6m)
- Tolérance d'angle : il s'agit de l'angle maximal par rapport à la verticale que peut former une face pour être considérée comme un mur (2°).

Il faut savoir qu'un jeu de paramètres n'est pas optimal pour un lot de données. Les paramètres optimaux doivent presqu'être définis bâtiment par bâtiment. Les valeurs par défaut proposée ci-dessus, sont celles qui ont permis les meilleurs résultats moyens.

Le seuil de fusion est le paramètre le plus important : il définit le degré de simplification du bâtiment. Une valeur trop élevée du seuil provoquerait la fusion de tous les buffers et il ne résulterait de la simplification qu'un seul mur.

La taille du buffer doit toujours être strictement inférieure au seuil de fusion. Si ce n'est pas le cas, aucune simplification ne sera effectuée (pas de fusions de buffer).
Le seuil de coupe ne doit pas avoir une valeur trop faible. Si cette valeur est vraiment faible (inférieure à 1.5m), il est possible que l'on obtienne de nombreux cycles non fermés et que cela soit mal géré par l'algorithme.

3. RESULTATS

→ Temps de calcul

La force de cet algorithme est sa rapidité. Pour simplifier un bâtiment ayant 80 faces (ce qui représente un nombre de faces élevé), 380 ms sont nécessaires. Pour les bâtiments Bati 3D, le temps de calcul est toujours inférieur à 100 ms.

Ces mesures ont été effectuées sur un Intel Xeon 2.33 GHZ avec 2 GO de mémoire.

→ Application sur des exemples de bâtiments Bati3D

- Exemple 1

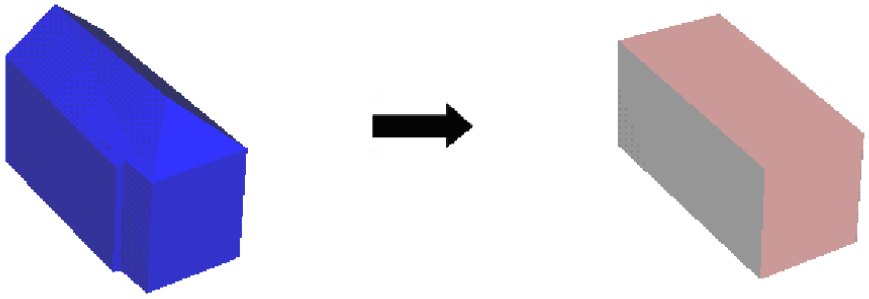

Figure 21 : Bâtiment avec décrochement

Ce bâtiment est de forme relativement simple. Il ne comporte qu'un léger décrochement sur un mur. L'algorithme élimine ce décrochement ainsi que la toiture. Le résultat ne peut être plus simple : il s'agit d'un parallélépipède. Ce bâtiment ne comporte qu'une seule « coupe en z » et un seul cycle. Pour ce genre de bâtiments très simplistes l'algorithme fonctionne sans problèmes. C'est un cas très courant parmi les bâtiments testés.

- Exemple 2

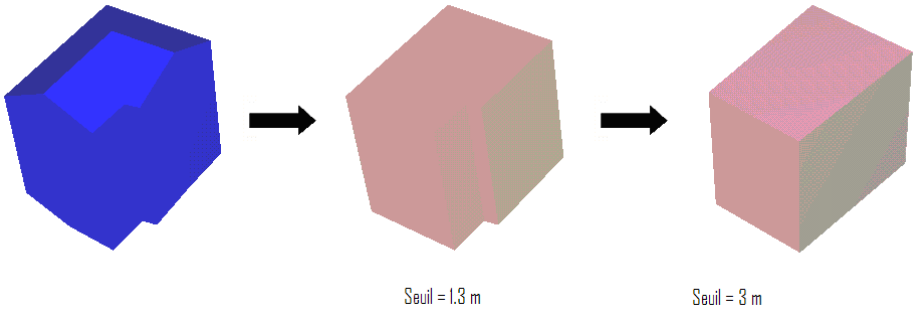

Figure 22 : Bâtiment avec décrochement (2)

Dans cet exemple le bâtiment est très similaire au bâtiment précédent, mais on constate que l'algorithme n'élimine pas le décrochement en utilisant la valeur de 1.3m pour le seuil de fusion. En le réglant à 3 m par exemple, le décrochement est alors éliminé. Cet exemple montre l'importance du choix des valeurs des paramètres.

- Exemple 3

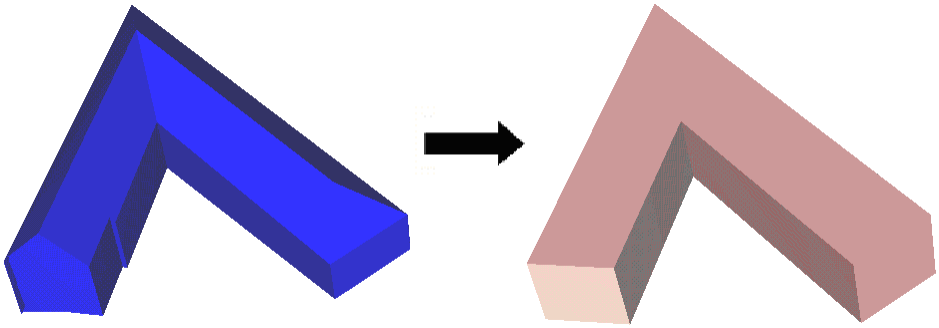

Figure 23 : Bâtiment concave avec décrochement

Voici un exemple avec un bâtiment non convexe qui possède un petit décrochement. Ce bâtiment se simplifie correctement.

- Exemple 4

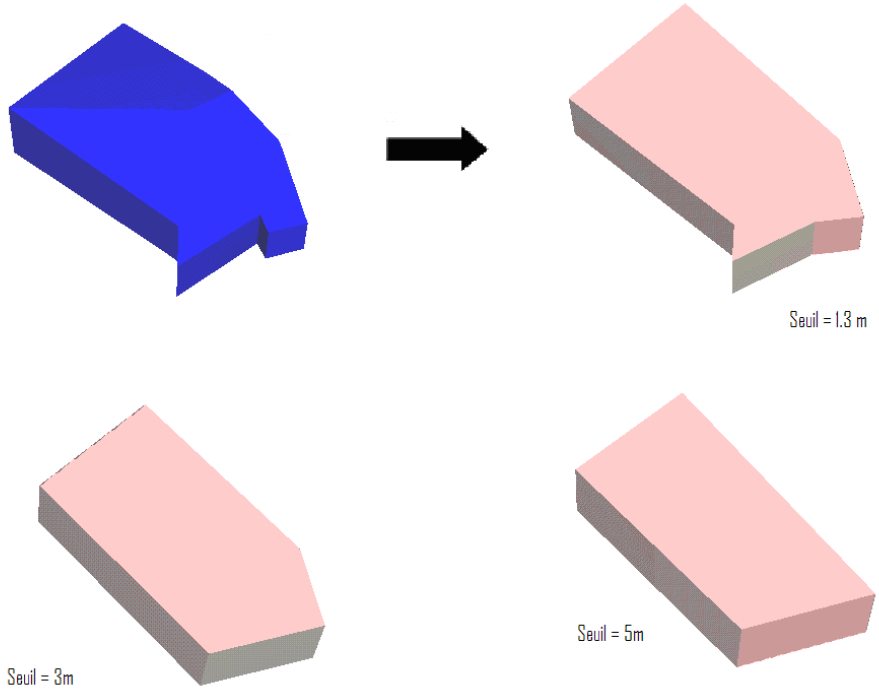

Figure 24 : Evolution en fonction du seuil de fusion

Dans cet exemple les décrochements ne se limitent plus à des faces parallèles comme sur les exemples précédents. Là encore le paramètre seuil est très important : il décide du degré de simplification.

- Exemple 5

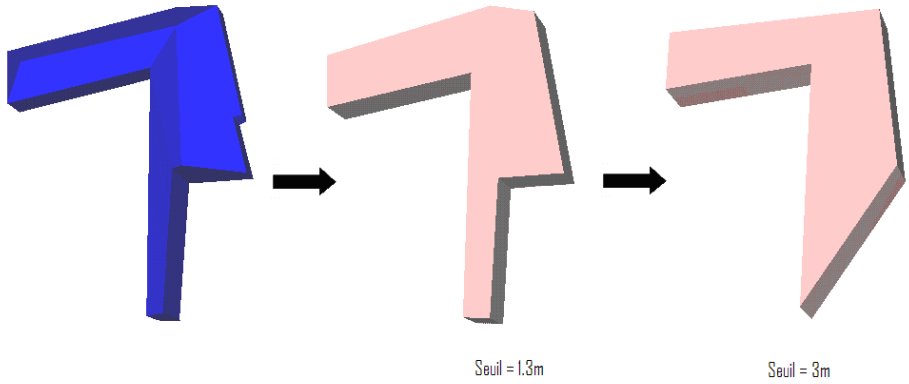

Figure 25 : Evolution en fonction du seuil de fusion (2)

Cet autre exemple illustre également la diminution du nombre de murs en fonction du seuil de fusion.

- Exemple 6

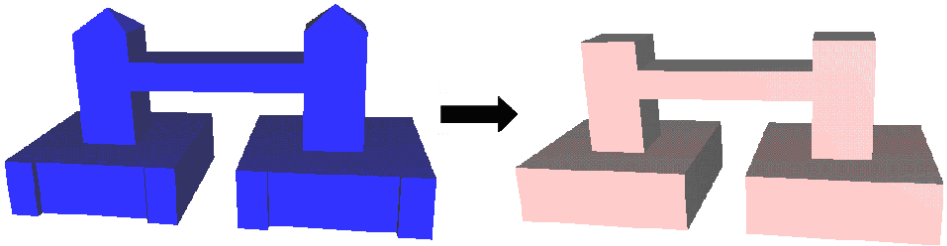

Figure 26 : Simplification avec coupes en Z

Cet exemple illustre l'intérêt de la division du bâtiment en « coupes en z » et en cycles. En effet, sans ces divisions, le résultat final aurait été le suivant :

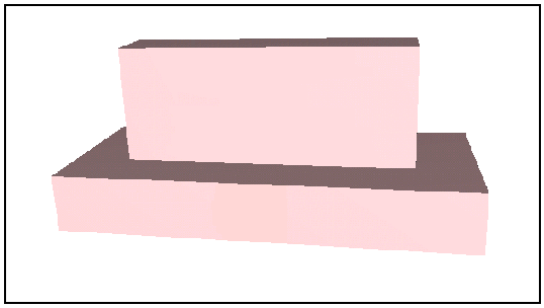

Figure 27 : Simplification sans coupe en Z

L'étape de détection des « coupes en z » crée trois blocs : les deux grandes parties au sol forment le premier bloc, les deux tours le deuxième bloc, et la passerelle centrale le troisième bloc. Ensuite la détection de cycles permet de séparer les deux parties au sol et les deux tours au sein d'un même bloc. On obtient donc au final cinq cycles, simplifiés séparément puis regroupés par la suite pour obtenir le bâtiment généralisé.

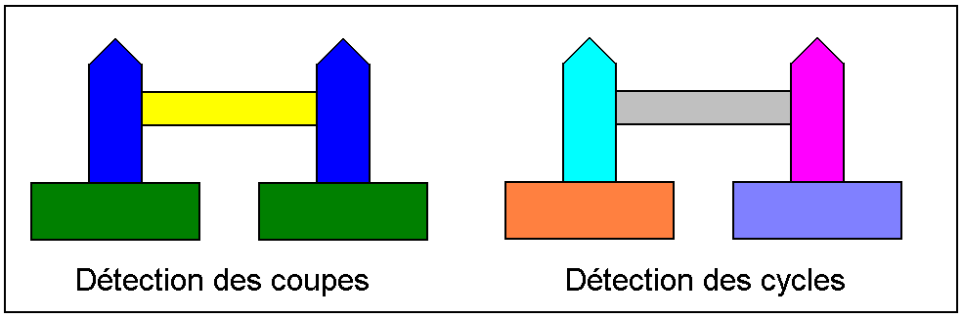

Figure 28 : Détection des coupes et des cycles

4. POINTS FORTS DE L'ALGORITHME

Le premier point fort est sa rapidité d'exécution. Cela est dû au fait qu'il ne réalise que des opérations élémentaires sur des objets en 2 dimensions et qu'il utilise un modèle de données très simple.

L'algorithme permet différents niveaux de simplification en fonction des paramètres choisis. Cela permet d'avoir une simplification plus ou moins détaillée.

Au niveau de la simplification, comme cela a été montré au fil des exemples, l'algorithme élimine bien les décrochements. Cette capacité à éliminer les décrochements permet également d'obtenir des murs droits grâce à la fusion de buffer et l'opération qui transforme un buffer en mur.

Le dernier point fort de cet algorithme est sa capacité à produire des sommets de murs de même niveau. Cela est dû à la fabrication des murs à partir des segments-approximants. Avoir les sommets de murs au même niveau permet de produire des toits parfaitement plats ou de pouvoir supporter plus facilement une structure de toit.

5. POINTS FAIBLES ET PISTES D'EVOLUTIONS

5.1.1. Absence de toiture

Actuellement les toits sont tout simplement éliminés pour être remplacés par des toits plats. Comme le montre l'exemple ci-dessous, l'élimination du toit peut modifier suffisamment le bâtiment pour qu'on ne puisse plus le connaître.

La réflexion pour résoudre ce problème s'est orientée autour de 2 approches introduites par Kada : tenter de simplifier le toit en utilisant des plans approximants ou en utilisant une typologie des toits.

Figure 29 : Exemple d'un bâtiment avec un très grand toit : Généralisation non concluante.

5.1.2. Le choix des paramètres

Bien que le choix des paramètres influe sur le niveau de simplification, il n'est jamais aisé de trouver les valeurs permettant d'obtenir le niveau souhaité de simplification. Hormis, les quelques conseils listés précédemment, il n'y a actuellement aucun moyen automatique permettant de déterminer les valeurs optimales. Ce fait est classique en généralisation.

Il est envisagé d'introduire des mesures du bâtiment, comme par exemple la taille des décrochements, la largeur des différentes portions du bâtiment ou la hauteur des « étages », afin d'obtenir des valeurs pertinentes pour les paramètres.

5.1.3. La fusion des buffers

Les buffers fusionnent actuellement avec le buffer suivant qui vérifie les conditions nécessaires de fusion (intersection et largeur de boîte englobante inférieure au seuil de fusion). La fusion n'est donc pas toujours optimale. La pré-fusion a été implantée pour palier à ce problème en permettant de faire fusionner les voisins entre eux.

L'exemple suivant présente un bâtiment (créé manuellement) et les différentes simplifications selon la valeur du seuil de fusion. On voit que les différentes simplifications ne respectent pas l'orientation du bâtiment initial et que la simplification ne tend pas vers le carré que l'on aurait obtenu en simplifiant manuellement.

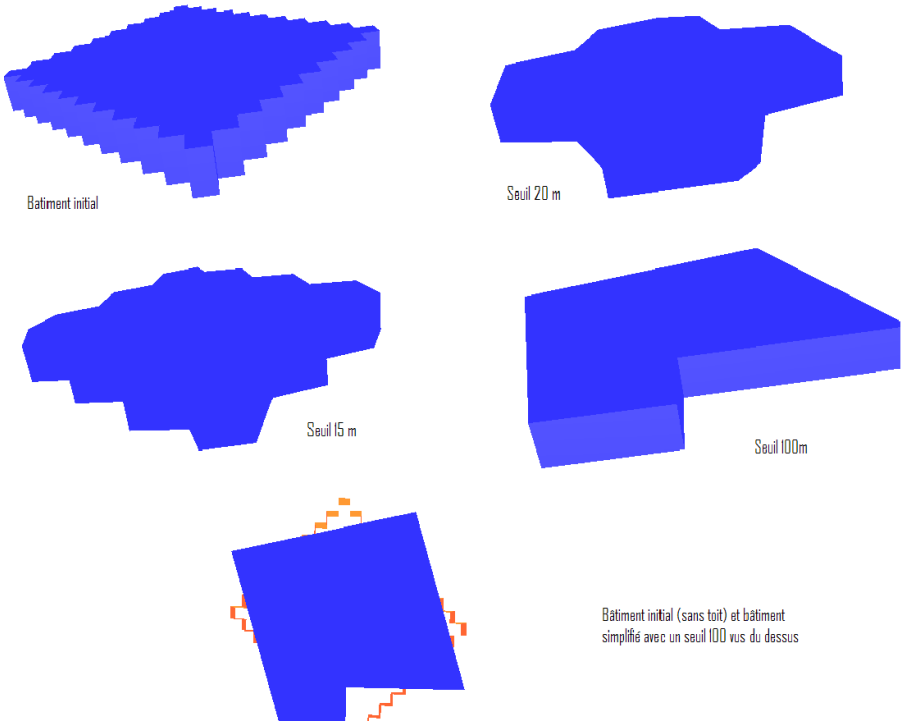

Figure 30 : Illustration du problème des fusions de buffers

Une première piste abordée était de tester les paramètres de fusion d'un buffer avec tous les autres buffers, puis, de le faire fusionner avec celui qui propose la plus petite largeur après fusion. Même si cela a apporté certaines améliorations, le problème n'a pas été entièrement corrigé car l'ordre de test des buffers influe toujours sur le résultat final.

Une seconde piste consisterait à faire fusionner les buffers par ordre décroissant de largeur de buffer après fusion. Ainsi, l'ordre de parcours des buffers n'aurait plus d'importance. Cela n'a pas encore été implémenté.

5.1.4. Les pans de murs non verticaux

Afin de ne pas traiter les toits, l'algorithme élimine les pans de murs non verticaux, or il existe des bâtiments possédant des murs non verticaux comme par exemple le Tokyo Big Sight. Lors de la simplification de tels bâtiments, les murs non-verticaux sont éliminés, ce qui provoque des trous dans le résultat de la simplification.

Figure 31: Bâtiment fictif s'inspirant du Tokyo Big Sight avant et après simplification

La résolution de ce problème n'a pas été abordée, il ne devrait être rencontré que très rarement. Il faudrait cependant donner la possibilité à l'algorithme de détecter ces cas pour les traiter indépendamment.

5.1.5. Problème de raccordement des segments-approximants

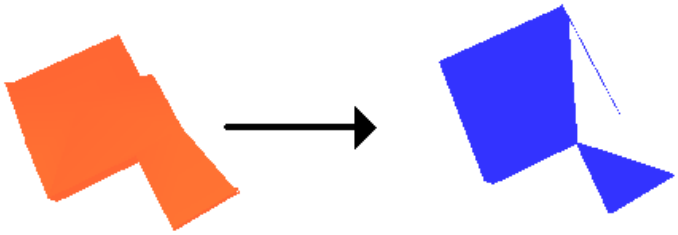

Figure 32 : Erreur de raccordement des buffers

Le raccordement s'effectue segment par segment, chaque segment est raccordé au segment le plus proche (sans utiliser 2 fois le même sommet). Il arrive que ce raccordement provoque des erreurs. Cela est dû au fait que l'ordre de raccordement a une importance. Une solution envisageable est de raccorder en essayant de faire des correspondances entre le bâtiment initial et le bâtiment simplifié. La réflexion n'est pas encore très avancée sur ce sujet.

5.1.6. Problème de cohérence avec le format BRep

La construction de toit ou de sol se réalise cycle après cycle. Ainsi, si un bâtiment nécessite plusieurs coupes pour être simplifié, il est possible que pour un bâtiment simplifié, le toit d'un cycle coïncide avec le sol d'un cycle supérieur. Le bâtiment possèderait des faces intérieures ce qui est contraire à la philosophie BREP, puisqu'elle consiste à ne représenter que les faces extérieures.

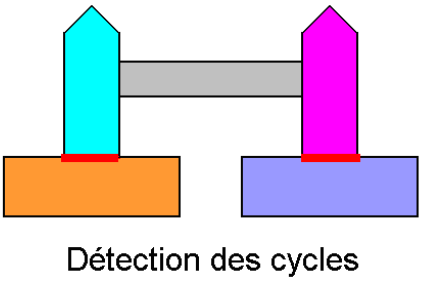

Figure 33 : Exemple de faces intérieures(en rouge)

Cela peut créer des problèmes s'il y a des calculs géométriques effectués sur la géométrie du bâtiment simplifié, mais n'interviendra pas dans l'affichage de bâtiments. Ce problème sera résolu lors de la mise en place de bibliothèques de correction de géométrie.

6. DESCRIPTION DE LA LIVRAISON

Comme il a été signalé auparavant, ce document s'accompagne d'une livraison. La livraison contient, outre la bibliothèque de l'algorithme, une application permettant d'afficher et de simplifier des bâtiments.

1. LE PACKAGE DE L'ALGORITHME DE SIMPLIFICATION

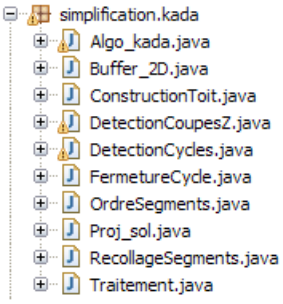

Figure 34 : Les classes contenus dans le package Kada

Le package Kada contient les classes permettant l'exécution de l'algorithme. Chaque classe permet la réalisation d'une étape spécifique de l'algorithme.

1. Algo_kada.java : est la classe principale de l'algorithme et permet son exécution. Son code permet la réalisation des étapes 1, 9 et 11 de l'algorithme.
2. Buffer_2D.java : il s'agit de la classe permettant les opérations sur les buffers (réaliser une fusion, créer un buffer, obtenir le segment central d'un buffer). *(Etape 5, 6 et 7 de l'algorithme)*
3. ConstructionToit.java : permet de construire le toit ou le sol d'un cycle à partir des arrêtes supérieures ou inférieures. *(Etape 10 de l'algorithme)*
4. DetectionCoupesZ.java et DectectionCycles.java : contiennent le code permettant de détecter respectivement les coupes en Z et les cycles du bâtiment. *(Etape 2 et 4 de l'algorithme)*
5. FermetureCycle.java : permet de fermer les cycles en ajoutant des segments. *(Etape 8 de l'algorithme)*
6. OrdreSegments.java : permet d'ordonner les segments pour la fusion. *(Etape 6 de l'algorithme)*
7. Proj_sol.java : est le code permettant de réaliser les projections au sol des murs du bâtiment. *(Etape 3 de l'algorithme)*
8. RecollageSegment.java : permet de recoller les segments approximants avant la phase de construction. *(Etape 8 de l'algorithme)*
9. Traitement.java : est la classe permettant un certain nombre d'opérations (recherche d'un point le plus proche …).

2. L'INSTALLATION DE LA LIVRAISON

Le package est fourni dans une petite application graphique qui permet de charger des bâtiments et de les simplifier (avec la possibilité de modifier les paramètres). Cette application est utile pour visionner les résultats et pour comprendre la mécanique de l'algorithme.

Voici les étapes permettant d'installer cette application pour Windows :

Etape 1 : Installer le java developper kit 1.6

Le JDK 1.6 est directement téléchargement depuis le site de Sun à cette adresse : http://java.sun.com/javase/downloads/index.jsp. Ce programme est indispensable pour compiler du code Java.

Etape 2 : Installer JAVA 3D

Cette bibliothèque Java permet de visualiser des objets en 3D. A télécharger à cette adresse : **http://java**.sun.com/**java**se/technologies/desktop/**java3d**

Etape 3 : Installer Jump

Jump est un SIG libre. Il contient certaines bibliothèques utilisées par la plate-forme Géoxygène pour réaliser des opérations géométriques (intersection, calcul de distance …).

Il faut le télécharger à cette adresse : http://www.jump-project.org/, puis le dézipper. On remarquera la présence dans l'archive d'un fichier nommé « jump.jar » qui sera utile pour la suite de l'installation.

Etape 4 : Installer les DLL

La racine du dossier de livraison contient 2 DLL : tetgendll.dll et trianguledll.dll. Il suffit de déplacer ces fichiers dans le dossier system32.dll pour qu'elles soient reconnues par l'application.

Etape 5 : Charger le code source dans un projet Java

Pour réaliser cette étape, il est nécessaire de disposer d'un IDE (par exemple Eclipse, téléchargeable à http://www.eclipse.org/).

Il suffit de créer un nouveau projet à partir du dossier de livraison et indiquer dans quel répertoire se trouve le fichier jump.jar.

Etape 6 : Exécution l'application

Voici les classes présentes dans le packages de livraison (répertoire src). Le package sig3d contient la surcouche Cristage. On y retrouve 5 packages :

- **affichage** concerne l'affichage en 3D d'objets
- **geometrie** contient les classes de géométrie et notamment celles du modèle BRep
- **ihm** contient les classes qui permettent de définir les menus de l'application
- **saisie.ChargementXML** contient les classes permettant le chargement et la sauvegarde d'objets sous le format Bati3D
- **semantique** contient la classe ObjetGeographique qui correspond aux entités géographiques représentés en 3D

- **tetraedrisation** contient des classes qui permettent la décomposition de géométrie en tétraèdres

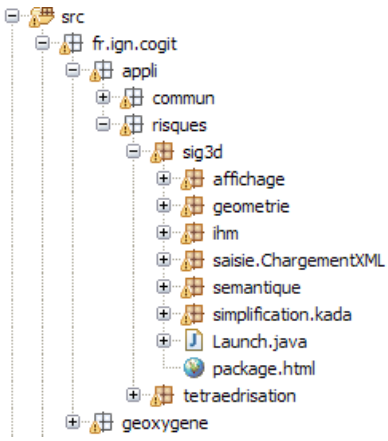

Figure 35 : Package de livraison

La classe permettant le lancement de l'application s'appelle FenetreVueGlobale.java et est contenue dans le package IHM.

3. UTILISER L'APPLICATION

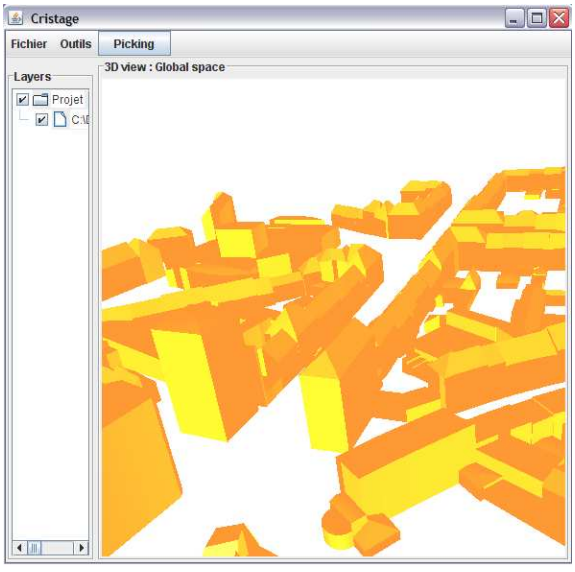

Figure 36 : L'interface de l'application

6.1.1. Les menus

Menu fichier

Ce menu contient 3 possibilités :

Ouvrir un fichier : Cet item permet d'ouvrir un fichier au format bati3D pour l'afficher. Des fichiers au format Bati3D devraient être fournis avec l'application, ils portent l'extension .xml.

Sauvegarder objet sélectionné : Cette option permet de créer un fichier .xml à partir d'un objet sélectionné. Cela est utile pour sauvegarder des objets simplifiés.

Quitter : Ce choix entraine la fermeture de l'application.

Menu outil

Algorithme de Kada : Il s'agit du menu permettant de déclencher une simplification sur un objet sélectionné, la simplification créera une nouvelle couche contenant l'objet simplifié.

Bouton Picking

Ce bouton permet d'activer le mode sélection. Une fois ce mode activé, il est possible de sélectionner des bâtiments en leur cliquant dessus. Lorsqu'un bâtiment est sélectionné, sa couleur d'affichage devient le cyan. Il n'est possible de sélectionner qu'un seul bâtiment à la fois.

6.1.2. Les déplacements

Il existe diverses manières de se déplacer :

Clavier

Les flèches gauches et droites du clavier permettent d'effectuer des rotations autour de l'axe vertical (pour l'écran) dans le sens direct ou indirect.

Les flèches bas et haut permettent de se déplacer en direction de l'écran ou au contraire de reculer.

Souris

Le contrôle « Clic gauche enfoncé + mouvement de la souris » permet d'effectuer des rotations de la caméra.

Le contrôle « Clic droit enfoncé + mouvement de la souris » permet d'effectuer des déplacements de type « Pan ».

4. LANCER UNE SIMPLIFICATION A PARTIR DU CODE SOURCE

Ce code permet l'exécution de l'algorithme. Dans l'application, le code est exécutée dans la classe BarreMenusFenetreGlobale contenue dans le package ihm.

```
Corps corpsInitial;//Le corps corpsInitial sera simplifié

algo_kada.largeur_buffer = 0.6;//Ici on indique les valeurs
algo_kada.seuil = 1.3;//souhaitées pour la simplificaiton
algo_kada.seuil_coupe = 3;

algo_kada.process(corpsInitial);//on execute l'algorithme

//On récupère les corps en sortis
ArrayList <Corps> ListeCorpsSimplifies2= algo_kada.getLCorpsS2();
```

Figure 37 : Code permettant d'exécuter l'algorithme

7. ANNEXE : APPLICATION DE L'ALGORITHME

Cette annexe montre l'application pas à pas de l'algorithme sur le bâtiment suivant :

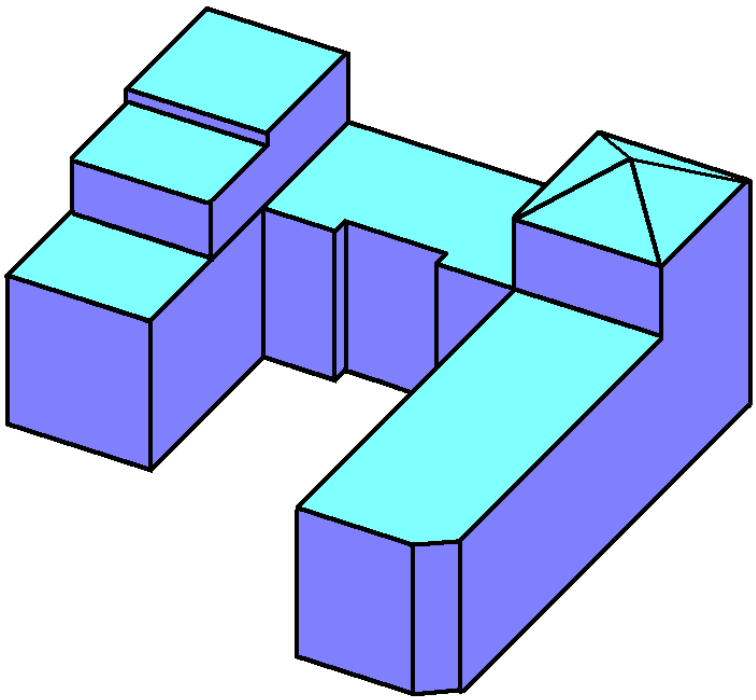

Figure 38 : Bâtiment initial

Etape 1 : *Elimination des faces inclinées*

Les parties inclinées du bâtiment sont détectées et éliminées : le bâtiment n'a alors plus de toit. Tous les pans dont l'angle est supérieur à la valeur du paramètre « Tolérance d'angle » par rapport à la verticale sont éliminés.

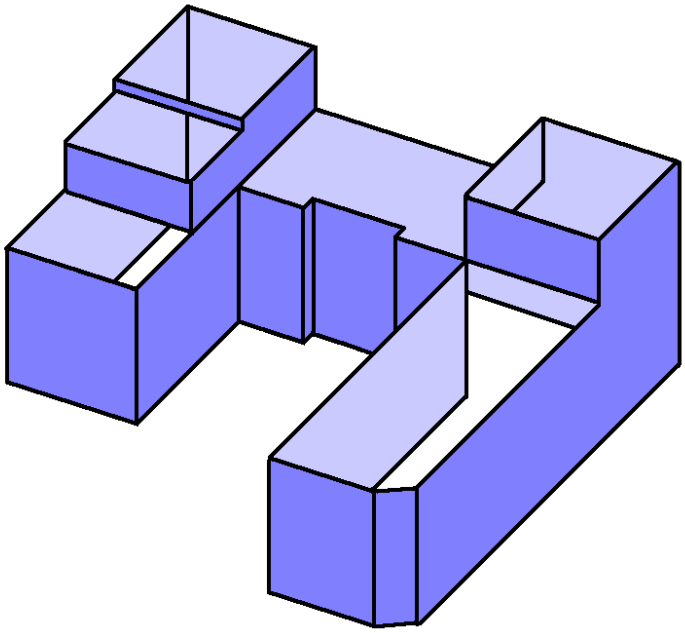

Figure 39 : Bâtiment privé de son toit

Etape 2 : *Détection des coupes en Z*

Les coupes en Z sont obtenues en regroupant les murs dont l'altitude maximale ne diffère que du seuil en Z.

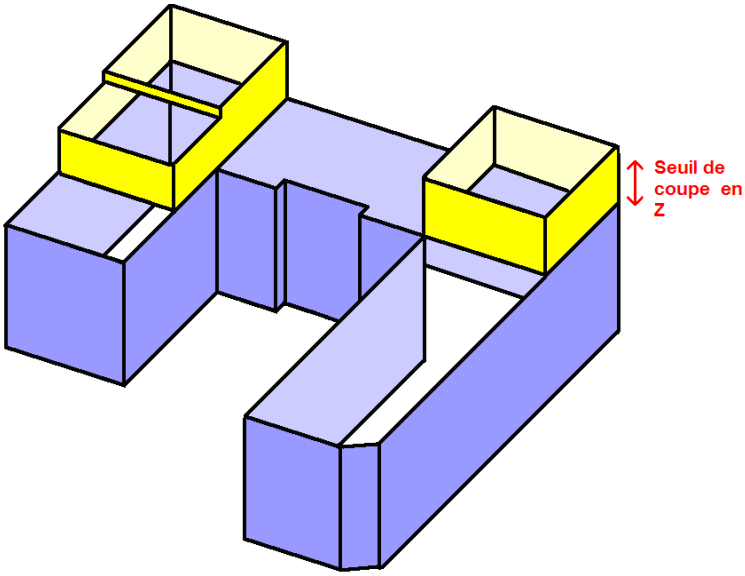

Figure 40 : Les coupes en Z du bâtiment

On obtient dans cet exemple 2 coupes :

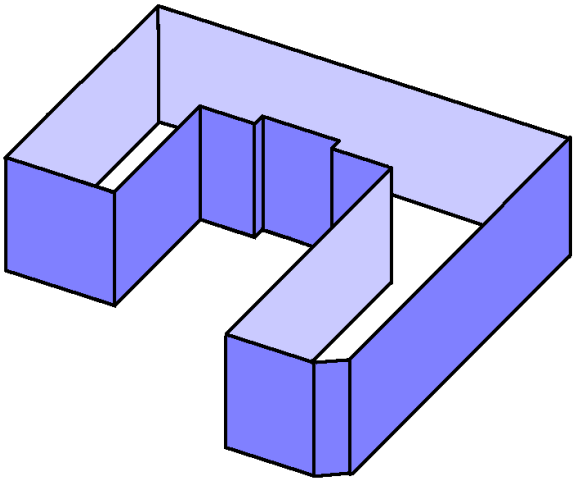

Figure 41 : Première coupe en Z

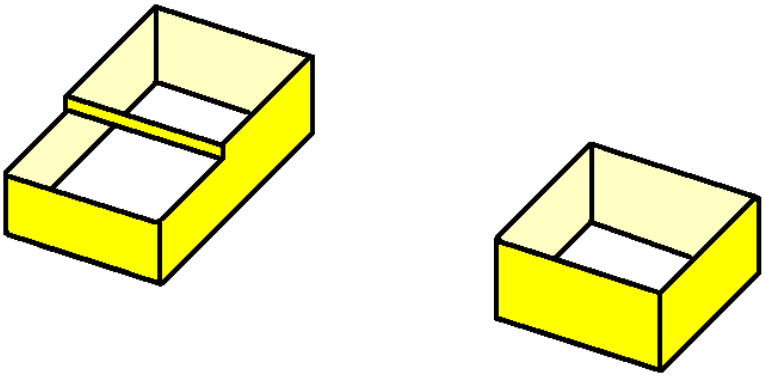

Figure 42 : Seconde coupe en Z

Etape 3 : *Projection des murs pour chaque coupe Détection des coupes en Z*

Pour chaque coupe obtenue, les murs sont projetés au sol pour former un ensemble de segments.

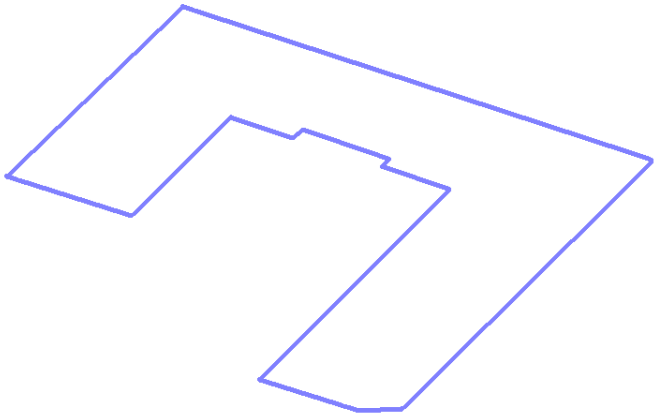

Figure 43 : Première coupe projetée

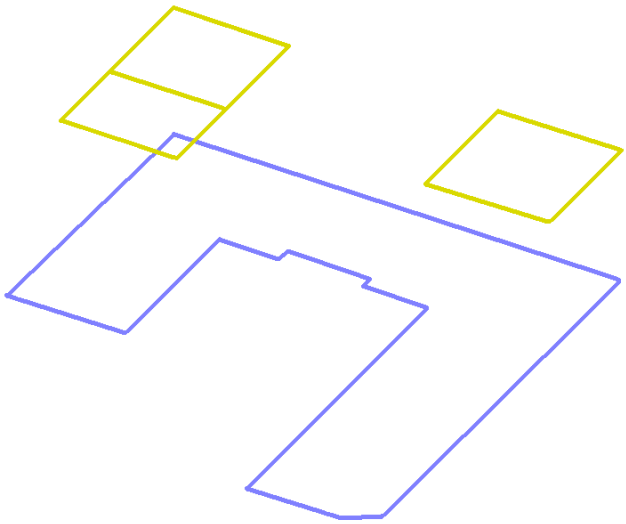

Figure 44 : Les 2 coupes projetées

Etape 4 : *Détection des cycles pour chaque coupe*

Les différentes coupes sont ensuite reparties en cycles, un cycle étant une liste de segments connexes.

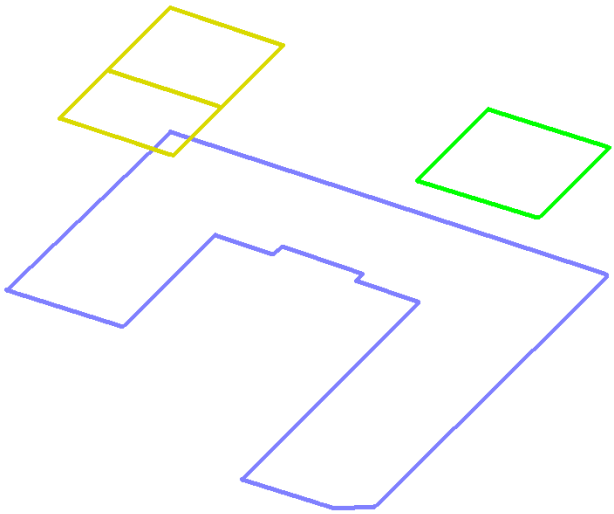

Figure 45 : Les 3 cycles du bâtiment

Etape 5 : *Construction des buffers initiaux*

A partir de cette étape, les cycles sont traités séparément. Seules les opérations sur le plus cycle comptant le plus de segments seront présentées.

Pour chaque segment du cycle, un buffer est calculé selon la méthode suivante.

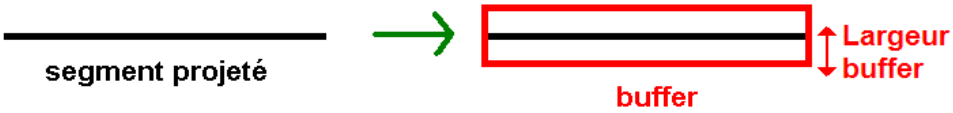

Figure 46 : Méthode de construction des buffers initiaux

La largeur de ce buffer est déterminée par le paramètre « Taille du buffer ».

Etape 6 : *Fusion des buffers*

Il s'agit de l'étape la plus importante de l'algorithme. On détermine comment les buffers doivent fusionner, cela permettra de définir les segments-approximants qui serviront de base pour la construction du bâtiment final.

Un semgment-approximant est un segment qui est obtenu à partir de buffers fusionnés. Il remplacera les segments dont le buffer a fusionné dans le bâtiment final.

On traite successivement chaque buffer pour déterminer si une fusion est possible avec les autres buffers.

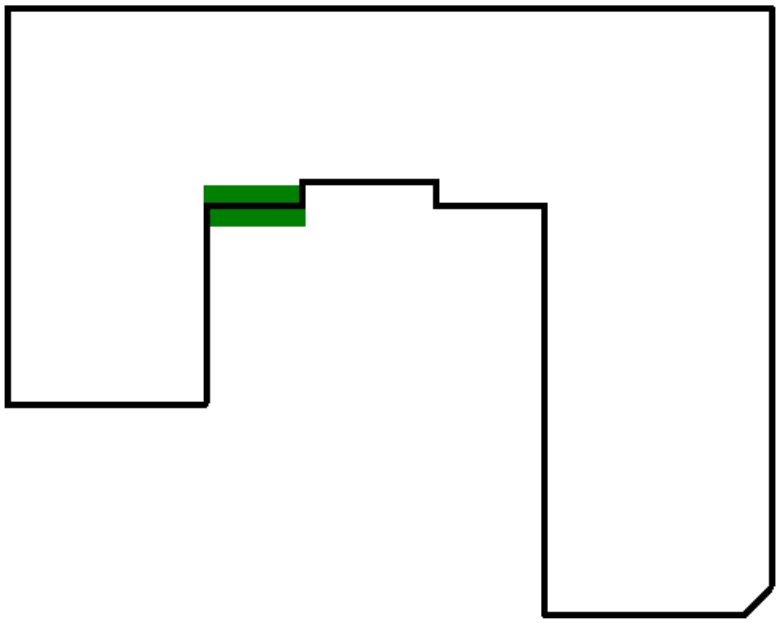

Figure 47: Buffer initial du premier segment

La fusion a lieu si le buffer englobant 2 buffers a une largeur inférieure au paramètre « seuil de fusion ».

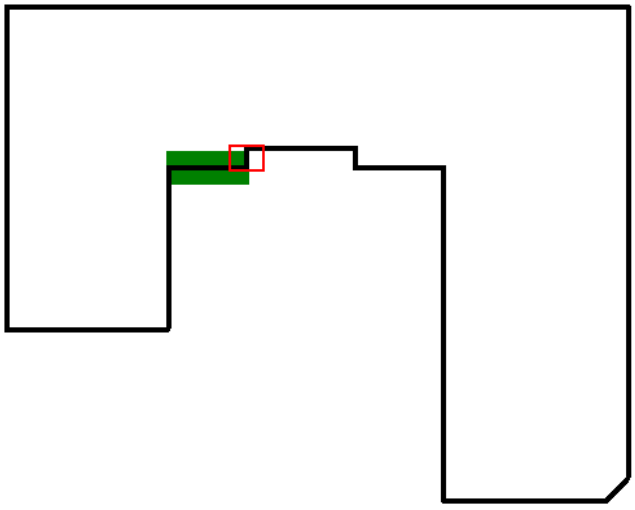

Figure 48 : En rouge, le buffer que l'on essaie de faire fusionner

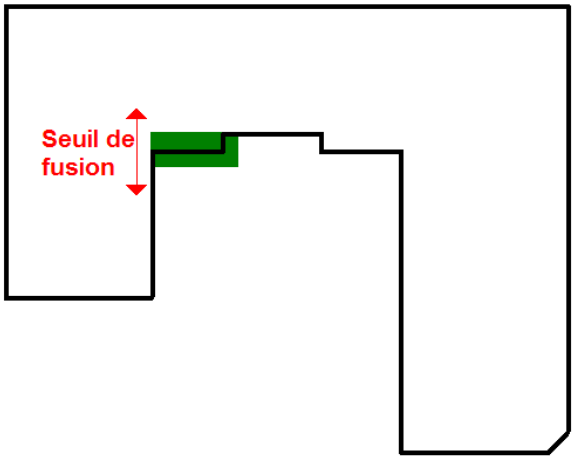

Figure 49 : On garde le buffer résultant

La largeur est inférieure au seuil de fusion, on garde le buffer et on le teste sur un autre buffer.

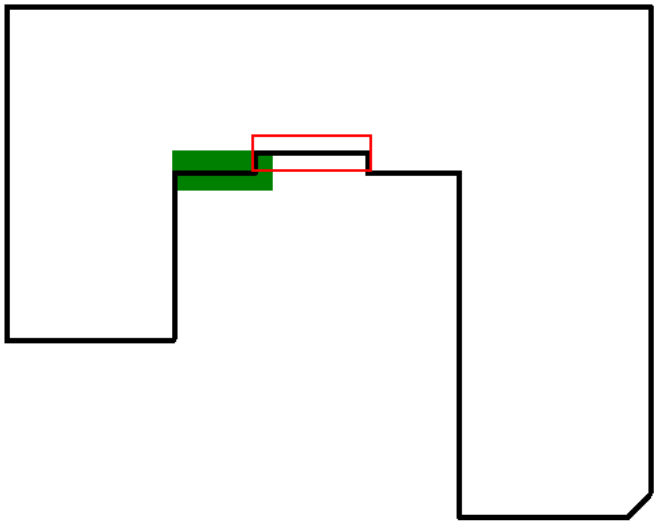

Figure 50 : En rouge, le buffer que l'on essaie de faire fusionner

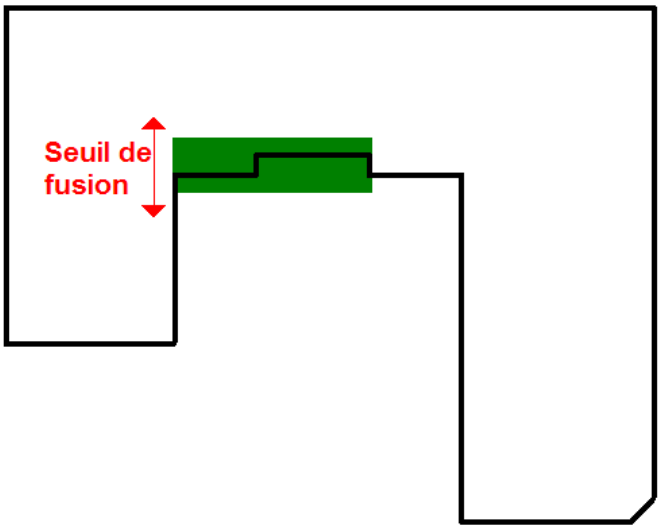

Figure 51: On garde le buffer résultant le buffer

Le buffer résultant a encore une largeur inférieure au seuil de fusion, on garde le buffer résultant et on le teste avec un autre buffer.

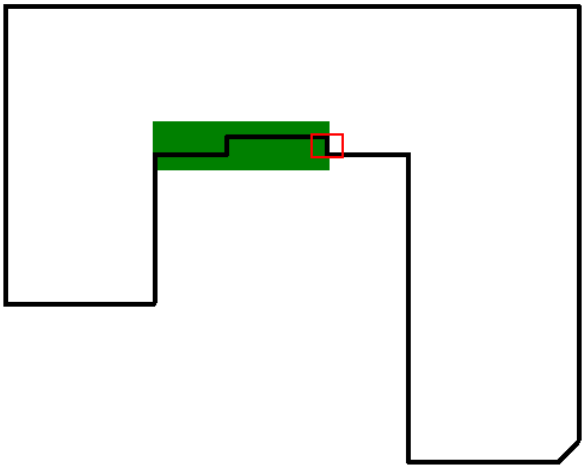

Figure 52 : En rouge, le buffer que l'on essaie de faire fusionner

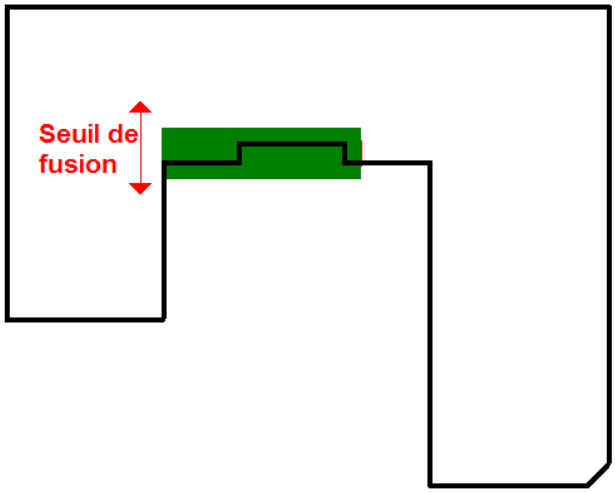

Figure 53 : On garde le buffer résultant

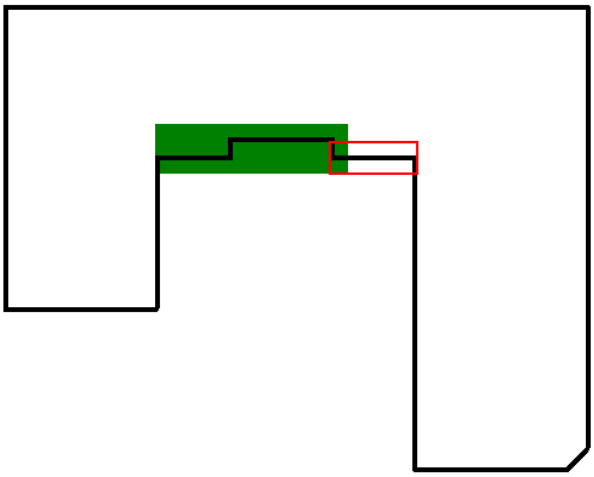

Figure 54 : En rouge, le buffer que l'on essaie de faire fusionner

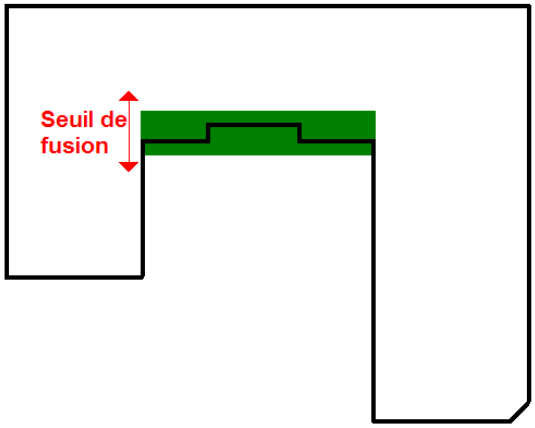

Figure 55 : On garde le buffer résultant

Pour l'instant, le buffer a pu fusionner avec tous les autres buffers.

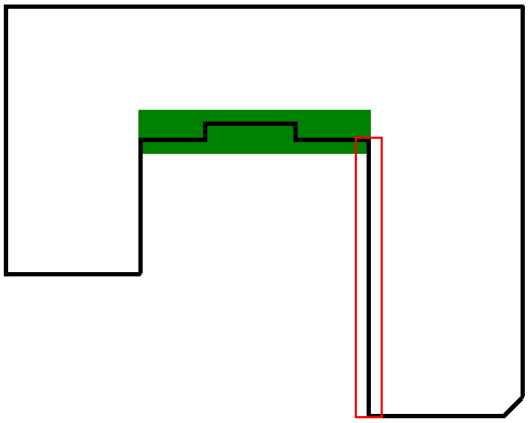

Figure 56 : En rouge, le buffer que l'on essaie de faire fusionner

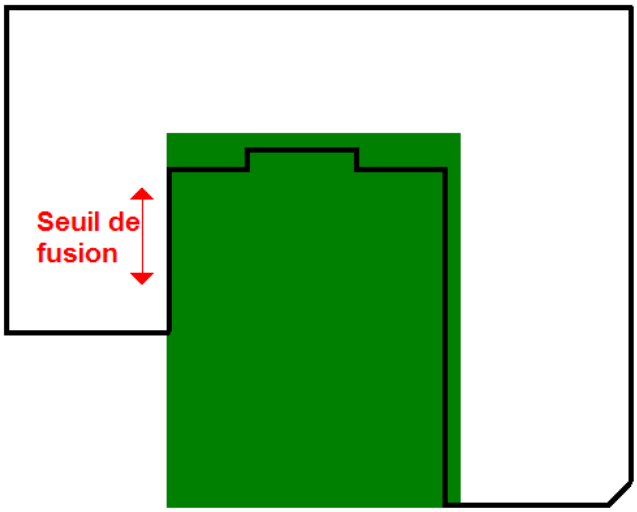

Figure 57: On rejette le buffer

Le buffer a une largeur plus importante que le seuil de fusion. On rejette le buffer, on ne gardera que le buffer précédent. Puisque le buffer précédent ne peut plus fusionner avec aucun autre buffer, il sera utilisé pour la fabrication des segments-approximants (Etape 7).

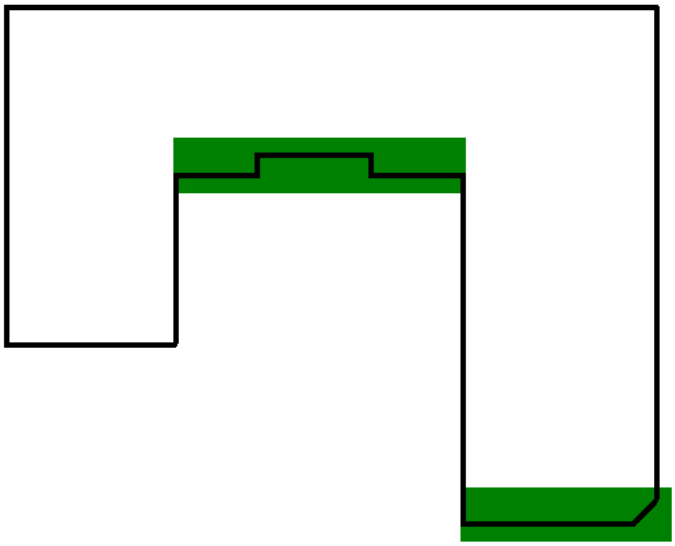

Figure 58 : Les buffers conservés

On répète les étapes précédentes sur tous les autres buffers qui n'ont pas fusionné. On obtient au final 2 nouveaux buffers.

Etape 7 : *Calcul des segments-approximants*

Les segments-approximants sont les axes centraux des buffers :

Figure 59 : Création d'un segment-approximant

On calcule les segments-approximants pour les « nouveaux » buffers.

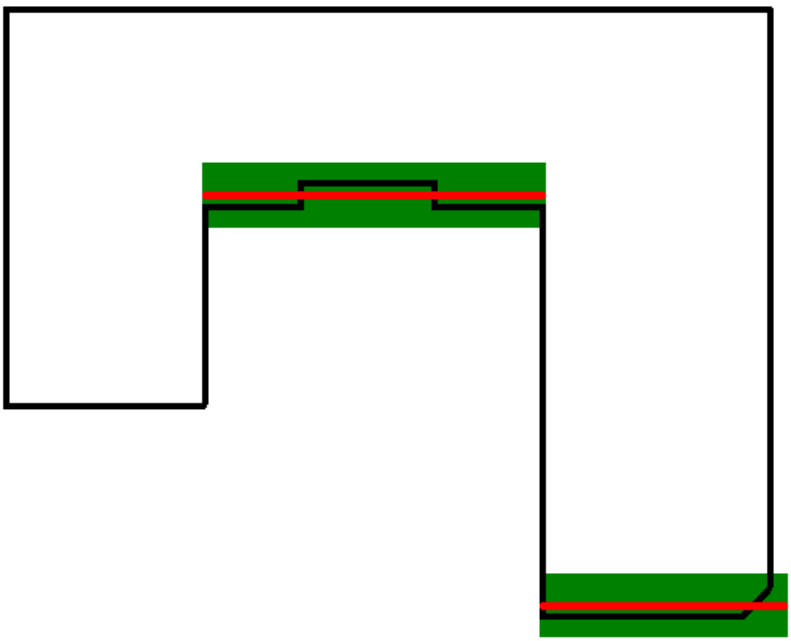

Figure 60: Calcul des segments-approximants

Ces segments-approximants remplaceront les segments dont les buffers ont fusionné.

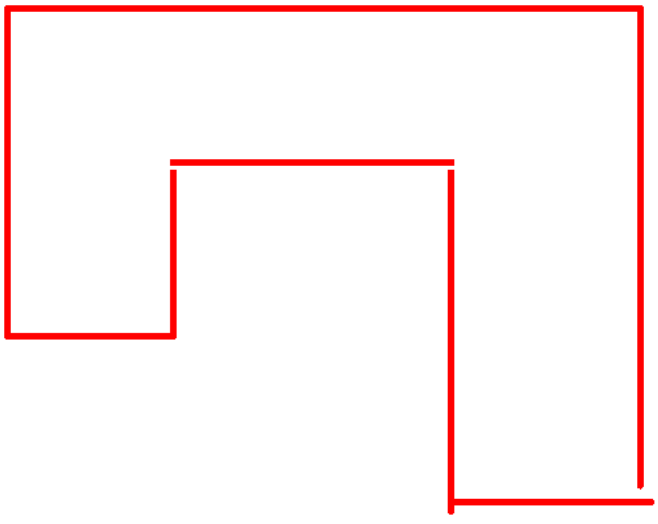

Figure 61: Les segments servant à la reconstruction

Etape 8 : *Recollage des segments-approximants*

Les segments-approximants ne permettent pas d'obtenir un cycle fermé. Il est nécessaire de les recoller.

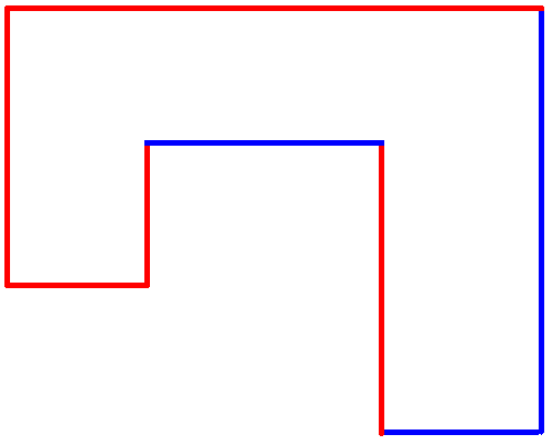

Figure 62 : Recollage des segments-approximants

Etape 9 : *Construction des murs approximants*

On obtient les murs du bâtiment en extrudant les segments-approximants d'une hauteur égale à la hauteur du plus haut pan de mur.

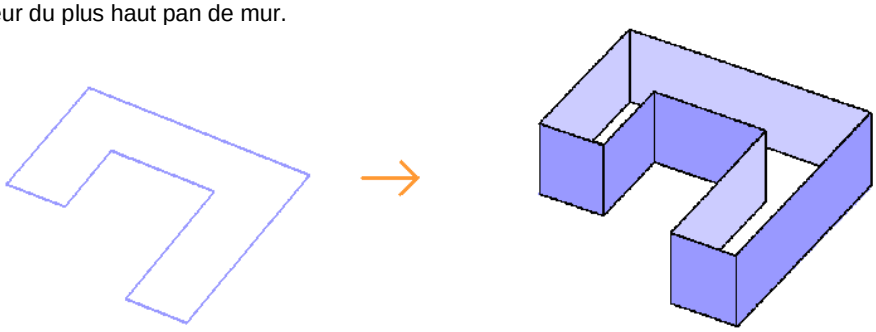

Figure 63 : Extrusion des segments-approximants

On en fait de même avec les autres cycles.

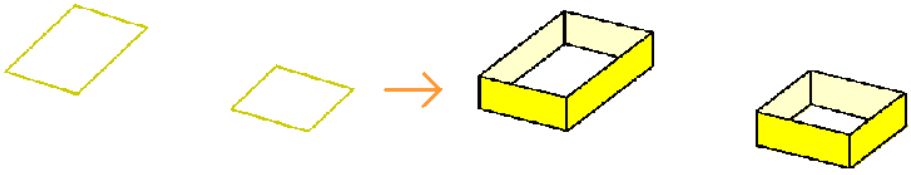

Figure 64 : Extrusion des semgents-approximants des autres cycles

Etape 10 : *Construction du toit et du plancher*

Les toits et le plancher sont construits par « couverture » pour chaque coupe. Un toit plat est ainsi obtenu.

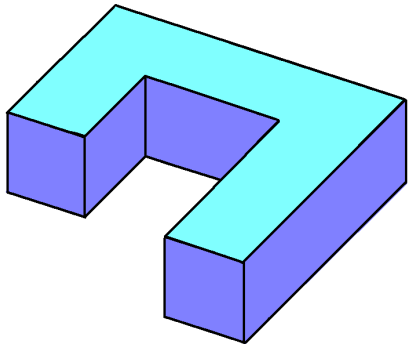

Figure 65 : La coupe inférieure avec son toit

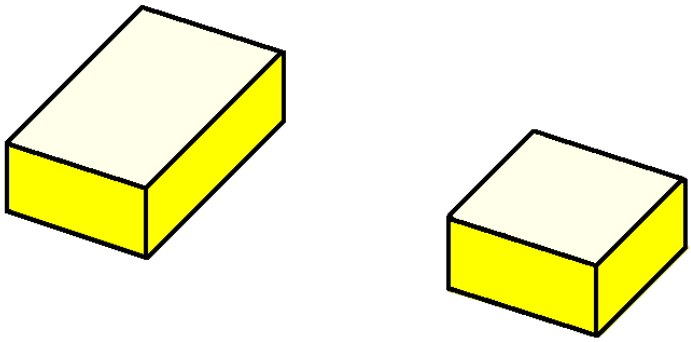

Figure 66 : La coupe supérieure avec son toit

Etape 11 : Fusion des coupes

Les deux coupes sont fusionnées pour obtenir le résultat final.

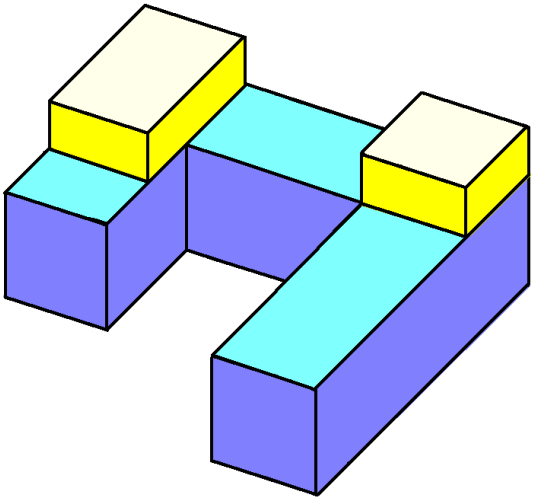

Figure 67 : Résultat de la simplification

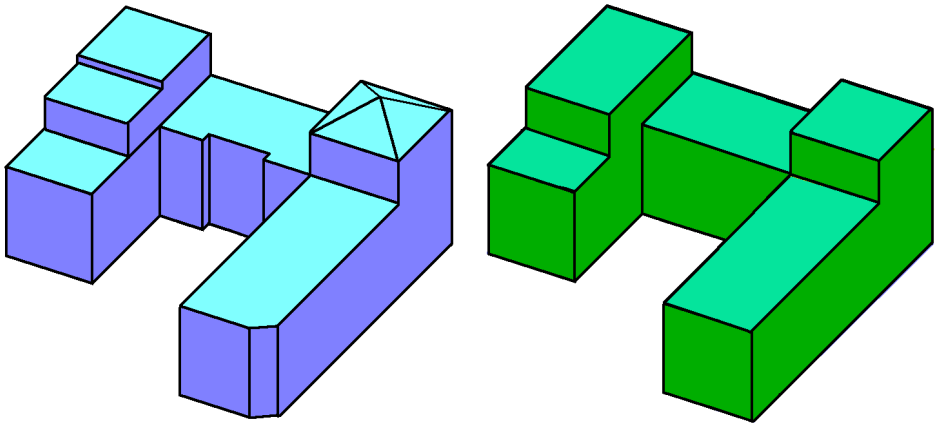

Figure 68 : Bâtiment avant et après simplification