Collège doctoral

N° attribué par la bibliothèque

T H E S E

pour obtenir le grade de
Docteur de l'Ecole des Mines de Paris
Spécialité « Géologie de l'ingénieur »

présentée et soutenue publiquement
par
Nicolas GUIARD

le 29 mai 2006

Construction de modèles géologiques 3D par co-raffinement de surfaces

Directeur de thèse : Michel PERRIN

Jury

M. Marc DANIEL	Rapporteur
M. Jean-François DUFOURD	Rapporteur
M. Yves BERTRAND	Examinateur
M. Jean-Paul CHILES	Examinateur
M. Michel PERRIN	Examinateur
M. Jean-François RAINAUD	Examinateur

Construction de modèles géologiques 3D par co-raffinement de surfaces

Résumé : Le travail présenté dans cette thèse concerne le développement d'outils géométriques à base topologique permettant la construction de modèles géologiques 3D. La construction de tels modèles nécessite l'utilisation d'outils permettant de calculer les intersections entre des surfaces géologiques et de les assembler de manière à obtenir un modèle composé de plusieurs volumes. Cependant, la plupart des algorithmes d'intersection existants permettent uniquement de gérer des volumes isolés.

Pour résoudre ce problème, nous proposons dans un premier temps des algorithmes de co-raffinement en dimensions 2 et 3. Ces algorithmes permettent de modéliser la subdivision de l'espace correspondant à la composition de plusieurs subdivisions. Ainsi, ils offrent la possibilité de calculer des intersections entre des objets dont la dimension topologique est équivalente à celle de leurs plongements géométriques. Nous expliquons aussi comment il est possible à partir de tels algorithmes d'obtenir les résultats fournis par des opérations plus classiquement utilisées en CAO qui sont les opérations booléennes.

Dans un second temps, nous présentons le domaine de la modélisation géologique ainsi que les différents problèmes existants et proposons des solutions pour les résoudre. Nous expliquons de quelle manière nous utilisons les précédentes opérations de co-raffinement pour assembler des surfaces correspondant à des objets géologiques 3D, et obtenir ainsi un modèle structural topologiquement cohérent correspondant à l'interprétation du géologue.

Construction of 3D geological models by surfaces co-refinement

Abstract: The goal of the work is the design of topological based geometrical tools, which allow the construction of 3D geological models. The construction of such models needs the use of tools which allow to compute the intersections between geological surfaces and to assemble them in order to obtain a model composed of several volumes. However, most of the existing intersection algorithms only deal with isolated volumes.

In order to solve this problem, we first propose co-refinement algorithms in dimensions 2 and 3. These algorithms allow to build the space subdivision corresponding to the composition of several subdivisions. Thus, they offer the possibility to compute intersections between objects whose topological dimension is equal to the dimension of their geometrical embeddings. We also explain how such algorithms can be used for obtaining the results currently provided in CAD by boolean operations.

We then present the geological modeling domain as well as the current related issues and we propose solutions to solve them. We explain how the co-refinement algorithms can be used for assembling surfaces corresponding to 3D geological objects, and for thus obtaining a 3D consistent structural model in correspondence to the geologist's interpretation.

Remerciements

Tout d'abord, je tiens à remercier Marc Daniel et Jean-François Dufourd pour avoir accepté la charge de rapporteur et espère que la lecture de ce mémoire leur a été agréable malgré les nombreuses parties très techniques. Je remercie également tous les membres du jury pour avoir accepté d'assister à la présentation de ce travail.

Je remercie sincèrement Michel Perrin pour avoir été mon directeur de thèse pendant ces trois années et demi et pour avoir tenu ce rôle d'une main de maître en dépit des divers problèmes rencontrés. Il m'a permis de découvrir un domaine qui m'était jusqu'alors inconnu et a toujours été disponible pour répondre à mes questions. Je le remercie pour toute l'aide qu'il m'a apportée ainsi que pour ses nombreux conseils et corrections lors de la rédaction de ce mémoire.

Je voudrais aussi remercier Yves Bertrand et Jean-François Rainaud pour m'avoir co-encadré et pour toute l'energie positive qu'ils ont sû me transmettre dans les moments les plus durs. Ils ont été d'une aide très précieuse tout au long de cette thèse et ont toujours été là lorsque le besoin s'en est fait sentir. Je les remercie aussi tout particulièrement pour les nombreuses relectures de mon mémoire et les corrections qu'ils y ont apportées.

Je remercie les sociétés Beicip-Franlab, ELF, Géocap et IRAP/RMS pour les modèles qu'ils nous ont fournis, m'ayant ainsi permis de tester mes outils sur différentes configurations géologiques.

Je remercie Guillaume et Fred pour avoir fait de Moka ce qu'il est, et sur lequel tous les travaux de cette thèse reposent. Je les remercie aussi pour toute l'aide qu'ils m'ont apportée au début.

Je remercie tous les thésards (anciens et actuels) du laboratoire SIC de Poitiers avec qui j'ai eu l'occasion de travailler, de partager mes nombreux bureaux et pour tous les bons moments passés ensemble : Adrian, Alexandru, Bruno, David, Fred, Matthias, Mehdi (pas encore thésard mais bientôt, je l'espère ;-), Pascal, Pierre-François, Rodolphe, Sam, Sylvain...

Je remercie également tous les autres membres du laboratoire SIC que j'ai eu l'occasion de côtoyer pour diverses raisons et avec lesquels j'ai été heureux de passer ces trois années.

Je remercie les quelques personnes de l'IFP à Rueil-Malmaison, du LSIIT à Strasbourg et du LSIS à Marseille avec qui j'ai eu l'occasion de travailler et pour les bons moments que nous avons partagés : Mehdi, Philippe, Sébastien, Sylvain, Van.

Pour finir, je tiens à adresser un grand merci à Martine pour ses encouragements, son soutien et ses conseils avisés qui m'ont été très utiles tout au long de cette thèse.

Table des matières

Introduction générale

Cette thèse s'inscrit dans deux domaines qui sont la modélisation solide et la modélisation géologique. Plus précisément, nous nous intéressons dans un premier temps à des techniques permettant d'assembler des objets topologiques 3D en les intersectant. Ensuite, nous utilisons ces outils dans le but de construire des modèles géologiques structuraux en dimension trois. Nous présentons tout d'abord le domaine de la modélisation solide ainsi que le sujet auquel nous nous intéressons. Puis nous expliquons les différents problèmes se posant actuellement en modélisation géologique et proposons des méthodes permettant de les résoudre.

La modélisation solide

Le but de la modélisation solide est de construire, à l'aide d'outils informatiques, des objets pouvant être utilisés pour différentes applications. Ils peuvent par exemple servir uniquement à des fins visuelles et n'ont alors pas besoin d'être modélisés avec une grande précision. C'est en particulier le cas pour le cinéma où seule l'esthétique des objets compte. Il peuvent aussi être utilisés dans le but d'effectuer des simulations afin de tester leurs propriétés physiques. Dans ce cas, la précision des objets est très importante et leur modélisation doit être la plus rigoureuse possible.

En fonction de l'application visée, les techniques de modélisation peuvent varier. Il est par exemple souvent utile de répliquer des objets existants dans la réalité. Dans le cas d'objets complexes il existe des techniques consistant à récupérer des ensembles de points sur la surface des objets (IRM, scanners lasers, ...) afin de pouvoir s'en servir pour les reconstruire informatiquement. La construction d'objets à partir de nuages de points fait alors appel à des techniques de reconstruction de surfaces qui permettent d'obtenir différents résultats en fonction de l'organisation des points et de leur précision. Il est par exemple possible d'obtenir des surfaces passant exactement par les points ou bien des surfaces approximées qui permettent de gommer les éventuelles irrégularités.

Dans le cas d'objets simples ou n'existant pas dans la réalité, la modélisation géométrique offre de nombreux outils permettant de les construire. La méthode la plus utilisée dans ce domaine consiste à combiner des éléments de base afin de constituer l'objet désiré. Cette approche est principalement utilisée dans le domaine de la CAO afin de modéliser des pièces mécaniques à l'aide d'opérations booléennes (union, intersection, différence).

Certaines formes étant cependant compliquées à obtenir à l'aide des outils précédents, il est possible de les obtenir grâce à des méthodes de déformation d'objets. Cette technique permet de

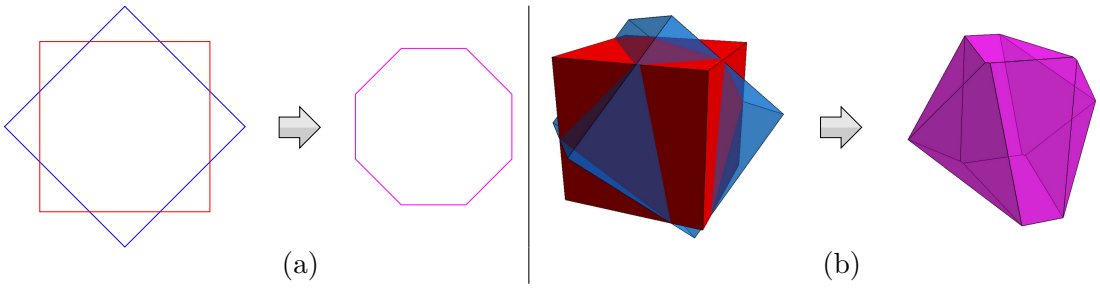


FIG. 1 – Exemples d'intersections entre des objets dont la dimension topologique est inférieure à la dimension de leurs plongements géométriques.

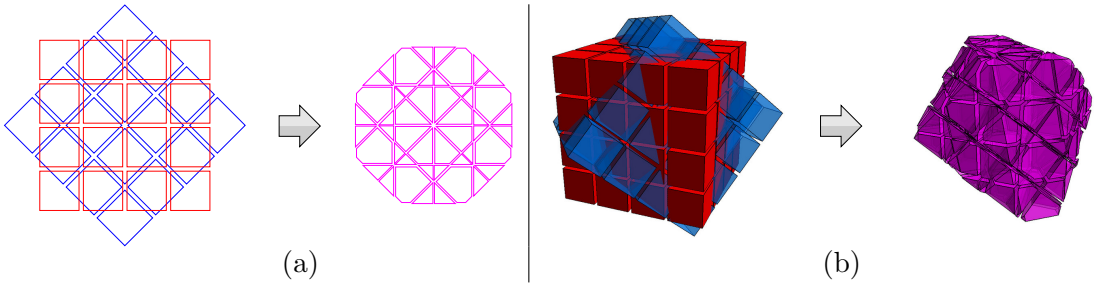


FIG. 2 – Exemples d'intersections entre des objets dont la dimension topologique est équivalente à la dimension de leurs plongements géométriques.

modifier la géométrie des objets sans toucher à leur structure. Elle est fréquemment utilisée en animation pour tordre des objets ou bien pour simuler le comportement d'objets mous. Cette technique peut cependant être utilisée comme une méthode de modélisation à part entière en déformant les objets interactivement.

Depuis maintenant une vingtaine d'années, la topologie a fait son apparition dans le domaine de la modélisation solide. Son but est de décrire précisément la structure des objets en les découpant en cellules de différentes dimensions et en décrivant les relations d'incidence et d'adjacence qu'elles entretiennent. À l'aide de structures de données topologiques définissant toutes ces relations, il est alors possible de concevoir des algorithmes complexes beaucoup plus simplement et dans le meilleur des cas, des algorithmes plus performants.

Le domaine d'intérêt de cette thèse concerne la prise en compte de la topologie dans l'assemblage d'objets géométriques. Plus précisément, nous nous intéressons à la réalisation d'intersections entre des objets maillés et aux modifications devant être opérées pour maintenir une topologie cohérente. Dans la littérature, ce problème est très généralement abordé pour des objets surfaciques et consiste à calculer des intersections entre des objets dont la dimension topologique (généralement 2) est inférieure à la dimension de leur plongements géométriques (généralement 3) (cf. FIG. 1). Notre but est de concevoir des algorithmes permettant de calculer les intersections entre des objets dont la dimension topologique est équivalente à celle de leur plongements, c'est-à-dire des maillages 2D composés de plusieurs faces adjacentes (cf. FIG. 2(a)), et des maillages 3D composés de plusieurs volumes adjacents (cf. FIG. 2(b)).

Dans cette optique, nous avons réalisé plusieurs algorithmes de co-raffinement sur le modèle

topologique des cartes généralisées. Ces algorithmes permettent de fusionner plusieurs subdivisions de l'espace en une seule. Nous avons commencé par étudier le problème en dimension deux en concevant un algorithme optimisé pour le traitement de maillages. Cet algorithme fonctionne à l'aide d'une méthode par propagation consistant à utiliser la topologie des objets pour détecter localement les intersections.

Sur ce même principe, nous avons ensuite développé un premier algorithme de co-raffinement en dimension trois. Celui-ci permet de calculer localement les intersections sur les surfaces des objets en suivant les lignes de coupe générées. Nous nous sommes alors confrontés à de nombreux problèmes, donc nous avons décidé de réaliser un nouvel algorithme utilisant une autre technique. Ce dernier est basé sur des méthodes qui permettent de calculer tous les segments de coupes existants entre deux faces isolées, et de réitérer ce processus pour chaque face des objets.

Les problématiques propres à la modélisation géologique

La modélisation géologique s'intéresse à la représentation des couches géologiques du sous-sol. Elle implique en particulier l'assemblage de surfaces élémentaires en vue de construire des modèles structuraux composés de plusieurs blocs géologiques. Ces modèles sont ensuite utilisés dans le but d'effectuer des simulations d'écoulement en peuplant les différents blocs par des propriétés pétrophysiques. Ces simulations servent alors à déterminer si les modèles sont succeptibles de renfermer des réservoirs à hydrocarbures.

La construction de tels modèles passe par de nombreuses étapes et nécessite l'utilisation d'outils provenant de la modélisation solide. Les surfaces permettant de construire ces modèles proviennent d'acquisitions obtenues par forages ou bien par propagation d'ondes sismiques dans le sol. Leur modélisation nécessite alors l'utilisation d'algorithmes de reconstruction de surfaces. Ces surfaces doivent ensuite être assemblées en les intersectant et en construisant la topologie associée au découpage du modèle en plusieurs blocs. De plus, lors de cette dernière étape, il peut être nécessaire de déformer les surfaces afin que leur découpe soit la plus propre possible.

Toutes ces étapes de construction sont opérées par un ingénieur géologue et dépendent alors de l'interprétation qu'il peut faire des données qu'il possède en entrée. En effet, les données sismiques étant souvent incertaines, il est relativement difficile de déterminer avec exactitude la configuration des différents objets géologiques. Ainsi, le géologue doit faire des choix dès le début de la chaîne de traitement et doit se tenir à son interprétation tout au long de celle-ci. Il s'agit alors d'un problème majeur lorsque le géologue décide de changer son interprétation suite à des nouvelles données ou bien suite à une incohérence soulevée lors du processus de construction.

Pour cette thèse, nous nous sommes tout particulièrement intéressés au découpage des différentes surfaces composant une scène géologique et à la construction de la topologie correspondant à leur assemblage. Nous avons alors conçu différents outils basés sur les algorithmes de co-raffinement développés auparavant et permettant de construire aisément des modèles structuraux complexes. De plus, ces outils ont pour but d'automatiser la construction des modèles en générant une topologie correspondant à l'interprétation du géologue.

Tous ces travaux ont été réalisés au sein d'une collaboration entre l'École des Mines de Paris, l'Institut Français du Pétrole et le laboratoire Signal Image Communication de Poitiers. Ils visent

à être intégrés dans une nouvelle génération de modeleurs géologique basés sur la connaissance. Les divers algorithmes ont été développés dans Moka[1], un modeleur basé sur un noyau de cartes généralisées de dimension trois.

Organisation du mémoire

Ce mémoire s'organise en trois parties. La première d'entre elle présente tout d'abord en quoi consiste un algorithme de co-raffinement (cf. chapitre 1) et en quoi il diffère des opérations booléennes classiques. Nous présentons ensuite les structures de données utilisées (cf. chapitre 2), puis nous donnons un premier algorithme en dimension deux (cf. chapitre 3). Pour finir, nous expliquons comment il est possible d'utiliser le co-raffinement dans le cadre d'opérations booléennes (cf. chapitre 4).

Dans la deuxième partie, nous présentons les deux algorithmes de co-raffinement 3D que nous avons réalisés. Nous commençons par présenter les différences existantes avec un algorithme 2D et nous discutons des différentes approches possibles (cf. chapitre 5). Nous expliquons ensuite le premier algorithme 3D que nous avons développé ainsi que ses avantages et inconvénients (cf. chapitre 6). Nous finissons par présenter le nouvel algorithme développé et nous donnons sa description complète ainsi que les résultats obtenus (cf. chapitre 7).

Dans la troisième partie, nous présentons les travaux effectués dans le domaine de la modélisation géologique. Nous commençons par introduire le domaine (cf. chapitre 8) puis nous présentons les différents outils déjà existants (cf. chapitre 9). Nous expliquons ensuite la nouvelle méthodologie que nous avons adopté et donnons les résutats que nous avons obtenus (cf. chapitre 10).

Pour finir, nous concluons et discutons des perspectives de ces travaux. Nous présentons aussi en annexe A et B des exemples d'application des différents algorithmes présentés dans cette thèse ainsi qu'une partie des modèles géologiques sur lesquels nous avons travaillé.

[1] `http://www.sic.sp2mi.univ-poitiers.fr/moka/`

Première partie

Co-raffinement et opérations booléennes : présentation et application en dimension 2

Chapitre 1

Introduction

La modélisation géométrique a pour objectif la construction « informatique » de tout type d'objet géométrique. Celle-ci est tout particulièrement utilisée en Conception Assistée par Ordinateur (CAO) où il est nécessaire de pouvoir construire des pièces de mécanismes complexes à l'aide d'un ensemble d'opérations de base. La méthode la plus fréquemment utilisée dans ce domaine consiste à simuler des usinages à l'aide d'opérations booléennes. Ces dernières permettent de calculer des intersections, des unions et des soustractions entre objets (cf. Fig. 1.1).

Le processus permettant de déterminer le résultat d'une opération booléenne se décompose en plusieurs phases. Il s'agit tout d'abord de détecter les cellules de chaque objet qui sont en intersection puis de les découper mutuellement en plusieurs morceaux afin de modéliser leur intersection (cf. Fig. 1.2(b)). Parmi les découpes obtenues, il est ensuite nécessaire de déterminer lesquelles appartiennent à l'objet final en fonction de l'opération booléenne choisie (cf. Fig. 1.2(c)). Pour terminer, il reste à supprimer les parties d'objets inutiles et à recoller celles restantes afin d'obtenir le bon résultat (cf. Fig. 1.2(d)). Les auteurs de [MK89] décrivent en détail une manière de procéder qui applique ces différents principes.

Les algorithmes récents [CD96, Caz97] s'appuient sur une opération unique appelée coraffinement. Celle-ci permet de calculer les intersections entre deux subdivisions de l'espace afin d'obtenir une nouvelle subdivision formée des deux précédentes. Le résultat de cette opération est alors constitué de différentes régions correspondant aux résultats pouvant être obtenus par les opérations booléennes classiques. L'intérêt majeur de cette opération est qu'elle ne se limite pas au traitement d'objets pleins car elle peut gérer n'importe quel type de subdivision comme par exemple des maillages (cf. Fig. 1.3).

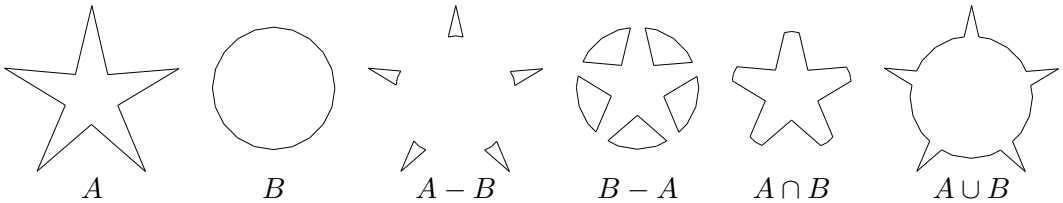


Fig. 1.1 – Exemple d'opérations booléennes.

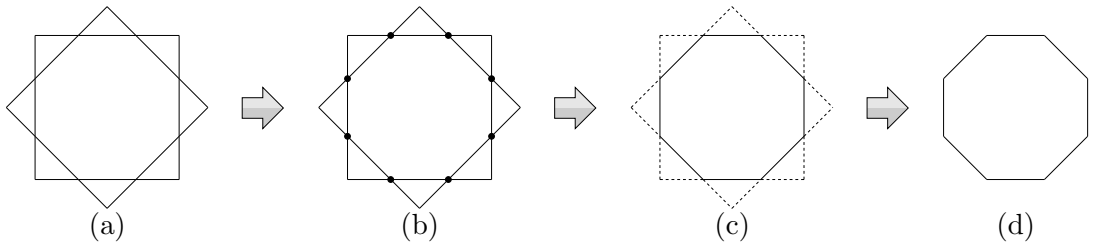


FIG. 1.2 – Calcul de l'intersection de deux objets à l'aide d'opérations booléennes : (a) objets originaux ; (b) détection des points d'intersection ; (c) détection des parties inutiles ; (d) suppression des parties inutiles et assemblage du résultat.

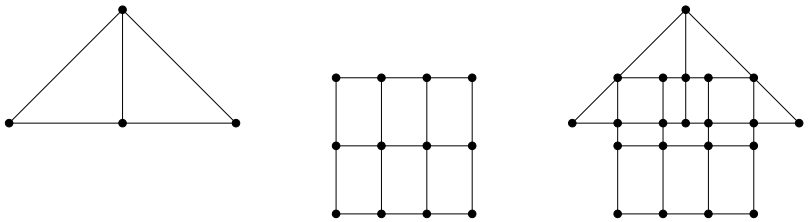


FIG. 1.3 – Exemple de co-raffinement entre deux subdivisions.

Dans cette première partie, nous proposons un algorithme de co-raffinement en dimension 2. Nous commençons par présenter dans le chapitre 2 les différentes structures de données que nous utilisons. Nous expliquons ensuite dans le chapitre 3 la manière dont cet algorithme est conçu. Nous finissons en expliquant dans le chapitre 4 comment il est possible de récupérer les résultats des différentes opérations booléennes à l'aide du co-raffinement.

Chapitre 2

Structures de données

Les travaux présentés dans ce mémoire s'appuient sur le modèle topologique des cartes généralisées. Ce modèle a été choisi car il permet de modéliser des topologies en dimension n. Ainsi, nous pouvons construire aisément des maillages volumiques en dimension trois et connaître toutes les relations d'adjacence et d'incidence existantes entre les différentes cellules.

Nous avons aussi besoin, dans les différents algorithmes, de structures de données annexes permettant de représenter des informations géométriques ou bien permettant de stocker des éléments dans des listes, des files ou des piles. Nous présentons ici toutes ces structures et indiquons la manière dont elles sont utilisées dans ce document.

Toutes les structures de données ainsi que les opérations définis dans cette section sont toutes décrites à l'aide de pseudo-code afin de ne pas rentrer dans des détails d'implémentation.

2.1 Cartes généralisées

Les cartes généralisées ou G-Cartes font partie des modèles de représentation des objets par leurs bords (B-rep). Ce modèle, introduit par Pascal Lienhardt [Lie94], permet de représenter des subdivisions de $\mathbb{R}^n$ en cellules de dimension i (i-cellules), $i \in [0, n]$. Nous commençons ici par donner quelques définitions, puis nous introduisons quelques concepts utiles dans la suite de ce document.

2.1.1 Définitions

Une G-Carte permet de représenter une quasi-variété cellulaire orientable ou non orientable, avec ou sans bord. Une quasi-variété correspond à un assemblage deux à deux de cellules de dimensions n le long de cellules de dimension $n-1$. Une G-Carte se compose d'éléments de base appelés brins sur lesquels sont définis des opérateurs α_i permettant de les relier entre eux (cf. Fig. 2.1).

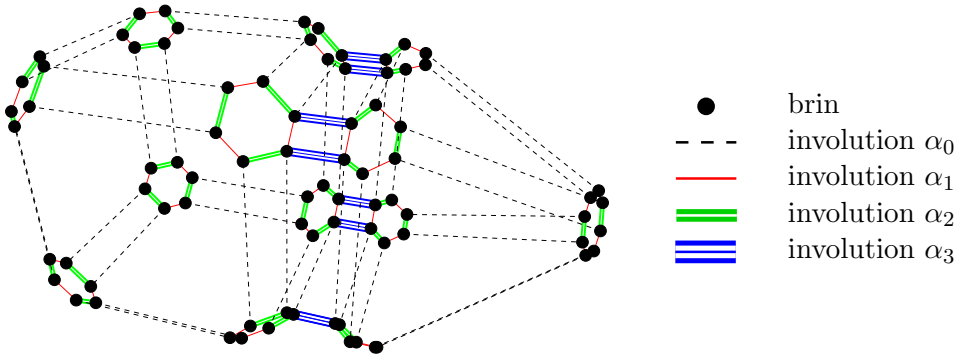


FIG. 2.1 – Représentation éclatée d'une 3-G-Carte.

DÉF. 1 – Une carte généralisée de dimension n $G = (B, \alpha_0, \alpha_1, \ldots, \alpha_n)$ appelée aussi n-G-Carte, est définie par :

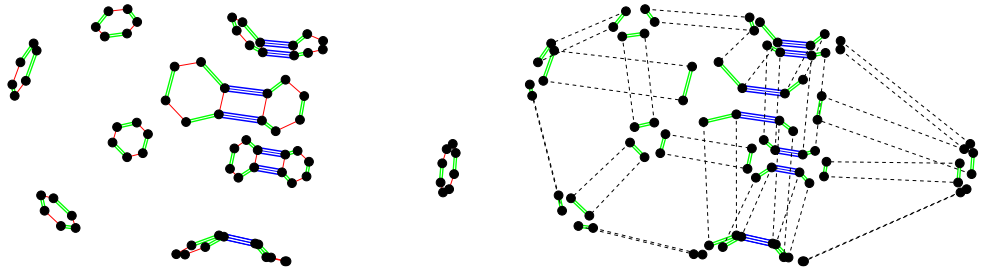
- B est un ensemble de brins,
- $(\alpha_i)_{i=0,\ldots,n}$ des bijections de B dans B telles que $\forall i$, α_i est une involution ($\alpha_i \circ \alpha_i = id$),
- $\forall i, j$ tels que $j \geq i + 2$ alors $\alpha_i \circ \alpha_j$ est une involution.

La notion de cellule de dimension i peut être retrouvée dans une n-G-Carte par l'intermédiaire de la notion plus générale d'orbite. Celle-ci permet d'obtenir un ensemble de brins par composition d'un ensemble d'involutions.

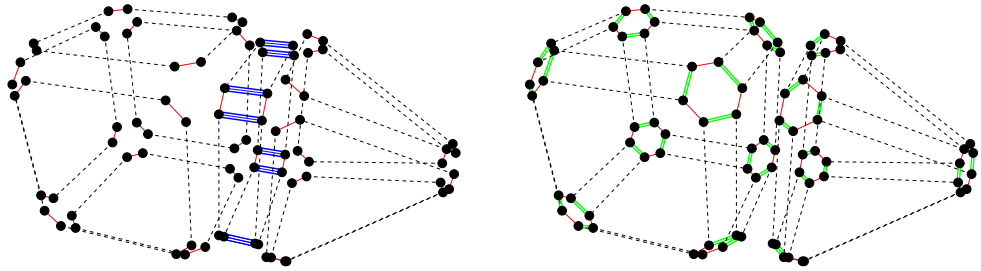
DÉF. 2 – Soit $G = (B, \alpha_0, \alpha_1, \ldots, \alpha_n)$ une n-G-Carte et $b \in B$ un brin. On note $<\alpha_{i_0}, \ldots, \alpha_{i_k}>(b)$, $0 \leq i_0 < \ldots < i_k \leq n$, l'orbite de b par rapport à $\alpha_{i_0}, \ldots, \alpha_{i_k}$. Ceci correspond à l'ensemble des brins accessibles depuis b par une composition quelconque de ces applications.

Une i-cellule correspond à une orbite contenant l'ensemble des involutions excepté α_i (cf. FIG. 2.2). Par exemple, en dimension 3, pour obtenir l'ensemble des brins définissant une face (cellule de dimension 2), nous utilisons l'orbite $<\alpha_0, \alpha_1, \alpha_3>$. Dans la suite de ce document, lorsque nous parlons d'orbite O ou de cellule C incidente à un brin b, nous faisons référence à l'ensemble des brins accessibles depuis b en utilisant l'orbite O ou bien l'orbite correspondant à la cellule C.

Une n-G-Carte peut être construite à l'aide de deux opérations de base. La première d'entre elles est l'ajout de brins qui sont invariants pour toutes les involutions. La deuxième est la couture par α_i de deux orbites $<\alpha_0, \ldots, \alpha_{i-2}, \alpha_{i+2}, \ldots, \alpha_n>$. En particulier, il est possible de construire une 3-G-Carte par assemblage de volumes (orbite $<\alpha_0, \alpha_1, \alpha_2>$) en les reliant par α_3. Ceci consiste intuitivement à assembler deux faces possédant la même structure, c'est-à-dire à coudre par α_3 deux orbites $<\alpha_0, \alpha_1>$ isomorphes.



(a) sommets (orbites $<\alpha_1, \alpha_2, \alpha_3>$) (b) arêtes (orbites $<\alpha_0, \alpha_2, \alpha_3>$)



(c) faces (orbites $<\alpha_0, \alpha_1, \alpha_3>$) (d) volumes (orbites $<\alpha_0, \alpha_1, \alpha_2>$)

FIG. 2.2 – Séparation des différentes cellules d'une 3-G-Carte.

DÉF. 3 – Soit $G = (B, \alpha_0, \alpha_1, \ldots, \alpha_n)$ une n-G-Carte et $b, b' \in B$ deux brins tels que b et b' sont invariants par α_i. La couture par α_i de b et b' peut être réalisée si et seulement si $<\alpha_0, \ldots, \alpha_{i-2}, \alpha_{i+2}, \ldots, \alpha_n>(b)$ est isomorphe à $<\alpha_0, \ldots, \alpha_{i-2}, \alpha_{i+2}, \ldots, \alpha_n>(b')$ par une application ϕ. Le résultat de cette opération est donc une n-G-Carte $G' = (B, \alpha_0, \ldots, \alpha_{i-1}, \alpha'_i, \alpha_{i+1}, \ldots, \alpha_n)$ telle que :

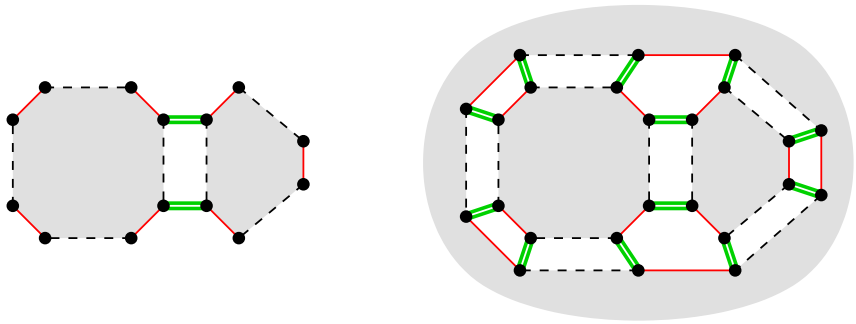
- $\forall b'' \in <\alpha_0, \ldots, \alpha_{i-2}, \alpha_{i+2}, \ldots, \alpha_n>(b)$ et
  $\forall b''' \in <\alpha_0, \ldots, \alpha_{i-2}, \alpha_{i+2}, \ldots, \alpha_n>(b')$,
  $b''\alpha'_i = b''\phi$ et $b'''\alpha'_i = b'''\phi^{-1}$,
- $\forall b'' \notin <\alpha_0, \ldots, \alpha_{i-2}, \alpha_{i+2}, \ldots, \alpha_n>(b) \cup <\alpha_0, \ldots, \alpha_{i-2}, \alpha_{i+2}, \ldots, \alpha_n>(b')$, $b''\alpha'_i = b''\alpha_i$.

De ce fait, pour réaliser cette opération, nous disposons d'une opération prenant en paramètre un brin de chaque orbite. Ainsi les orbites des cellules à coudre sont parcourues en reliant chacun des brins qui les composent.

Une autre particularité des G-Cartes est de pouvoir couvrir seulement une partie de l'espace dans lequel elles sont définies ou la totalité de celui-ci. On parle alors respectivement de G-Cartes avec ou sans bord (cf. FIG. 2.3).

DÉF. 4 – Soit $G = (B, \alpha_0, \alpha_1, \ldots, \alpha_n)$ une n-G-Carte. G est sans bord si et seulement si $\forall i \in [0, n]$ et $\forall b \in B, \alpha_i(b) \neq b$.

Par la suite, nous travaillons essentiellement avec des subdivisions de l'espace. Pour cela nous utilisons des G-Cartes sans bord qui nous permettent de simplifier les algorithmes en ne traitant pas les cas particuliers liés aux bords.



(a) 2-G-Carte avec bord (b) 2-G-Carte sans bord

FIG. 2.3 – Exemple de G-Carte avec et sans bord.

2.1.2 Plongements

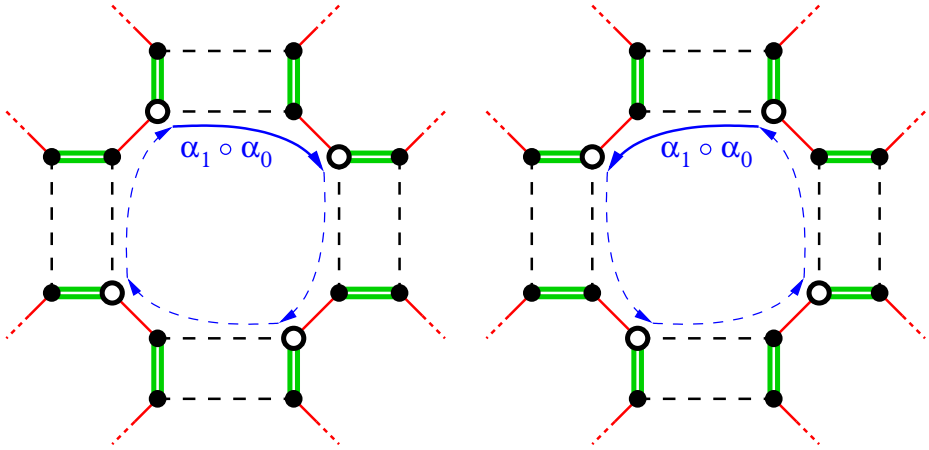
Les G-Cartes ne représentent que la topologie des objets et non leur forme géométrique. Cependant, sans information géométrique, une G-Carte n'est pas représentable car il faut pouvoir positionner ses brins dans l'espace. Pour cela, nous associons aux brins ou bien aux orbites des informations qui peuvent être de différentes natures (coordonnées cartésiennes, couleurs, équations, etc.).

Ces informations portent le nom de plongements et sont la plupart du temps associées aux différentes cellules d'une G-Carte. Par exemple, afin d'afficher une 3-G-Carte, nous associons à chacun des sommets (orbites $<\alpha_1, \alpha_2, \alpha_3>$) des coordonnées cartésiennes. Dans notre cas, ces informations suffisent car nous travaillons avec des objets polyédriques. Ainsi, nous pouvons retrouver l'équation d'une droite associée à une arête à l'aide des plongements sommets qu'elle possède à chacune de ces deux extrémités. Nous pouvons de même retrouver l'équation du plan d'une face en utilisant l'ensemble des sommets la composant. Cependant, si nous voulions traiter des objets surfaciques, nous pourrions associer (en plus des sommets), des courbes aux arêtes (orbites $<\alpha_0, \alpha_2, \alpha_3>$) et des surfaces aux faces (orbites $<\alpha_0, \alpha_1, \alpha_3>$).

2.1.3 Parcours et orientation

Les G-Cartes permettant de représenter des objets non-orientables, la structure n'impose pas un sens de parcours unique sur les cellules. Cependant, nous avons besoin dans nos algorithmes de respecter un sens de parcours sur les objets traités. Un sens de parcours sur un objet se définit par des déplacements depuis un brin donné à l'aide de trois compositions d'involutions qui sont $\alpha_1 \circ \alpha_0$, $\alpha_2 \circ \alpha_1$ et $\alpha_2 \circ \alpha_0$ en dimension 2. Ceci se généralise aisément en dimension 3 et supérieure.

Il est alors possible d'associer une orientation à un objet en lui associant un sens de parcours. En effet, le fait de parcourir une face dans un certain sens permet de définir la direction de la normale à cette face. Si nous associons un repère orthonormé (O, x, y, z) à une face de sorte que le plan (O, x, y) soit coplanaire au plan de la face, un parcours de la face dans le sens trigonométrique définit une normale dirigée dans le même sens que Oz tandis qu'un parcours dans le sens horaire définit une normale dirigée dans le sens contraire.



(a) brins mal orientés (b) brins correctement orientés

FIG. 2.4 – Recherche des brins respectant l'orientation d'une face.

Réciproquement, si nous définissons une normale à une face, nous pouvons déterminer les brins « correctement orientés » de la face. Pour cela, il suffit de considérer un brin b de la face et de la parcourir à l'aide de compositions d'involutions $\alpha_1 \circ \alpha_0$. Si le parcours s'effectue dans le sens trigonométrique par rapport à la normale définie, les brins parcourus sont dit « correctement orientés ». Dans le cas contraire les brins recherchés correspondent à ceux parcourus de la même manière depuis le brin $\alpha_0(b)$ (cf. FIG. 2.4).

2.1.4 Utilisation des G-Cartes

Pour utiliser la structure de données des n-G-Cartes, nous devons définir un certain nombre de commandes de bases. Nous définissons donc les types `Brin`, `Orbite` et `Marque` ainsi que les différentes opérations. Nous avons tout d'abord les opérations de base permettant de coudre et découdre les brins ou de se déplacer de brin en brin :

`CréerBrin → Brin` Fonction permettant de créer un nouveau brin isolé dans la G-Carte.

`SupprimerBrin(`b `: Brin)` Procédure permettant de supprimer un brin en le retirant de la G-Carte.

α_i`(`b `: Brin) → Brin` Fonction renvoyant l'image du brin b par l'involution α_i, $i \in [0, n]$.

`Coudre(`b_1, b_2 `: Brin,` α_i `: Involution)` Procédure permettant de coudre les brins b_1 et b_2 par l'involution α_i, $i \in [0, n]$.

`Découdre(`b `: Brin,` α_i `: Involution)` Procédure permettant de découdre le brin b de son image par l'involution α_i, $i \in [0, n]$.

`SurOrbite(`b `: Brin,` O `: Orbite) → booléen` Fonction indiquant si le brin b appartient à l'orbite O.

Pour effectuer certains types de parcours ou bien simplement pour différentier certains brins des autres, nous avons besoin de les marquer. Pour cela, nous définissons les opérations suivantes :

`Marquer(`O : `Orbite`, Ⓜ : `Marque)` Procédure permettant de marquer tous les brins de l'orbite O avec la marque Ⓜ. Si O se limite à un brin, seul ce dernier est marqué.

`MarquerSi(`O : `Orbite`, Ⓜ, Ⓢ : `Marque)` Procédure équivalente à la précédente qui ne marque par Ⓜ que les brins déjà marqués par Ⓢ.

`Démarquer(`O : `Orbite`, Ⓜ : `Marque)` Procédure permettant de démarquer tous les brins de l'orbite O de la marque m. Si O se limite à un brin, seul ce dernier est démarqué.

`EstMarqué(`b : `Brin`, Ⓜ : `Marque)` $\rightarrow$ `booléen` Fonction indiquant si le brin b est marqué par la marque Ⓜ.

Pour finir, nous avons aussi besoin de stocker des informations géométriques ou autres dans les cellules de nos objets. Nous associons alors ces informations aux orbites de la G-Carte à l'aide des opérations suivantes :

`Attribuer(`O : `Orbite`, I : `Information)` Procédure permettant d'attribuer l'information I aux brins de l'orbite O.

`Récupérer(`O : `Orbite`, T : `Type)` $\rightarrow$ `Information` Fonction permettant de récupérer l'information de type T stockée dans l'orbite O. Si une telle information n'existe pas, la fonction renvoie `nil`.

`Retirer(`O : `Orbite`, T : `Type)` Procédure permettant de rétirer une information de type T stockée dans l'orbite O.

2.2 Structures géométriques

Comme la topologie ne suffit pas à représenter un objet dans l'espace, nous associons des plongements aux cellules de la G-Carte afin de lui donner une représentation géométrique (cf. section 2.1.2). Comme nous travaillons ici avec des objets polyédriques, nous plongeons les orbites sommets à l'aide de points.

Pour représenter un point, nous utilisons une structure `Point` contenant les coordonnées cartésiennes (x, y, z) de celui-ci dans l'espace. Dans notre cas, ces coordonnées sont de type `flottant`. Ainsi, si nous avons un point P dans un espace 3D, nous pouvons accéder à ses coordonnées de la manière suivante : $P.x$, $P.y$ et $P.z$. Dans nos algorithmes, nous avons aussi besoin de manipuler des vecteurs. Pour cela, nous utilisons une structure `Vecteur` équivalente à la structure `Point` et nous accédons à ses coordonnées de la même manière. Afin d'utiliser ces deux précédentes structures, nous définissons de plus les opérations suivantes :

`Point(`b : `Brin)` $\rightarrow$ `Point` Fonction renvoyant le plongement point associé à l'orbite sommet incidente au brin b. Ceci correspond à un appel de la fonction `Récupérer(`$<\alpha_1, \alpha_2, \alpha_3>(b)$, `Point)`.

`Plonger(`b : `Brin`, P : `Point)` Procédure permettant de plonger le sommet incident au brin b avec le point P. Ceci correspond à l'appel de la procédure `Attribuer(`$<\alpha_1, \alpha_2, \alpha_3>(b)$, P`)`.

`Vecteur(`P_1, P_2 : `Point)` $\rightarrow$ `Vecteur` Fonction renvoyant le vecteur formé par les points P_1 et P_2, c'est-à-dire le vecteur $\overrightarrow{P_1P_2}$.

`Vecteur` (b : `Brin`) $\rightarrow$ `Vecteur` Fonctions renvoyant le vecteur correspondant à l'arête incidente au brin b. Ceci est équivalent à :

`Vecteur(Point(`b`)`, `Point(`$\alpha_0(b)$`))`

Pour les algorithmes en dimension 3, nous avons aussi besoin de manipuler des plans. Nous disposons alors d'une structure `Plan` contenant une équation de la forme $a.x + b.y + c.z + d = 0$ où (a, b, c) représentent les coordonnées de la normale au plan et d représente sa position dans l'espace. Les principales opérations dont nous avons besoin pour cette structure sont les suivantes :

`Plan(`v : `Vecteur`, p : `Point)` $\rightarrow$ `Plan` Fonction renvoyant le plan de normale v passant par le point p.

`Normale(`P : `Plan)` $\rightarrow$ `Vecteur` Fonction renvoyant le vecteur normal au plan P. Ce vecteur correspond aux coefficients (a, b, c) du plan.

`Distance(`P : `Plan`, p : `Point)` $\rightarrow$ `flottant` Fonction renvoyant la distance entre le point p et sa projection orthogonale sur le plan P. Cette distance peut être positive ou négative en fonction du côté où se trouve p par rapport à l'équation de P.

`Intersection(`P : `Plan`, p : `Point`, v : `Vecteur)` $\rightarrow$ `Point` Fonction renvoyant le point d'intersection entre le plan P et la droite de vecteur directeur v passant par le point p. S'il n'existe pas d'unique solution, la fonction renvoie `nil`.

2.3 Structures annexes

Tout au long de l'algorithme, nous avons besoin d'autres structures de données permettant de simplifier les traitements. Les principales sont les files et les listes. Pour cela, nous définissons un type `File` et un type `Liste` permettant de contenir des éléments génériques. Les opérations permettant de manipuler ces deux structures sont les suivantes :

`Enfiler(`F : `File/Liste`, e : `Élément)` Procédure ajoutant un élément en queue de la file/liste F.

`Défiler(`F : `File/Liste)` $\rightarrow$ `Élément` Fonction récupèrant et retirant l'élément qui se trouve en tête de la file/liste F.

`Tête(`F : `File/Liste)` $\rightarrow$ `Élément` Fonction permettant de renvoyer l'élément se trouvant en tête de la file/liste F.

`Taille(`F : `File/Liste) → entier` Fonction permettant d'indiquer le nombre d'éléments se trouvant dans la file/liste F.

`Suivant(`L : `Liste`, e : `Élément) → Élément` Fonction permettant de récupérer l'élément succédant e dans la liste L.

`Précédent(`L : `Liste`, e : `Élément) → Élément` Fonction permettant de récupérer l'élément précédent e dans la liste L.

Chapitre 3

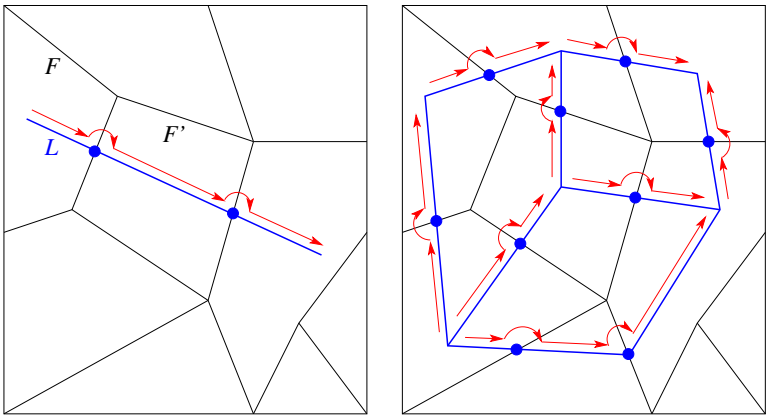
Co-raffinement de maillages en dimension 2

Le co-raffinement en dimension 2 a déjà fait l'objet de plusieurs travaux de recherche. Par exemple, les auteurs de [CD96, Caz97] proposent un algorithme de co-raffinement basé sur un système de réécriture et fonctionnant sur un modèle de cartes combinatoires [Lie91].

Le principal problème posé par le co-raffinement réside dans la recherche des points d'intersection. En effet, cette phase étant la plus importante de l'algorithme et la plus coûteuse en temps, elle doit donc être la plus optimisée possible. Il existe de nombreux algorithmes permettant d'optimiser ce calcul d'intersection qui sont la plupart du temps basés sur une décomposition de l'espace. Les auteurs de [dBDDS97] proposent une décomposition de l'espace en trapèzes construite incrémentalement en insérant les segments les uns après les autres. Ceux de [RF00] effectuent pour leur part une décomposition de l'espace en simplexes dont chacun est formé d'un segment et d'un point d'origine. Pour finir, une des méthodes les plus employées consiste à trier lexicographiquement les sommets des polygones dans l'espace pour ensuite pouvoir les balayer à l'aide d'une ligne [Caz97, GHPT89, Ž00]. Dans chacun des cas précédents, ces décompositions permettent de limiter les tests d'intersection à certaines zones de l'espace en éliminant tous les segments ne s'y trouvant pas.

Les auteurs de [NP82] proposent une méthode de co-raffinement incrémentale dont chaque étape consiste en l'ajout d'une courbe de bezier à l'espace de travail afin de le décomposer en plusieurs régions. Cette décomposition de l'espace leur permet d'accélérer la recherche des intersections en se propageant de région en région lors de l'insertion d'une nouvelle courbe. Il leur suffit de déterminer la région contenant le point de départ de la courbe, puis de localiser les intersections avec le bord de cette région afin de déterminer la région voisine dans laquelle la courbe continue son parcours. L'avantage de cette technique par rapport à celles citées précédemment est qu'elle ne nécessite pas de prétraitement des données comme par exemple un tri des éléments dans l'espace 2D.

Nous présentons ici un algorithme s'appuyant sur cette méthode mais qui est capable comme pour [CD96, Caz97] de traiter des subdivisions complètes. Pour des raisons de simplicité, notre algorithme travaille avec des segments de droites et non des courbes. Pour la strucutre de



(a) le long d'une ligne (b) le long d'un graphe

FIG. 3.1 – Propagation dans les faces d'un maillage.

données, nous avons choisi de travailler avec des *Cartes Généralisées* (cf. section 2.1) qui nous apportent un contrôle total sur la topologie.

Dans ce chapitre, nous commençons par présenter la méthode en section 3.1 en donnant le principe de fonctionnement de l'algorithme et le raisonnement que nous avons suivi pour le mettre en œuvre. Nous donnons ensuite la description complète de celui-ci en section 3.2 en expliquant en détail chacune des étapes. Nous finissons par donner des résultats et des temps de calcul en section 3.3.

3.1 Présentation de la méthode

Nous présentons ici la méthode utilisée pour co-raffiner deux maillages entre eux. Nous ne présentons que succinctement les différentes étapes de l'algorithme ainsi que le raisonnement suivi. Le détail complet de l'algorithme est donné en section 3.2.

3.1.1 Principe de fonctionnement

L'algorithme par propagation permet de limiter le nombre de tests d'intersections entre les segments de deux maillages distincts. Il s'appuie sur le fait qu'il est possible, étant donné un maillage M et une ligne L, de déterminer quelles sont les faces de M rencontrées par L et ceci de manière locale sans devoir tester tous les segments de M. Pour ce faire, nous commençons par détecter la position d'une extrémité de la ligne L par rapport aux faces du maillage M. Nous trouvons alors une face F depuis laquelle nous allons commencer le parcours. Ensuite, nous cherchons quelle est la face F' adjacente à F par laquelle passe la ligne L en cherchant la plus proche arête de F intersectant L. Nous pouvons alors passer dans la face F' puis continuer le parcours de la même manière jusqu'à atteindre l'autre extrémité de L (cf. FIG. 3.1(a)).

De cette manière, nous pouvons remarquer qu'à chaque étape du parcours, nous avons une recherche d'intersection dont la complexité dépend du degré de la face courante. De plus, nous pouvons aussi nous rendre compte que dans cette méthode, seules les faces intersectées du maillage M sont testées. Ainsi, nous obtenons un algorithme dont la complexité dépend du nombre de faces intersectées et du degré moyen de ces faces.

Il a été démontré [BY95] qu'un arrangement de n droites dans le plan génère n^2 segments (bornés par les points d'intersection entre les droites). De plus, le *théorème de la zone* affirme que le nombre total de segments bordant les faces intersectées par une nouvelle droite est linéaire. Ainsi, le calcul des points d'intersection entre cette nouvelle droite et les autres possède une complexité en $\mathcal{O}(n)$. De plus, un arrangement de droites peut être assimilé à un maillage 2D et réciproquement dans le cas où les faces du maillage possèdent des degrés similaires variant entre 3 et 6. Ceci étant généralement le cas [BF97], nous considérons dans la suite de ce chapitre que nous travaillons avec de tels maillages. De ce fait, si nous considérons un maillage composé de n segments (*i.e.* $\sqrt{n}$ droites), nous pouvons estimer que la complexité moyenne du calcul d'intersection entre une droite et les segments de ce maillage peut être exprimée en $\mathcal{O}(\sqrt{n})$.

Il est possible d'étendre très facilement ce concept de propagation en utilisant un graphe à la place d'une ligne. Pour cela, il suffit de traiter chaque liaison se trouvant entre deux noeuds du graphe de la même manière que pour une ligne. Pour traiter chaque liaison une seule et unique fois, nous effectuons un parcours en largeur du graphe en partant d'un noeud donné et en nous propageant (cf. Fig. 3.1(b)). Comme précédemment, la position du premier noeud par rapport aux faces du maillage doit être déterminée. Ensuite, les positions des autres noeuds sont déterminées automatiquement par l'algorithme. En effet, les nouveaux noeuds atteints étant des extrémités de liaisons ayant été parcourues, nous connaissons les dernières faces rencontrées par ces liaisons.

Un maillage étant composé de sommets et d'arêtes reliant ces derniers, nous pouvons considérer qu'il s'agit d'un graphe. Ainsi, nous associons les noeuds du graphe aux sommets du maillage et les arcs du graphe aux segments du maillage. Nous pouvons alors parcourir un maillage de la même manière qu'un graphe et de cette façon, nous obtenons un algorithme permettant de détecter les intersections entre les arêtes de deux maillages.

Du point de vue de la complexité, le calcul d'intersection entre deux maillages correspond à effectuer l'intersection de chacun des segments du premier maillage avec ceux du second. Comme nous l'avons vu précédemment, un arrangement de n droites dans le plan génère n^2 segments. Nous pouvons alors estimer qu'un maillage composé de n segments correspond approximativement à un arrangement de $\sqrt{n}$ droites. Ainsi, la complexité moyenne du calcul d'intersection entre deux maillages composés de n segments peut être exprimée en $\mathcal{O}(n)$ (*i.e.* $\mathcal{O}(\sqrt{n} \times \sqrt{n})$).

3.1.2 Hypothèses

Même si le principe de fonctionnement de l'algorithme semble simple, il doit gérer un très grand nombre de cas et peut devenir vite très compliqué. Pour notre étude, nous avons donc décidé de poser certaines hypothèses afin de simplifier l'algorithme.

Tout d'abord, nous avons choisi de travailler sur un modèle de 2-G-Cartes. Ce modèle nous permet de définir la topologie des objets de manière précise quelle que soit la dimension et

nous permet de connaître toutes les incidences et adjacences existantes entre les cellules de nos maillages. Nous avons ensuite décidé de travailler uniquement avec des plongements linéaires[1] et de ne pas traiter le cas des plongements courbes[2]. De plus, nous avons supposé que les maillages traités étaient géométriquement corrects et ne possédaient pas de défauts tels des auto-intersections[3], des arêtes doubles[4] ou bien des arêtes de longueur nulle.

La version de l'algorithme détaillée dans ce chapitre est capable de traiter le co-raffinement entre deux composantes connexes représentées par des maillages sans bords (cf. section 2.1.1). À chaque niveau de l'algorithme nous donnons une explication des méthodes employées et les endroits sensibles aux erreurs numériques. Le but de cet algorithme n'est pas de résoudre tous les problèmes numériques mais nous verrons tout de même en section 3.2.1.2 comment il est possible d'éviter certains d'entre eux.

3.1.3 Raisonnement

Afin de réaliser un tel algorithme, il est important de commencer par analyser les cas pouvant être rencontrés et la manière dont nous pouvons découper l'algorithme. Comme nous l'avons vu précédemment (cf. section 3.1.1), pour détecter les intersections entre les arêtes de deux maillages, nous devons parcourir l'un d'entre eux et nous propager dans les faces de l'autre. Pour nous propager dans le second maillage, nous devons détecter et créer les intersections avec les arêtes du premier maillage. De plus, pour débuter l'algorithme, nous devons être capable de localiser le sommet de départ du premier maillage par rapport aux faces du second.

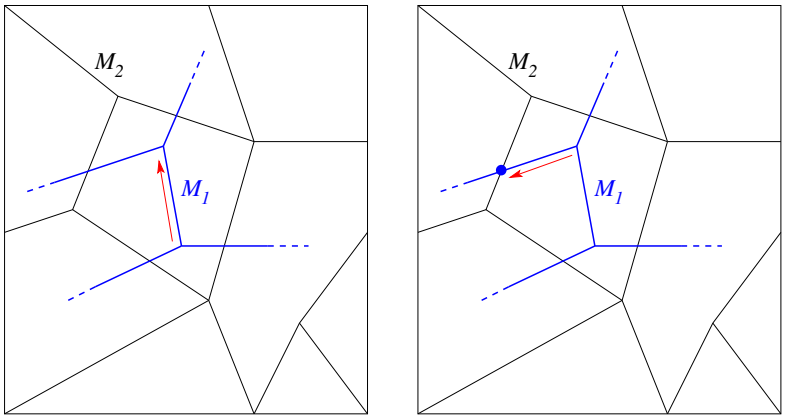
Nous pouvons alors séparer les traitements en quatre catégories qui sont le parcours, la détection d'une intersection, sa création et l'initialisation du parcours. Nous détaillons ici ces quatre étapes en considérant que nous parcourons les arêtes d'un maillage M_1 et que nous recherchons les intersections existantes avec le bord des faces d'un maillage M_2.

3.1.3.1 Parcours

Comme nous considérons le maillage M_1 comme un graphe, nous avons plusieurs solutions pour le parcourir. Nous avons décidé d'utiliser un parcours en largeur afin de simuler une propagation progressive. Nous partons donc d'un sommet de M_1 et nous parcourons chaque arête incidente à ce sommet jusqu'aux prochains sommets. Nous réitérons ensuite le processus à chaque nouveau sommet atteint (cf. Fig. 3.1(b)). À tout moment du parcours, nous devons de plus connaître la face de M_2 dans laquelle nous sommes. Pour cela, nous passons d'une face à une autre en testant si l'arête parcourue possède une intersection avec le bord de la face courante.

À chaque étape, nous pouvons alors nous trouver dans deux cas possibles. Soit l'arête courante ne possède pas d'intersection avec la face et dans ce cas nous devons passer au sommet suivant (cf. Fig. 3.2(a)). Soit elle en possède une et dans ce cas nous devons traiter cette intersection et passer dans la face adjacente (cf. Fig. 3.2(b)). Lorsque nous passons dans la face

[1] Coordonnées 2D associées aux sommets des maillages
[2] Courbes associées aux arêtes des maillages
[3] Arêtes d'un même maillage se recoupant
[4] Faces ayant seulement deux côtés



(a) aucune intersection trouvée (b) une intersection trouvée

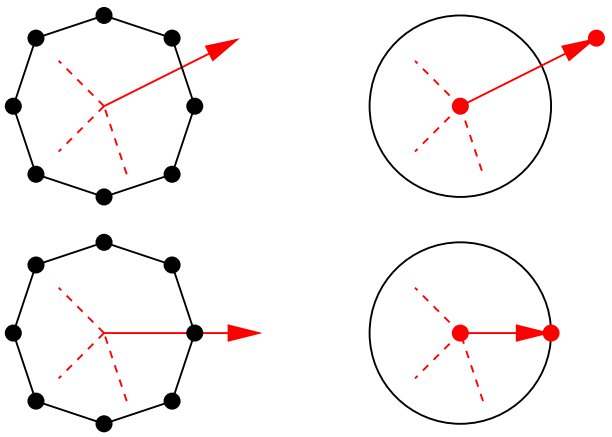
FIG. 3.2 – Parcours des arêtes du premier maillage.

adjacente, deux possibilités s'offrent à nous. Soit nous décidons de poursuivre notre chemin sur cette arête jusqu'au prochain noeud du maillage, soit nous considérons que comme nous avons éclaté l'arête en deux, nous obtenons un nouveau noeud et par conséquent, nous pouvons arrêter le parcours de l'arête et réitérer le processus sur ce nouveau sommet.

Si nous utilisons la première méthode lors du passage dans une face adjacente, nous devons parcourir les arêtes intégralement juqu'à ce qu'il n'y ait plus d'intersection avec les arêtes du second maillage. Cela implique donc d'effectuer une boucle supplémentaire pour traiter chaque arête. Cependant, si nous utilisons la deuxième méthode, nous traitons le reste de l'arête à l'aide de la boucle principale et aucun traitement supplémentaire n'est à effectuer. C'est d'ailleurs pour cette dernière méthode que nous optons dans notre algorithme.

Pour finir, il est important de noter que nous devons éviter de tester plusieurs fois les mêmes arêtes entre elles afin de ne pas effectuer des calculs inutiles. Pour cela, il suffit de marquer les arêtes du maillage M_1 déjà parcourues et de ne traiter par la suite que les arêtes non marquées. Nous devons aussi marquer les intersections déjà réalisées mais pour une autre raison. Ce dernier marquage nous est en fait utile pour différencier les sommets de M_1 provenant des noeuds d'origine du maillage, de ceux provenant des intersections. En effet, selon le sommet rencontré lors du parcours, la méthode pour connaître la face dans laquelle nous sommes diffère :

- lorsque nous arrivons sur un sommet d'origine du maillage et que celui-ci ne possède pas d'intersection avec des arêtes de M_2, nous devons nous souvenir de la dernière face de M_2 dans laquelle nous nous trouvions. Ainsi, pour continuer le parcours, nous testons chacune des arêtes incidentes à ce sommet avec le bord de cette face pour déterminer un point d'intersection ;
- si nous arrivons sur un sommet provenant d'une intersection, nous devons alors trouver pour chaque arête incidente à ce sommet, la face de M_2 dans laquelle nous nous dirigeons. Ceci se fait simplement en inspectant la topologie du sommet qui est censé représenter l'intersection entre les maillages M_1 et M_2 (cf. section 3.1.3.3). Ainsi, nous obtenons directement la face de M_2 dans laquelle se trouve l'arête parcourue.



(a) sur une face (b) sur une arête

FIG. 3.3 – Cas d'intersections pouvant être détectés.

3.1.3.2 Détection d'une intersection

Lors du parcours, nous avons besoin de détecter quelle partie de la face est coupée par l'arête que nous suivons. Nous pouvons d'ailleurs remarquer que le sommet d'origine de cette arête a obligatoirement déjà été traité. En effet, si nous sommes dans une face, c'est que nous sommes arrivés dans celle-ci en suivant une arête. Par conséquent, le sommet sur lequel nous sommes provient soit d'une intersection avec le bord de cette face, soit ce sommet est un noeud du maillage M_1 correspondant à l'autre extrémité de l'ancienne arête suivie. Dans les deux cas, nous pouvons en conclure que la prochaine intersection ne peut se produire qu'avec l'arête privée de sa première extrémité.

Nous pouvons maintenant différencier les cas que nous sommes amenés à traiter. Ces cas dépendent de la position de l'intersection sur le bord de la face et de sa position sur l'arête parcourue (cf. FIG. 3.3). En ce qui concerne la face, l'intersection peut se produire soit sur un sommet, soit sur une arête. Pour l'arête, l'intersection peut se trouver soit sur l'arête, soit sur sa deuxième extrémité. Nous devons maintenant traiter chacun de ces cas en construisant l'intersection associée.

3.1.3.3 Création d'une intersection

En fonction de la position du point d'intersection dont nous avons discuté précédemment, nous obtenons différents types d'intersections entre deux arêtes (cf. FIG. 3.4). Nous pouvons être confrontés soit à une intersection franche entre deux arêtes, soit à une arête passant par un sommet ou bien soit à deux sommets confondus. Il existe de plus un dernier cas se produisant lorsque deux arêtes sont partiellement ou complètement confondues. Les traitements à effectuer pour chacun de ces cas sont les suivants :

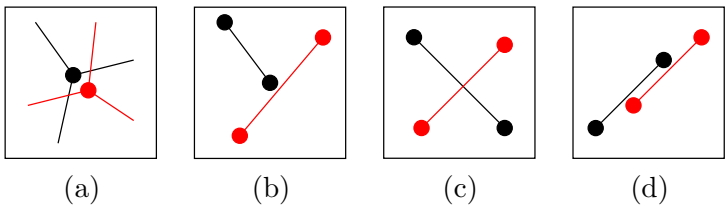


FIG. 3.4 – Les différents types d'intersection possibles : (a) sommets confondus ; (b) intersection entre un sommet et une arête ; (c) intersection franche ; (d) arêtes partiellement confondues.

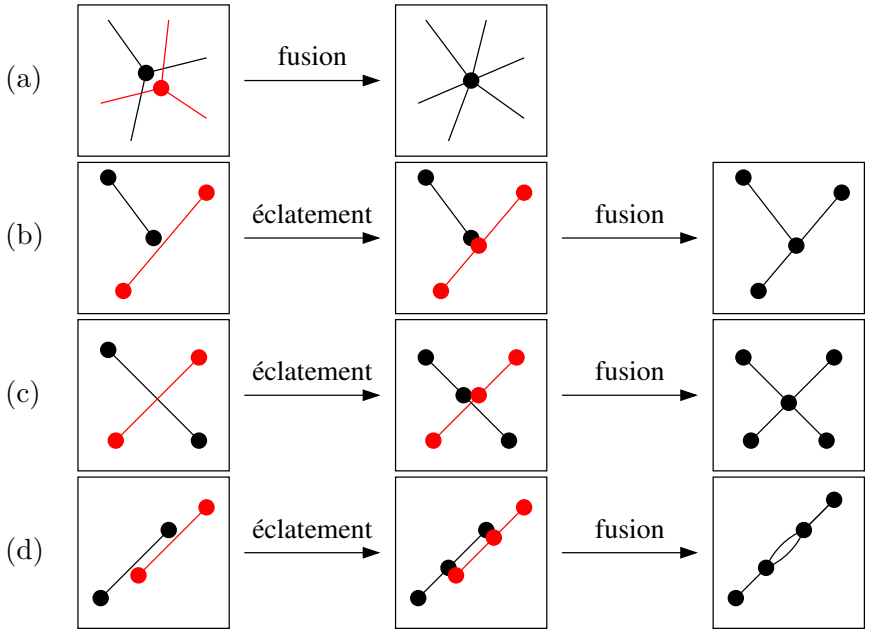


FIG. 3.5 – Créations des différentes intersections possibles.

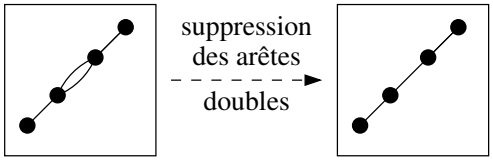


FIG. 3.6 – Suppression d'une arête double par post-traitement.

a. Lorsque deux sommets sont confondus, nous devons simplement les fusionner en triant de manière angulaire les arêtes qui leurs sont incidentes (cf. Fig. 3.5(a)).

b. Lorsque nous sommes dans le cas d'une intersection entre une arête et un sommet, nous devons couper l'arête en deux et insérer chacune des parties entre les arêtes incidentes au sommet. Nous pouvons alors remarquer qu'en éclatant l'arête, nous obtenons un nouveau sommet qui doit être confondu avec l'autre. Il est donc possible d'utiliser le même traitement que pour deux sommets confondus (cf. Fig. 3.5(b)).

c. Lorsque nous sommes dans le cas d'une intersection franche entre deux arêtes, nous devons couper en deux chacune d'entre elles pour pouvoir les relier au point d'intersection. Ici, nous pouvons traiter ce cas comme précédemment en éclatant les arêtes et en fusionnant les sommets résultants (cf. Fig. 3.5(c)).

d. Lorsque deux arêtes sont confondues, nous devons les fusionner. Selon la partie qui est commune aux deux arêtes, nous avons besoin d'insérer les sommets de l'une sur l'autre. Ceci peut donc être réalisé en utilisant la même technique que précédemment, c'est-à-dire en éclatant les arêtes et en fusionnant les sommets résultants (cf. Fig. 3.5(d)). Cependant, cette technique génère dans le maillage une arête double qui n'est pas désirée. Dans ce cas, nous pouvons effectuer un post-traitement qui se charge de supprimer toutes les arêtes doubles d'un maillage (cf. Fig. 3.6).

En résumé, tous les cas d'intersection pouvant être rencontrés lors du traitement peuvent être gérés de la même manière. Il suffit d'éclater les arêtes si nécessaire et ensuite de fusionner les sommets confondus. Ainsi, nous favorisons la simplicité algorithmique au détriment d'un léger surcoût en temps de calcul. En effet, la fusion de sommet consistant en un tri angulaire d'arêtes, elle n'est pas justifiée dans le cas des autres types d'intersection dont on connaît la configuration des arêtes. Cependant, le traitement de tous ces cas particuliers rend l'algorithme beaucoup plus compliqué et ne favorise pas son extension en dimension supérieure.

3.1.3.4 Initialisation du parcours

Pour démarrer le parcours le long des arêtes du premier maillage M_1 et nous propager dans les faces du second maillage M_2, nous avons besoin de localiser le point de départ P du parcours sur M_1 par rapport aux faces de M_2. Il existe de nombreuses techniques permettant d'effectuer ce traitement. La plus simple à mettre en œuvre consiste à tester pour chaque face de M_2 si elle contient P. Cependant, cette méthode engendre un algorithme dont la complexité est en $\mathcal{O}(n)$, n correspondant aux nombre de faces contenues dans le maillage M_2.

Il existe une méthode beaucoup plus efficace permettant d'obtenir le même résultat. En utilisant le principe de propagation précédemment expliqué, nous pouvons trouver la face contenant P en ne testant qu'un faible nombre de faces de M_2. Comme nous l'avons vu précédemment (cf. section 3.1.1), lorsque nous atteignons un sommet du maillage M_1 à la suite du suivi d'une arête, nous connaissons dans quelle face de M_2 se trouve ce sommet. Nous pouvons donc déterminer la position de P en utilisant exactement la même méthode.

Le problème ici est alors toujours le même car pour effectuer un tel traitement, nous devons partir d'un point de départ dont nous connaissons la localisation par rapport aux faces du

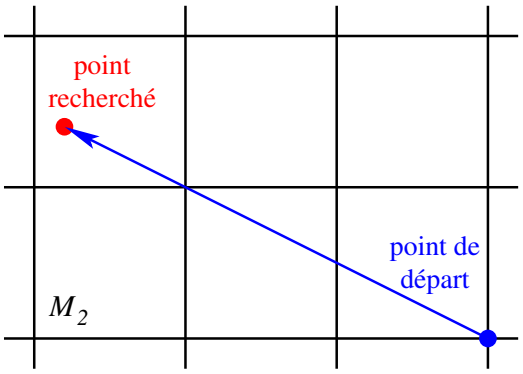


FIG. 3.7 – Localisation d'un sommet dans un maillage.

maillage M_2. L'idée permettant de résoudre ce problème est alors de prendre comme point de départ un sommet du maillage M_2 et de suivre une ligne droite L jusqu'au point P recherché (cf. FIG. 3.7).

Ainsi, comme le point de départ se trouve sur M_2, nous connaissons automatiquement sa position par rapport aux faces de M_2. De cette manière, pour détecter le point P, seules les faces intersectées par la ligne L sont testées.

3.2 Description de l'algorithme

Dans cette partie, nous expliquons en détail les différentes parties de notre algorithme. À chaque étape, nous discutons s'il y a lieu des choix effectués ainsi que des différents problèmes rencontrés. Nous commençons par donner la procédure principale de l'algorithme (cf. ALGO. 3.1) pour ensuite expliquer en détail chacune des sous-fonctions.

ALGO. 3.1 – Procédure `CoRaffiner`

Entrées : Deux brins b_1 et b_2 incidents à deux maillages.
Résultat : Le maillage résultant du co-raffinement des deux maillages.

début
  $\mathcal{F} \leftarrow$ `Initialiser`(b_1, b_2)
  `Parcourir`($\mathcal{F}$)
  `Nettoyer`(b_1)
  `SupprimerArêtesDoubles`(b_1)
fin

Tout au long de l'algorithme, nous travaillons avec une 2-G-Carte notée G et tous les brins manipulés appartiennent à cette dernière. Nous ajoutons à cette G-Carte une involution α'_1 correspondant à une copie de l'involution α_1 et qui nous permet de stocker les modifications topologiques à effectuer sans toucher à la topologie originale des objets.

L'algorithme principal effectue le co-raffinement de deux maillages distincts. La procédure correspondante prend alors en paramètre deux brins appartenant à chacun de ces deux maillages. Au final, si les deux maillages se chevauchent, les deux brins appartiennent au même maillage correspondant à la fusion des deux précédents.

Cette procédure effectue un parcours en largeur le long des arêtes du premier maillage en se propageant dans les faces de l'autre. Pour éviter d'obtenir un algorithme récursif, nous utilisons une file $\mathcal{F}$ dans laquelle nous stockons des couples de brins (b_s, b_f). Ces brins appartiennent respectivement à l'orbite du prochain sommet à parcourir et à l'orbite de la face dans laquelle effectuer la recherche d'intersection (cf. section 3.1.3.1).

Pour initialiser le parcours, nous devons commencer par localiser un sommet du premier maillage par rapport aux faces du second à l'aide de la procédure `Initialiser`. Une fois cette localisation effectuée, nous pouvons stocker le couple trouvé dans la file $\mathcal{F}$ et débuter le parcours.

Durant le parcours, nous devons nous souvenir des arêtes déjà parcourues et des intersections déjà réalisées. Pour cela, nous marquons les arêtes et les sommets du premier maillage avec deux marques différentes notées respectivement Ⓐ et Ⓢ. Les brins sont ensuite démarqués à la fin du parcours en effectuant un post-traitement consistant à nettoyer le maillage résultant. Ceci se fait à l'aide de la procédure `Nettoyer`.

Une fois le co-raffinement effectué, il est possible que le maillage résultant comporte des arêtes doubles. Pour corriger ce défaut, nous devons donc effectuer un autre post-traitement qui détecte et supprime toutes les arêtes doubles existantes. Ceci se fait grâce à l'appel de la procédure `SupprimerArêtesDoubles`.

Dans la suite, nous donnons la procédure principale `Parcourir` de l'algorithme puis nous revenons après sur l'initialisation et les post-traitements à effectuer.

3.2.1 Parcours

Afin de démarrer le parcours, nous devons connaître le sommet de départ du premier maillage M_1 depuis lequel nous partons et sa position par rapport aux faces du second maillage M_2. Cependant, nous savons que ceci est connu et déterminé par la fonction `Initialiser` (cf. section 3.2.2) qui insère un premier élément dans le file $\mathcal{F}$.

Ainsi, le parcours débute en récupérant le premier couple (s, f) de la file $\mathcal{F}$ contenant respectivement un sommet et une face. Ensuite, il consiste à se propager le long des arêtes incidentes au sommet s et à rechercher pour chacune d'entre elles la plus proche intersection avec le bord de la face f. Cependant, si le sommet s provient d'une intersection, la face f à parcourir doit être déterminée pour chacune des arêtes incidentes à s (cf. section 3.1.3.1).

À chaque étape, le couple retiré de la file est conservé dans une variable C jusqu'à ce que toutes les arêtes incidentes à s soient testées. Pour continuer, nous ajoutons à la file $\mathcal{F}$ le prochain sommet atteint (qu'il provienne d'une intersection ou non) ainsi que la face dans laquelle il se trouve. Nous répétons ce processus jusqu'à ce que la file soit vide. La figure 3.8 montre les premières étapes du parcours sur un exemple simple. L'algorithme 3.2 donne les traitements correspondants et se découpe de la manière suivante :

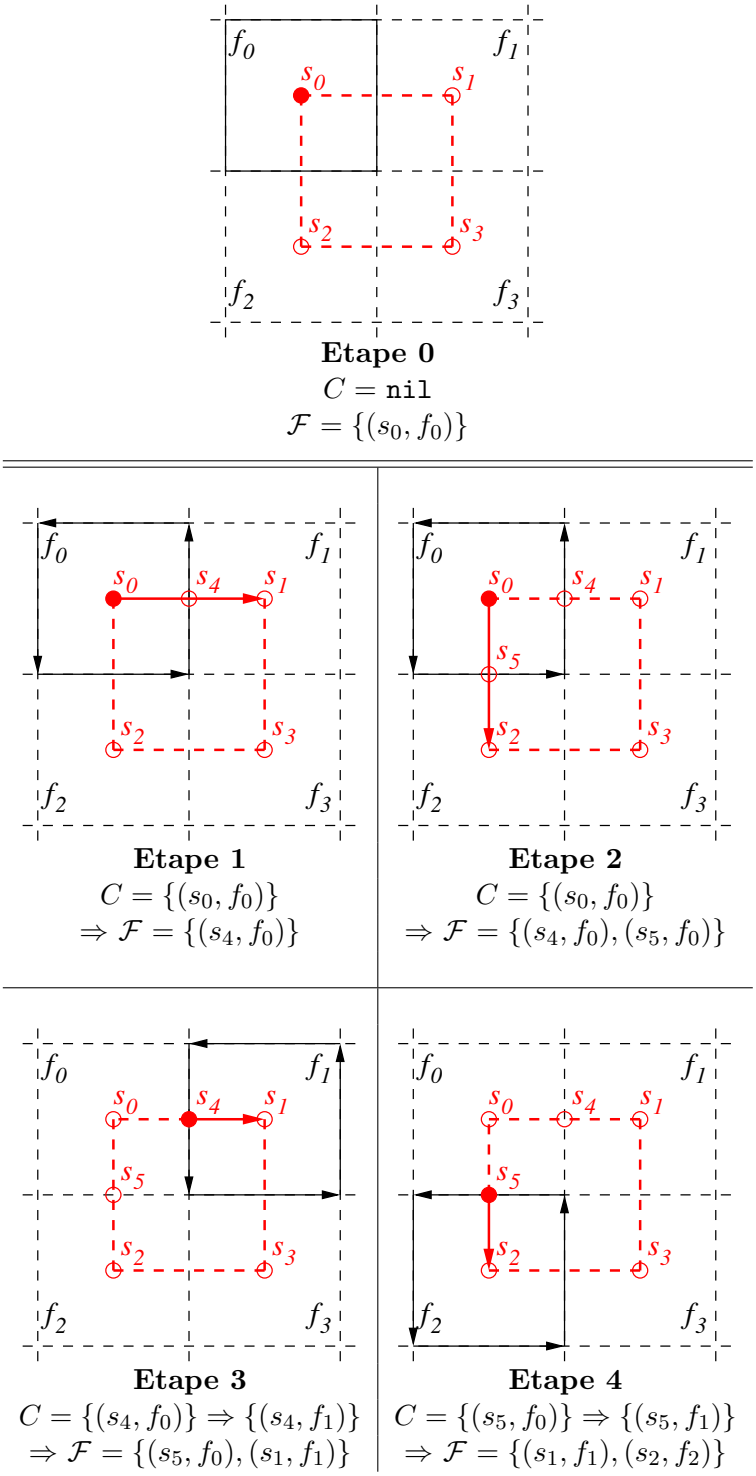


FIG. 3.8 – Premières étapes du parcours par propagation : à chaque étape, le couple $(sommet, face)$ est indiqué ainsi que le contenu de la file $\mathcal{F}$.

ALGO. 3.2 – Procédure `Parcourir`

Entrées : La file $\mathcal{F}$ contenant des couples de brins.
Résultat : L'ensemble des intersections calculées.

```
début
1   tant que F ≠ ∅ faire
        (b_s, b_f) ← Défiler(F)
        O ← Point(b_s)
        b_a ← b_s
2       répéter
            si EstMarqué(b_a, Ⓐ) ≠ vrai alors
                P ← Point(α_0(b_a))
3               si EstMarqué(b_a, Ⓢ) alors
                    b'_f ← RechercherFace(b_a)
                sinon
                    b'_f ← b_f
4               (b_c, dim, I, pos) ← RechercherIntersection(O, P, b'_f)
                si b_c ≠ nil alors
5                   si pos = SurArête alors
                        b'_a ← EclaterArête(b_a, I)
                    sinon
                        b'_a ← α_2(α_0(b_a))
6                   si dim = 1 alors
                        b'_f ← EclaterArête(b_c, I)
                    sinon
                        b'_f ← b_c
7                   si EstMarqué(b'_a, Ⓢ) ≠ vrai alors
                        Marquer(<α_1, α_2>(b'_a), Ⓢ)
                        FusionnerSommets(b'_a, b'_f)
                sinon
                    b'_a ← α_2(α_0(b_a))
8               Marquer(<α_0, α_2>(b_a), Ⓐ)
                Enfiler(F, (b'_a, b'_f))
            b_a ← α_2(α_1(b_a))
        jusqu'à b_a = b_s
fin
```

1. Nous avons tout d'abord une première boucle qui se charge de récupérer des couples de brins (b_s, b_f) correspondant à des couples $(sommet, face)$. Cette boucle permet d'effectuer un parcours en largeur du premier maillage en utilisant la file $\mathcal{F}$ qui est mise à jour à chaque nouveau noeud atteint. Pour chaque sommet récupéré, nous conservons son plongement associé (point O).

2. Pour chaque sommet récupéré précédemment dans la liste, nous traitons toutes les arêtes incidentes à ce sommet à l'aide d'une boucle qui parcourt les brins autour du sommet. Chaque arête est désignée par un brin noté b_a qui appartient au même sommet que b_s. Avant d'effectuer tout traitement, nous regardons si l'arête a déjà été traitée en testant si b_a est marqué par la marque Ⓐ. Si ce n'est pas le cas, nous récupérons alors le plongement associé à l'autre sommet (point P) de l'arête puis nous pouvons continuer les traitements avec cette arête.

3. Si b_a est marqué par la marque Ⓢ, le sommet de départ provient d'une intersection. Nous devons alors déterminer la face du second maillage qui est incidente à ce sommet et qui contient l'arête courante. Pour cela, nous faisons appel à la fonction `RechercherFace`. Si par contre b_a n'est pas marqué, nous gardons comme face celle récupérée dans la file $\mathcal{F}$.

4. Nous pouvons maintenant rechercher si l'arête courante possède une intersection avec la face en faisant appel à la fonction `RechercherIntersection`. Cette dernière nous fournit quatre valeurs (b_c, dim, I, pos) qui correspondent aux informations suivantes :
   - b_c : brin de la plus proche cellule intersectée,
   - dim : dimension de la plus proche cellule intersectée (0 ou 1),
   - I : plus proche point d'intersection entre l'arête et le bord de la face,
   - pos : information sur la position de l'intersection sur l'arête (`SurPremierSommet`, `SurSecondSommet`, `SurArête`, `HorsArête`).

   S'il existe une intersection (si b_c est défini), nous devons tester les autres valeurs fournies par la fonction `RechercherIntersection` afin de déterminer quels sont les traitements à effectuer pour créer l'intersection. Lorsque l'intersection est générée, nous obtenons un brin du sommet suivant. S'il n'y a pas d'intersection, nous prenons un brin de l'autre extrémité de l'arête.

5. Pour créer l'intersection, il nous faut effectuer les traitements détaillés dans la section 3.1.3.3. Si l'intersection se trouve au milieu de l'arête, nous devons éclater l'arête à l'aide de la fonction `EclaterArête` qui renvoie un brin du nouveau sommet créé. Si ce n'est pas le cas, l'intersection se trouve obligatoirement sur le second sommet de l'arête, il suffit alors de récupérer un brin de l'autre extrémité de l'arête. Nous obtenons ainsi un brin b'_a correspondant au premier sommet à fusionner.

6. Si la dimension de la cellule intersectée est 1, il s'agit d'une arête et il faut alors l'éclater comme précédemment. Sinon, la cellule intersectée est obligatoirement un sommet et il suffit de garder ce brin. Nous obtenons ici un brin b'_f du second sommet à fusionner.

7. Maintenant que nous avons deux brins b'_a et b'_f appartenant à deux sommets, il nous faut regarder si ces sommets ne proviennent pas d'une intersection qui a déjà été traitée. Pour cela, nous regardons si le brin b'_a du premier sommet est marqué par la marque Ⓢ. Si ce n'est pas le cas, nous pouvons marquer ce sommet comme traité et fusionner les deux

sommets à l'aide de la fonction `FusionnerSommets`. Notons que cette fonction ne modifie pas la topologie car elle utilise l'involution α'_1 pour coudre les brins.

8. À la fin, nous marquons la portion de l'arête déjà traitée à l'aide de la marque Ⓐ et nous ajoutons le couple (b'_a, b'_f) à la file $\mathcal{F}$.

Dans la suite de cette section, nous détaillons l'ensemble des sous-fonctions utilisées dans l'algorithme de parcours ainsi que quelques autres fonctions nécessaires.

3.2.1.1 Recherche de la face contenant une arête

Dans l'algorithme de parcours, nous avons besoin de savoir avec quelles faces du second maillage nous devons tester les arêtes du premier maillage. Nous avons vu précédemment que nous pouvions tomber dans deux cas de figure. Le premier est le cas où la première extrémité de l'arête provient d'une intersection et le deuxième cas est celui où ce sommet ne provient pas d'une intersection. Nous nous intéressons ici au premier cas en fournissant une méthode permettant de résoudre le problème (cf. Algo. 3.3) à moindre coût.

Algo. 3.3 – Fonction `RechercherFace`

Entrées : Un brin b d'une arête du premier maillage M_1.
Sorties : Un brin de la face du deuxième maillage M_2.
Pré-condition : b doit être marqué par la marque Ⓢ

```
début
    b' ← α2(b)
    tant que α'1(b') = α1(b') faire
        b' ← α2(α1(b'))
    retourner α'1(b')
fin
```

Étant donné que le premier sommet de l'arête provient d'une intersection, celui-ci est relié à un sommet du second maillage à l'aide de l'involution α'_1. Il nous est alors possible de retrouver le secteur angulaire du second sommet dans lequel se trouve l'arête à l'aide d'un simple parcours (cf. Fig. 3.9). Pour cela, nous devons parcourir les brins du sommet jusqu'à tomber sur un brin étant cousu par α'_1. Une fois ce brin trouvé, nous savons que le brin auquel il est relié par α'_1 appartient à l'autre maillage et qu'il correspond de plus à la face contenant notre arête.

3.2.1.2 Recherche de la plus proche intersection dans une face

Lorsque dans le parcours nous avons un couple formé d'une arête a et d'une face f, nous devons rechercher quelle est la plus proche intersection entre a et le bord de f. Pour cela, nous utilisons un algorithme qui se charge de trouver une éventuelle intersection et de nous fournir toutes les informations nécessaires la concernant.

Pour effectuer un tel traitement, la méthode la plus simple serait de parcourir le bord de la face f, de calculer les points d'intersection entre les arêtes de f et l'arête a et de garder la

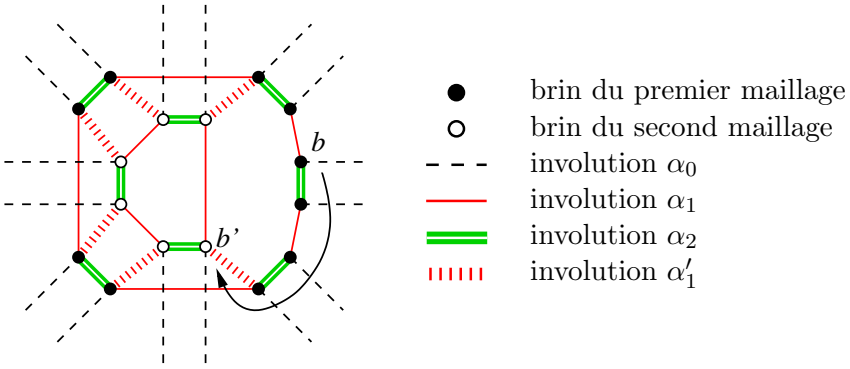


FIG. 3.9 – Recherche de la face contenant une arête à l'aide de la topologie

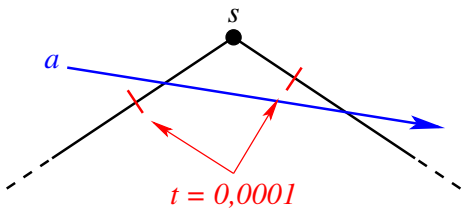


FIG. 3.10 – Problèmes de détection d'intersections avec des sommets liés aux erreurs numériques.

plus proche intersection. De plus, s'il existe une intersection et que son paramètre sur l'arête de f correspondante est proche de 0 ou de 1, il s'agit d'une intersection passant par un sommet, sinon il s'agit d'une intersection franche.

Cependant, cette méthode peut induire de nombreuses erreurs numériques lorsqu'il s'agit de déterminer si la cellule intersectée de la face est un sommet ou bien une arête. En effet, comme cette détection se base uniquement sur la valeur des paramètres de l'intersection sur les arêtes de la face, il se peut que pour deux arêtes adjacentes à un sommet s, le type de l'intersection soit reconnu différemment sur chacune d'entre elles. Nous pouvons par exemple considérer qu'une intersection est reconnue comme passant par un sommet si son paramètre sur une arête de la face est inférieur à une valeur fixée à 10^{-4}. De cette manière, il est possible que l'intersection soit détectée comme franche sur une des deux arêtes alors qu'elle peut être considérée comme passant par s pour l'autre arête (cf. FIG. 3.10). Ce problème est surtout dû au fait que les arêtes de la face ont une longueur quelconque et que pour un même paramètre, la distance au sommet est différente. Cependant, même pour deux arêtes de même longueur, le problème est toujours présent car les points d'intersection sur chaque arête ne se trouvent pas nécessairement à la même distance de s.

Pour résoudre ce problème, nous devons calculer la distance entre la droite et le sommet. Nous n'utilisons alors plus les paramètres sur les arêtes et nous éliminons ainsi les incohérences liés à cette détection. L'algorithme 3.4 propose une méthode permettant d'effectuer ces traitements et se découpe selon les étapes suivantes :

ALGO. 3.4 – Fonction `RechercherIntersection`

Entrées :
- les deux sommets A et B de l'arête parcourue ;
- un brin b_f de la face courante.

Sorties :
- un brin de la plus proche cellule intersectée ;
- la dimension de la plus proche cellule intersectée ;
- le point correspondant à la plus proche intersection ;
- une information sur la position de l'intersection sur l'arête.

début
- $m \leftarrow$ `Marque`
- $b \leftarrow b_f$
- $b_c \leftarrow$ `nil`
- $t \leftarrow 1$

1 **répéter**
- $P \leftarrow$ `Point`(b)
- $(pos', t') \leftarrow$ `LocaliserPointSurArête`(P, A, B)
- **si** $(pos' =$ `SurArête` **et** $t' < t)$ **ou**
- $(pos' =$ `SurSecondSommet` **et** $t' \leq t)$ **alors**
  - $b_c \leftarrow b$; $dim \leftarrow 0$; $pos \leftarrow pos'$; $t \leftarrow t'$
  - `Marquer`(b, m)
  - `Marquer`$(\alpha_0(\alpha_1(b)), m)$
- $b \leftarrow \alpha_1(\alpha_0(b))$

jusqu'à $b = b_f$

2 **répéter**
- **si** `EstMarqué`(b, m) **alors**
  - `Démarquer`(b, m)
- **sinon**
  - $P_1 \leftarrow$ `Point`(b)
  - $P_2 \leftarrow$ `Point`$(\alpha_0(b))$
  - $(pos', t_1, t_2) \leftarrow$ `LocaliserIntersectionArêtes`(P_1, P_2, A, B)
  - **si** $0 \leq t_1 \leq 1$ **et**
  - $((pos' =$ `SurArête` **et** $t_2 < t)$ **ou**
  - $(pos' =$ `SurSecondSommet` **et** $t_2 \leq t))$ **alors**
    - $b_c \leftarrow b$; $dim \leftarrow 1$; $pos \leftarrow pos'$; $t \leftarrow t_2$
  - $b \leftarrow \alpha_1(\alpha_0(b))$

jusqu'à $b = b_f$

3 **si** $b_c \neq$ `nil` **alors**
- $I \leftarrow A + t *$ `Vecteur`(A, B)

retourner (b_c, dim, I, pos)

fin

1. Nous commençons par effectuer un premier parcours de la face afin de chercher si des sommets de celle-ci se trouvent sur l'arête désignée par les points A et B. Pour cela, nous faisons appel à la fonction `LocaliserPointSurArête` qui nous indique la position du sommet sur l'arête ainsi que son paramètre. Nous testons ainsi tous les sommets et gardons le sommet le plus proche de A se trouvant sur l'arête. De plus, nous marquons toutes les arêtes incidentes aux sommets se trouvant sur l'arête $[AB]$. Ceci nous permet de ne pas les tester lors du deuxième parcours qui consiste à rechercher les intersections avec les arêtes de la face. En effet, si nous avons détecté une intersection avec un sommet sur la face, il est impossible qu'il y ait une intersection avec les arêtes incidentes à ce sommet.

2. Nous effectuons ensuite un deuxième parcours en cherchant les intersections existantes entre notre arête $[AB]$ et les arêtes de la face. Nous testons ici uniquement les arêtes qui ne sont pas marquées et nous conservons l'arête dont l'intersection avec $[AB]$ est la plus proche de A. Ceci se fait grâce à l'appel de la fonction `LocaliserIntersectionArêtes` qui nous indique la position de l'intersection sur $[AB]$ ainsi que son paramètre sur chacune des deux arêtes. Si un point d'intersection est plus proche qu'un point détecté lors du premier parcours, il est gardé, sinon il est rejeté.

3. Pour finir, nous calculons les coordonnées du point d'intersection le plus proche s'il existe et nous renvoyons un brin de la plus proche cellule intersectée en indiquant sa dimension, le point d'intersection et la position de ce point sur $[AB]$.

Localisation d'un point sur une arête

Lors de la détection des intersections, nous avons besoin de déterminer la position d'un point sur une arête. Nous utilisons alors l'algorithme 3.5 qui se charge de déterminer cette position. Cette algorithme a besoin de tester si deux points sont égaux et si un point se trouve sur une droite. Ces comparaisons sont faites à l'aide d'un calcul de distance au carré et renvoient `vrai` si le carré est inférieur à une valeur ε fixée.

Localisation du point d'intersection entre deux arêtes

Comme pour la localisation d'un point sur une arête, nous avons besoin de calculer le point d'intersection entre deux arêtes et d'analyser la position de ce point sur chacune d'entre elles. Pour cela, nous utilisons l'algorithme 3.6 qui se charge d'effectuer ces traitements. Cet algorithme commence par tester si les sommets de l'arête parcourue ne se trouvent pas sur l'arête intersectée et si ce n'est pas le cas, il calcule le point d'intersection entre les deux arêtes en testant si ce dernier se trouve bien sur l'arête parcourue.

3.2.1.3 Création d'une intersection

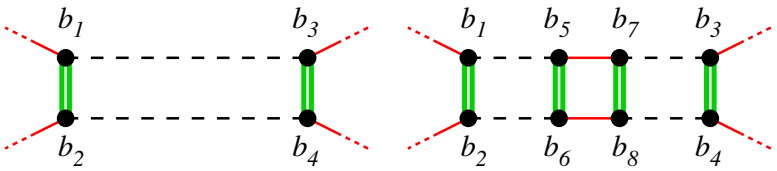
La création d'une intersection s'effectue en deux principales étapes. Il y a tout d'abord l'insertion d'un sommet sur une arête lorsqu'il s'agit d'une intersection franche et l'interclassement d'arêtes autour d'un sommet qui permet de fusionner deux sommets topologiques. Nous détaillons dans cette section le fonctionnement de ces deux algorithmes.

ALGO. 3.5 – Fonction `LocaliserPointSurArête`

Entrées : ▪ un point P à localiser ;
▪ deux points A et B définissant l'arête.

Sorties : ▪ la position de P ;
▪ le paramètre de P sur le segment $[AB]$.

début
  suivant la position de P **faire**
    cas où $P = B$ **alors retourner** (`SurSecondSommet`, 1)
    cas où $P = A$ **alors retourner** (`SurPremierSommet`, 0)
    cas où $P \in (AB)$ **alors**
      $t \leftarrow$ paramètre de P sur $[AB]$
      si $0 < t < 1$ **alors**
        retourner (`SurArête`, t)
      sinon
        retourner (`HorsArête`, t)
    autres cas alors retourner (`HorsArête`, -1)
fin



(a) arête d'origine (b) arête éclatée

FIG. 3.11 – Opération d'insertion d'un point sur une arête

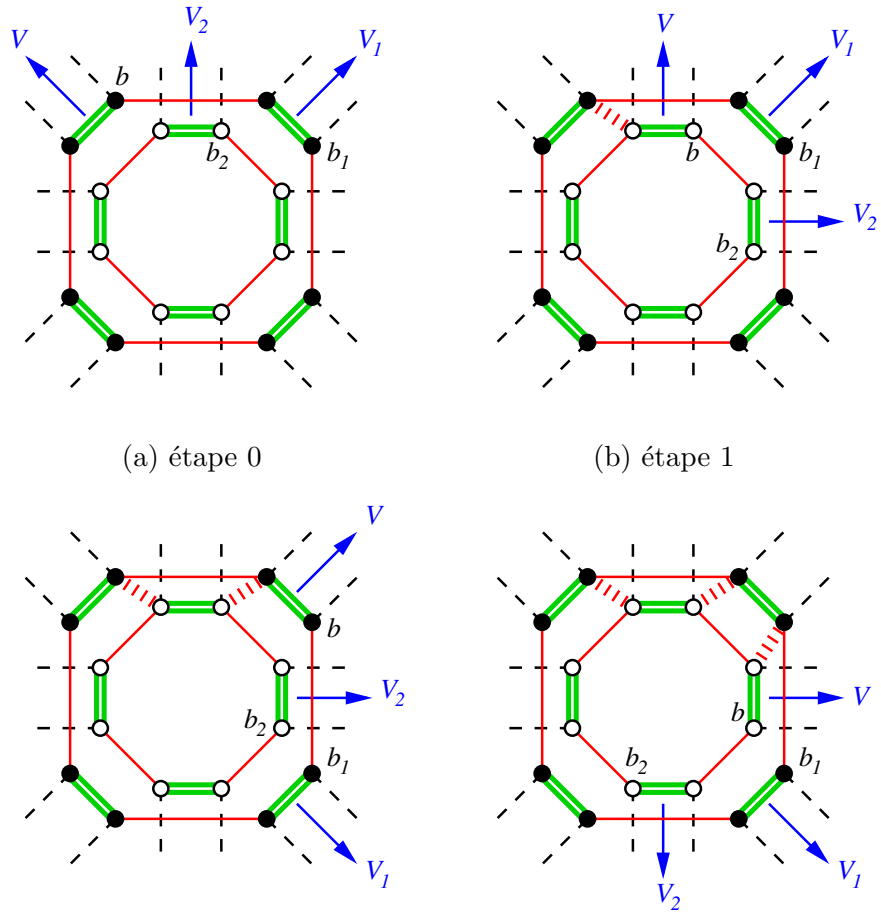
Insertion d'un sommet sur une arête

L'insertion d'un sommet sur une arête est une opération élémentaire qui consiste à ajouter quatre brins sur une arête en effectuant les coutures permettant de les relier topologiquement aux autres brins de l'arête. Cette opération est décrite dans l'algorithme 3.7 et nous pouvons en avoir une représentation sur la figure 3.11.

Interclassement d'arêtes autour d'un sommet

Pour réaliser les intersections et fusionner un sommet du premier maillage avec un sommet du deuxième maillage, nous avons besoin d'interclasser les arêtes incidentes aux deux sommets afin de n'en former plus qu'un. L'algorithme 3.8 propose une solution permettant d'effectuer cette opération et la figure 3.12 montre son principe de fonctionnement. L'algorithme se découpe en plusieurs étapes qui sont les suivantes :

1. Nous commençons par initialiser l'algorithme en prenant comme arête de référence une



(a) étape 0 (b) étape 1

(c) étape 2 (d) étape 3

FIG. 3.12 – Principe utilisé pour l'interclassement d'arêtes autour d'un sommet.

ALGO. 3.6 – Fonction `LocaliserIntersectionArêtes`

Entrées : ▪ deux points A et B définissant l'arête intersectée de la face ;
▪ deux points C et D définissant l'arête parcourue.

Sorties : ▪ la position de l'intersection sur l'arête $[CD]$;
▪ le paramètre de l'intersection sur l'arête $[AB]$;
▪ le paramètre de l'intersection sur l'arête $[CD]$.

début
  $(pos, t_1) \leftarrow$ `LocaliserPointSurArête(`D, A, B`)`
  si $pos \neq$ `HorsArête` **alors retourner** (`SurSecondSommet`, t_1, 1)
  $(pos, t_1) \leftarrow$ `LocaliserPointSurArête(`C, A, B`)`
  si $pos \neq$ `HorsArête` **alors retourner** (`SurPremierSommet`, t_1, 0)
  $V_1 \leftarrow$ `Vecteur(`A, B`)` ; $V_2 \leftarrow$ `Vecteur(`C, D`)`
  $\Delta \leftarrow V_1.y \times V_2.x - V_1.x \times V_2.y$
  si $\Delta = 0$ **alors**
    retourner (`HorsArête`, -1, -1)
  sinon
    $V_3 \leftarrow$ `Vecteur(`C, A`)`
    $t_1 \leftarrow (V_2.y \times V_3.x - V_2.x \times V_3.y)/\Delta$
    $t_2 \leftarrow (V_1.y \times V_3.x - V_1.x \times V_3.y)/\Delta$
    si $0 < t_2 < 1$ **alors**
      retourner (`SurArête`, t_1, t_2)
    sinon
      retourner (`HorsArête`, t_1, t_2)
fin

arête du premier sommet et en cherchant l'arête du second sommet qui doit se trouver juste après elle dans le sens horaire. Ceci se fait à l'aide de la fonction `RechercherSecteur` qui permet de nous renvoyer un brin du second sommet dont le secteur incident contient l'arête du premier sommet.

2. Ensuite, nous parcourons nos deux sommets en parallèle en testant à chaque itération et pour chaque sommet, quelle est l'arête suivante la plus proche de notre arête de référence. Nous pouvons alors être confronté à trois cas possibles.

3. Si l'arête de référence et l'arête suivante sur le second sommet sont confondues, nous ne pouvons pas déterminer si l'arête du premier sommet se trouve entre ces deux arêtes ou en dehors car elles sont colinéaires. Nous devons alors interclasser l'arête du second sommet ici et la prendre comme nouvelle référence (cf. étape 5).

4. Si les arêtes suivantes sur chacun des sommets sont confondues, nous cousons à l'aide de l'involution α_1' l'arête du second sommet avec l'arête de référence et l'arête du premier sommet avec l'arête du second. Nous passons à l'arête suivante sur chacun des sommets et notre arête de référence devient alors l'arête du premier sommet qui a été cousue.

ALGO. 3.7 – Fonction `EclaterArête`

Entrées : ▪ un brin b de l'arête à éclater ;
▪ un point P définissant la position du nouveau sommet.

Sorties : Un brin du sommet inséré.

début
  $b_1 \leftarrow b$; $b_2 \leftarrow \alpha_2(b)$; $b_3 \leftarrow \alpha_0(b_1)$; $b_4 \leftarrow \alpha_0(b_2)$
  `Découdre`(b_1, α_0) ; `Découdre`(b_2, α_0)
  pour $i \in [5..8]$ **faire** $b_i \leftarrow$ `CréerBrin`
  `Coudre`(b_5, b_6, α_2) ; `Coudre`(b_7, b_8, α_2)
  `Coudre`(b_5, b_7, α_1) ; `Coudre`(b_6, b_8, α_1)
  pour $i \in [1..4]$ **faire** `Coudre`(b_i, b_{i+4}, α_0)
  `Plonger`(b_5, P)
  retourner b_7
fin

5. Si l'arête suivante sur le second sommet se trouve dans le secteur formé par l'arête de référence et l'arête suivante sur le premier sommet, nous devons interclasser l'arête du second sommet car il s'agit de la plus proche. Nous cousons alors l'arête du second sommet avec l'arête de référence. L'arête de référence devient alors cette nouvelle arête et nous pouvons passer à l'arête suivante sur le second sommet.

6. Si aucun des deux cas précédents n'est vérifié, l'arête la plus proche n'est autre que l'arête suivante sur le premier sommet. Comme précédemment, nous devons donc la coudre avec l'arête de référence, prendre cette arête comme nouvelle référence et passer à l'arête suivante sur le premier sommet.

Recherche d'un secteur angulaire contenant un vecteur

Pour interclasser des arêtes autour d'un sommet, nous avons besoin de rechercher le secteur angulaire autour d'un sommet contenant un vecteur. Pour cela, nous devons parcourir les secteurs incidents à un sommet et tester pour chacun d'entre eux s'il contient le vecteur (cf. ALGO. 3.9).

Afin de tester si un vecteur se trouve dans un secteur angulaire, nous faisons appel à la fonction `VecteurDansSecteur`. Cette fonction est détaillée dans l'algorithme 3.10 qui se charge de tester la position d'un vecteur par rapport à deux autres vecteurs définissant un secteur. Ce test est effectué en analysant simplement le signe de la composante en z du produit vectoriel entre les vecteurs. Ici, nous devons tenir compte du fait que le secteur angulaire peut être aigu ou obtus sachant que les vecteurs du secteur forment un repère direct (cf. FIG. 3.13).

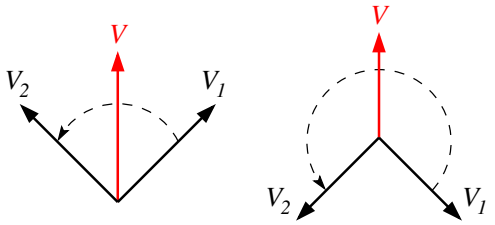
3.2.2 Initialisation du parcours

Maintenant que nous connaissons le principe de fonctionnement de l'algorithme, intéressons nous plus particulièrement à son initialisation. Pour débuter le parcours, nous devons partir d'un

ALGO. 3.8 – Fonction `Interclasser`

Entrées : Deux brins s_1 et s_2 appartenant à deux sommets distincts.
Résultat : s_1 et s_2 sont reliés topologiquement à l'aide de l'involution α'_1.

début
1 $b \leftarrow s_1$; $V \leftarrow$ `Vecteur(`b`)`
$b_1 \leftarrow \alpha_2(\alpha_1(s_1))$; $V_1 \leftarrow$ `Vecteur(`b_1`)`
$b_2 \leftarrow \alpha_2(\alpha_1($`RechercherSecteur(`$V$, s_2`)`$))$
$V_2 \leftarrow$ `Vecteur(`b_2`)`
2 **répéter**
suivant la position des vecteurs V, V_1 et V_2 **faire**
3 **cas où** V et V_2 sont colinéaires et dans le même sens **alors**
si $\alpha_1(b) \neq \alpha_2(b_2)$ **alors** `Coudre(`b, $\alpha_2(b_2)$, α'_1`)`
$b \leftarrow b_2$; $V \leftarrow V_2$
$b_2 \leftarrow \alpha_2(\alpha_1(b_2))$; $V_2 \leftarrow$ `Vecteur(`b_2`)`
4 **cas où** V_1 et V_2 sont colinéaires et dans le même sens **alors**
si $\alpha_1(b) \neq \alpha_2(b_2)$ **alors** `Coudre(`b, $\alpha_2(b_2)$, α'_1`)`
si $\alpha_1(b_2) \neq \alpha_2(b_1)$ **alors** `Coudre(`b_2, $\alpha_2(b_1)$, α'_1`)`
$b \leftarrow b_1$; $V \leftarrow V_1$
$b_1 \leftarrow \alpha_2(\alpha_1(b_1))$; $V_1 \leftarrow$ `Vecteur(`b_1`)`
$b_2 \leftarrow \alpha_2(\alpha_1(b_2))$; $V_2 \leftarrow$ `Vecteur(`b_2`)`
5 **cas où** `VecteurDansSecteur(`V_2, V, V_1`)` **alors**
si $\alpha_1(b) \neq \alpha_2(b_2)$ **alors** `Coudre(`b, $\alpha_2(b_2)$, α'_1`)`
$b \leftarrow b_2$; $V \leftarrow V_2$
$b_2 \leftarrow \alpha_2(\alpha_1(b_2))$; $V_2 \leftarrow$ `Vecteur(`b_2`)`
6 **autres cas alors**
si $\alpha_1(b) \neq \alpha_2(b_1)$ **alors** `Coudre(`b, $\alpha_2(b_1)$, α'_1`)`
$b \leftarrow b_1$; $V \leftarrow V_1$
$b_1 \leftarrow \alpha_2(\alpha_1(b_1))$; $V_1 \leftarrow$ `Vecteur(`b_1`)`
jusqu'à $b = s_1$
fin



(a) angle aigu (b) angle obtus

FIG. 3.13 – Test d'appartenance d'un vecteur à un secteur angulaire

ALGO. 3.9 – Fonction `RechercherSecteur`

Entrées : ▪ un vecteur V ;
▪ un brin b d'un sommet s.
Sorties : Un brin du secteur incident à s contenant V.

début
  $b' \leftarrow b$
  $V_1 \leftarrow$ `Vecteur(`b'`)`
  $V_2 \leftarrow$ `Vecteur(`$\alpha_1(b')$`)`
  tant que `VecteurDansSecteur(`V, V_1, V_2`)` **faire**
    $b' \leftarrow \alpha_2(\alpha_1(b'))$
    $V_1 \leftarrow V_2$
    $V_2 \leftarrow$ `Vecteur(`$\alpha_1(b')$`)`
  retourner b'
fin

ALGO. 3.10 – Fonction `VecteurDansSecteur`

Entrées : ▪ un vecteur V à tester ;
▪ deux vecteurs V_1, V_2 définissant un secteur angulaire.
Sorties : `vrai` si V est dans le secteur formé par V_1 et V_2, `faux` sinon.

début
  si $(V_1 \wedge V_2).z < 0$ **alors**
    retourner $(V_1 \wedge V).z \geq 0$ **ou** $(V_2 \wedge V).z \leq 0$
  sinon
    retourner $(V_1 \wedge V).z \geq 0$ **et** $(V_2 \wedge V).z \leq 0$
fin

sommet du premier maillage et nous propager le long de ses arêtes incidentes. Cependant, nous devons connaître la face du deuxième maillage qui contient ce sommet en le localisant.

Une fois que nous avons localisé notre sommet, nous devons traiter un premier cas particulier. En effet, comme nous l'expliquons dans la section 3.1, nous supposons durant le parcours que les intersections ne peuvent avoir lieu qu'avec l'arête parcourue privée de sa première extrémité. Hors, comme ici le parcours n'est pas encore commencé et que nous n'avons toujours pas traité d'intersection, il se peut que le sommet ne se trouve pas dans une face mais sur une arête ou bien un sommet. Nous devons donc analyser la position exacte de ce sommet par rapport au bord de la face trouvée et générer une première intersection si nécessaire.

ALGO. 3.11 – Fonction `Initialiser`

Entrées : Deux brins b_1 et b_2 appartenant à deux maillages distincts.
Sorties : La file $\mathcal{F}$ contenant un couple de brins qui correspond à une première intersection entre les deux maillages.

```
début
    P1 ← Point(b1)
1   b2 ← LocaliserPointDansMaillage(P1, b2)
    P2 ← Point(b2)
2   si P1 = P2 alors
        Marquer(<α1, α2>(b1), Ⓢ)
        FusionnerSommets(b1, b2)
    sinon
3       (pos, t) ← LocaliserPointSurArête(P1, b2)
        si pos = SurArête alors
            Marquer(<α1, α2>(b1), Ⓢ)
            b2 ← EclaterArête(b2, P1)
            FusionnerSommets(b1, b2)
    F ← ∅
4   Enfiler(F, (b1, b2))
fin
```

L'algorithme 3.11 montre les différentes étapes à effectuer afin d'initialiser le parcours. Ces étapes sont les suivantes :

1. Nous commençons par localiser un sommet s du premier maillage dans les faces du second. Pour cela, nous faisons appel à la fonction `LocaliserPointDansMaillage` qui se charge de nous indiquer sur quelle cellule du second maillage se situe ce point (sommet, arête ou face).
2. Ensuite, nous analysons la position de s par rapport à la cellule trouvée. Si s possède la même position que le sommet incident au brin renvoyé par le traitement précédent, il est alors confondu avec ce sommet. Nous devons alors traiter cette première intersection en fusionnant les deux sommets.
3. Par contre, si s se trouve sur l'arête incidente au brin renvoyé par la localisation, nous devons traiter cette intersection en éclatant l'arête et en fusionnant le nouveau sommet

avec s. Pour tous ces traitements nous utilisons les fonctions et procédures déjà existantes détaillées dans les sections précédentes.

4. Pour finir, qu'il y ait eu une intersection ou non, nous ajoutons à la file $\mathcal{F}$ les brins formant notre couple $(sommet, face)$.

3.2.2.1 Localisation d'un point dans un maillage

Pour localiser le sommet de départ du premier maillage par rapport aux faces du second, nous utilisons la technique décrite en section 3.1.3.4 (page 24).

ALGO. 3.12 – Fonction `LocaliserPointDansMaillage`

Entrées : ▪ un point P à localiser ;
▪ un brin b d'un maillage M dans lequel la reherche doit s'effectuer.
Sorties : Un brin du maillage M incident à la face contenant P.

```
début
    b_c ← b -- brin de la plus proche cellule intersectée
    b_f ← nil -- brin de la face courante
    d ← 0 -- dimension de la plus proche cellule intersectée
    I ← nil -- plus proche point d'intersection
1   répéter
        suivant la valeur de d faire
2           cas où d = 0 alors
                O ← Point(b_c)
                b_f ← RechercherSecteur(Vecteur(O, P), b_c)
3           cas où d = 1 alors
                O ← I
                b_f ← α2(α0(b_c))
4       (b_c, d, I, pos) ← RechercherIntersection(O, P, b_f)
    jusqu'à b_c = nil
5   retourner b_f
fin
```

Nous obtenons ainsi l'algorithme 3.12 qui permet de localiser un point dans un maillage en ne testant qu'un nombre limité de faces. Cet algorithme se découpe selon les étapes suivantes :

1. Nous effectuons une boucle qui s'arrête lorsque nous n'avons plus d'intersection. À chaque tour de boucle nous regardons la dimension de la cellule intersectée afin de savoir dans quelle face nous devons ensuite nous diriger.
2. Si la cellule intersectée est un sommet, nous recherchons la face incidente à ce dernier qui contient le vecteur directeur que nous suivons. Pour cela, nous faisons appel à la fonction `RechercherSecteur`.
3. Si la cellule intersectée est une arête, nous passons simplement dans la face adjacente.

4. Lorsque nous sommes dans une nouvelle face, il nous suffit de chercher la plus proche intersection à l'aide de la fonction `RechercherIntersection`.

5. Pour finir, nous renvoyons un brin de la dernière face dans laquelle nous étions. Notons ici que ce brin peut aussi bien définir un sommet, une arête ou une face.

3.2.3 Post-traitements

Lorsque le parcours par propagation est terminé et que toutes les intersections ont été traitées, le maillage résultant n'est pas encore tout à fait exploitable. En effet, comme toutes les fusions de sommets ont été réalisées à l'aide de l'involution α_1', les deux maillages ne sont pas encore réellement cousus. De plus le co-raffinement des deux maillages a pu provoquer la création d'arêtes doubles qu'il est nécessaire d'éliminer. Dans cette section, nous détaillons donc deux algorithmes permettant d'effectuer ces traitements afin d'obtenir au final un résultat correct.

3.2.3.1 Assemblage final et nettoyage

Pour n'obtenir plus qu'un seul maillage à la fin de notre algorithme, nous devons coudre par α_1 tous les brins des maillages qui ont été cousus par α_1'. La procédure `Nettoyer` décrite dans l'algorithme 3.13 se charge donc de parcourir tous les brins d'un des deux maillages pour les coudre avec ceux de l'autre maillage. Lorsque ce premier parcours est effectué, les deux maillages ne forment plus qu'un seul maillage et il est alors possible de le parcourir afin de démarquer tous ses brins des marques utilisées lors du parcours par propagation.

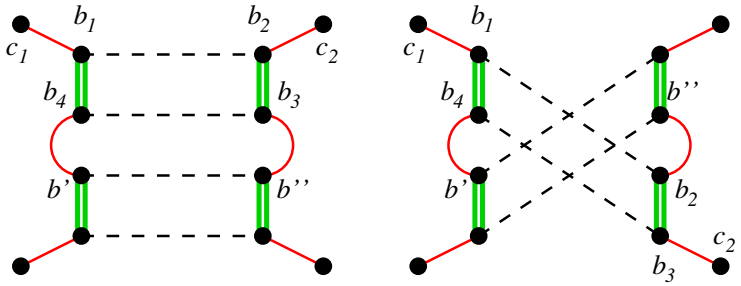
ALGO. 3.13 – Procédure `Nettoyer`

Entrées : Un brin b d'un maillage.
Résultat : Les brins du maillage sont démarqués des marques Ⓢ et Ⓐ et les coutures par α_1' sont appliquées sous la forme de coutures par α_1.

```
début
    pour chaque brin b' du maillage incident à b faire
        si α'1(b') ≠ b' alors
            Découdre(b', α1)
            Coudre(b', α'1(b'), α1)
    pour chaque brin b' du maillage incident à b faire
        Démarquer(b', Ⓢ) ; Démarquer(b', Ⓐ)
fin
```

3.2.3.2 Suppression des arêtes doubles

Lors de la fusion des sommets, notre algorithme a pu générer deux sortes d'arêtes doubles comme le montre la figure 3.14. Nous devons alors détecter s'il existe des arêtes doubles dans



(a) arêtes parallèles (b) arêtes croisées

FIG. 3.14 – Les deux types d'arêtes doubles pouvant être générées par l'algorithme.

le maillage et reconnaître leur type pour pouvoir les supprimer. La détection et la suppression d'une arête double sont des opérations purement topologiques qui ne peuvent donc poser aucun problème. La fonction `SupprimerArêtesDoubles` décrite dans l'algorithme 3.14 permet d'effectuer ces traitements et se découpe de la manière suivante :

1. Pour chaque brin du maillage, nous testons s'il appartient à une arête double dont nous connaissons le type. Si c'est le cas, nous stockons dans des variables intermédiaires b_i les brins appartenant à l'arête en trop.
2. Selon le type de l'arête, nous conservons dans les variables b', b'', c_1 et c_2 les brins que nous devons coudre entre eux pour obtenir une topologie correcte une fois cette arête supprimée.
3. Nous décousons ensuite tous les brins de cette arête afin de la détacher du maillage et nous détruisons tous ses brins.
4. Pour finir, nous cousons les brins contenus dans les variables intermédiaires.

3.3 Résultats

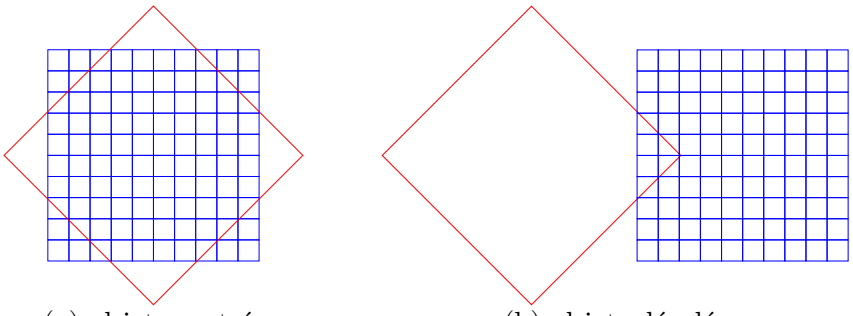
Nous présentons dans cette partie les résultats que nous obtenons sur plusieurs exemples. Pour la majorité d'entre eux, nous effectuons le co-raffinement entre un objet qui diffère selon les exemples et un maillage dont nous faisons varier le nombre de mailles. Pour chaque couple d'objets, nous testons le cas où les deux objets sont centrés l'un par rapport à l'autre afin d'obtenir le plus grand nombre d'intersections et le cas où ils sont décalés afin de n'obtenir qu'une petite partie des objets se chevauchant. Nous pouvons voir les premiers objets testés sur la figure 3.15.

Pour chacun des tests, nous obtenons un ensemble de résultats correspondant au nombre d'intersections réalisées ainsi qu'aux temps de calcul pour chaque étape de l'algorithme. Les tableaux 3.1 et 3.2 indiquent les résultats obtenus pour les deux premiers objets. Ces derniers ont été obtenus à l'aide d'un machine PC équipée d'un processeur Pentium II 400 Mhz et 256 Mo de mémoire vive. Les colonnes de ces tableaux correspondent aux informations suivantes :

ALGO. 3.14 – Fonction `SupprimerArêtesDoubles`

Entrées : Un brin b d'un maillage.
Résultat : Toutes les arêtes doubles du maillage ont été supprimées.

début
  pour chaque brin b' du maillage incident à b **faire**
1 **si** $\alpha_1(\alpha_0(b')) = \alpha_0(\alpha_1(b'))$ **ou** $\alpha_0(\alpha_2(\alpha_1(b'))) = \alpha_1(\alpha_2(\alpha_0(b')))$ **alors**
      $b_1 \leftarrow \alpha_2(\alpha_1(b'))$; $b_2 \leftarrow \alpha_0(b_1)$; $b_3 \leftarrow \alpha_2(b_2)$; $b_4 \leftarrow \alpha_0(b_3)$
2 **si** $\alpha_1(\alpha_0(b')) = \alpha_0(\alpha_1(b'))$ **alors**
        $b'' \leftarrow \alpha_0(b')$; $c_1 \leftarrow \alpha_1(b_1)$; $c_2 \leftarrow \alpha_1(b_2)$
      sinon
        $b'' \leftarrow \alpha_2(\alpha_0(b'))$; $c_1 \leftarrow \alpha_1(b_1)$; $c_2 \leftarrow \alpha_1(b_3)$
3 **pour** $i \in [1..4]$ **faire**
        `Découdre`(b_i, α_1)
        `SupprimerBrin`(b_i)
4 `Coudre`(b', c_1, α_1)
      `Coudre`(b'', c_2, α_1)
fin



(a) objets centrés (b) objets décalés

FIG. 3.15 – Test de performance du co-raffinement entre un carré et un maillage.

$\#S$	$\#I$	t_i	t_o	t_l	t_p	t_a	t_n	t_s	t_t
220	24	35 ms	2 ms	$<$ 1 ms	29 ms	22 ms	32 ms	18 ms	141 ms
840	48	99 ms	14 ms	$<$ 1 ms	43 ms	31 ms	81 ms	32 ms	304 ms
1860	72	204 ms	3 ms	1 ms	57 ms	36 ms	161 ms	57 ms	521 ms
3280	96	371 ms	13 ms	1 ms	70 ms	47 ms	283 ms	108 ms	897 ms
5100	120	577 ms	14 ms	9 ms	87 ms	57 ms	422 ms	150 ms	1 s 321 ms
7320	144	825 ms	6 ms	11 ms	105 ms	68 ms	598 ms	212 ms	1 s 829 ms
9940	168	1 s 121 ms	16 ms	10 ms	119 ms	77 ms	808 ms	285 ms	2 s 438 ms
12960	192	1 s 458 ms	16 ms	3 ms	137 ms	86 ms	1 s 48 ms	368 ms	3 s 119 ms
16380	216	1 s 841 ms	18 ms	3 ms	155 ms	96 ms	1 s 315 ms	465 ms	3 s 896 ms
20200	240	2 s 270 ms	18 ms	4 ms	169 ms	107 ms	1 s 618 ms	570 ms	4 s 759 ms
24420	264	2 s 746 ms	19 ms	13 ms	186 ms	116 ms	1 s 947 ms	685 ms	5 s 715 ms
29040	288	3 s 255 ms	20 ms	5 ms	200 ms	126 ms	2 s 316 ms	812 ms	6 s 737 ms
34060	312	3 s 821 ms	22 ms	14 ms	217 ms	136 ms	2 s 702 ms	981 ms	7 s 896 ms
39480	336	4 s 426 ms	24 ms	6 ms	232 ms	147 ms	3 s 125 ms	1 s 121 ms	9 s 84 ms
45300	352	5 s 87 ms	23 ms	15 ms	240 ms	153 ms	3 s 578 ms	1 s 250 ms	10 s 349 ms
51520	376	5 s 823 ms	26 ms	6 ms	256 ms	165 ms	4 s 67 ms	1 s 419 ms	11 s 766 ms
58140	400	6 s 519 ms	27 ms	7 ms	277 ms	171 ms	4 s 581 ms	1 s 600 ms	13 s 185 ms
65160	424	7 s 341 ms	28 ms	15 ms	289 ms	184 ms	5 s 133 ms	1 s 791 ms	14 s 785 ms
72580	448	8 s 152 ms	29 ms	8 ms	311 ms	191 ms	5 s 715 ms	1 s 994 ms	16 s 403 ms
80400	472	9 s 2 ms	30 ms	17 ms	323 ms	202 ms	6 s 326 ms	2 s 205 ms	18 s 110 ms

Tab. 3.1 – Temps de calcul obtenus pour le co-raffinement entre un carré et un maillage centrés l'un par rapport à l'autre.

$\#S$	$\#I$	t_i	t_o	t_l	t_p	t_a	t_n	t_s	t_t
220	11	37 ms	2 ms	$<$ 1 ms	22 ms	4 ms	33 ms	18 ms	119 ms
840	19	101 ms	2 ms	$<$ 1 ms	27 ms	19 ms	76 ms	31 ms	260 ms
1860	27	205 ms	3 ms	9 ms	26 ms	19 ms	155 ms	53 ms	474 ms
3280	35	369 ms	12 ms	1 ms	33 ms	22 ms	267 ms	96 ms	803 ms
5100	43	579 ms	6 ms	11 ms	38 ms	25 ms	412 ms	146 ms	1 s 220 ms
7320	51	829 ms	15 ms	2 ms	44 ms	28 ms	587 ms	207 ms	1 s 716 ms
9940	59	1 s 122 ms	8 ms	11 ms	50 ms	31 ms	793 ms	277 ms	2 s 294 ms
12960	67	1 s 461 ms	16 ms	12 ms	55 ms	35 ms	1 s 32 ms	360 ms	2 s 974 ms
16380	75	1 s 845 ms	19 ms	11 ms	63 ms	38 ms	1 s 300 ms	453 ms	3 s 731 ms
20200	83	2 s 274 ms	20 ms	13 ms	68 ms	41 ms	1 s 600 ms	559 ms	4 s 578 ms
24420	91	2 s 753 ms	21 ms	4 ms	75 ms	46 ms	1 s 959 ms	671 ms	5 s 533 ms
29040	99	3 s 270 ms	22 ms	5 ms	80 ms	49 ms	2 s 291 ms	800 ms	6 s 519 ms
34060	107	3 s 834 ms	23 ms	14 ms	86 ms	51 ms	2 s 692 ms	962 ms	7 s 666 ms
39480	115	4 s 439 ms	25 ms	14 ms	92 ms	54 ms	3 s 136 ms	1 s 82 ms	8 s 846 ms
45300	127	5 s 97 ms	26 ms	6 ms	101 ms	60 ms	3 s 581 ms	1 s 239 ms	10 s 113 ms
51520	135	5 s 797 ms	27 ms	15 ms	105 ms	63 ms	4 s 62 ms	1 s 409 ms	11 s 481 ms
58140	143	6 s 534 ms	29 ms	16 ms	112 ms	67 ms	4 s 584 ms	1 s 588 ms	12 s 933 ms
65160	151	7 s 312 ms	29 ms	16 ms	118 ms	71 ms	5 s 138 ms	1 s 777 ms	14 s 464 ms
72580	159	8 s 160 ms	30 ms	17 ms	123 ms	74 ms	5 s 710 ms	1 s 980 ms	16 s 96 ms
80400	167	9 s 28 ms	31 ms	18 ms	129 ms	78 ms	6 s 330 ms	2 s 189 ms	17 s 805 ms

Tab. 3.2 – Temps de calcul obtenus pour le co-raffinement entre un carré et un maillage décalés l'un par rapport à l'autre.

$\#S$ Nombre de segments contenus dans le deuxième objet (i.e. le maillage).

$\#I$ Nombre d'intersections réalisées durant le traitement.

t_i Durée de l'initialisation des objets. Cette étape permet de rendre valide les objets en fermant les 2-bords (bords libres par α_2) et en ajoutant dans les brins quelques informations qui peuvent varier selon l'implémentation.

t_o Durée de l'orientation des objets. Cette phase est facultative[5] et sert uniquement à déterminer l'orientation de la G-Carte.

t_l Durée de la localisation d'un sommet du premier objet dans les faces du second (cf. section 3.2.2.1).

t_p Durée du parcours par propagation dans les faces du second objet (cf. section 3.2.1).

t_a Durée de l'assemblage des objets. Cette étape consiste uniquement à appliquer les modifications enregistrées dans les involutions α'_1 (cf. section 3.2.3.1).

t_n Durée du nettoyage des objets. Cette phase permet d'enlever des brins les marques utilisées pendant le parcours ainsi que les informations rajoutées durant l'initialisation (cf. section 3.2.3.1).

t_s Durée de la suppression des arêtes doubles (cf. section 3.2.3.2).

t_t Durée totale du calcul. Ceci correspond à la somme des temps précédents.

Les courbes des figures 3.16 et 3.17 correspondent aux données des tableaux précédents et nous permettent de faire quelques commentaires sur le comportement de l'algorithme. Nous pouvons ainsi voir que pour les objets de la figure 3.15, la majeure partie du temps est utilisée pour l'initialisation, le nettoyage et la suppression des arêtes doubles. En ce qui concerne la durée du parcours par propagation, elle est négligeable par rapport au reste. Nous pouvons de plus remarquer que quelle que soit la position des objets, la forme des courbes confirme les suppositions faites en section 3.1.1 concernant la complexité de l'algorithme. L'évolution de la durée du parcours par rapport au nombre de segments dans le maillage correspond alors grossièrement à une fonction racine carrée. De plus, nous pouvons observer sur les graphiques en bas à droite des figures que les intersections sont calculées en temps constant.

[5]Elle est facultative dans le sens où l'orientation de la G-Carte peut être fournie directement par l'utilisateur et n'a alors pas besoin d'être calculée.

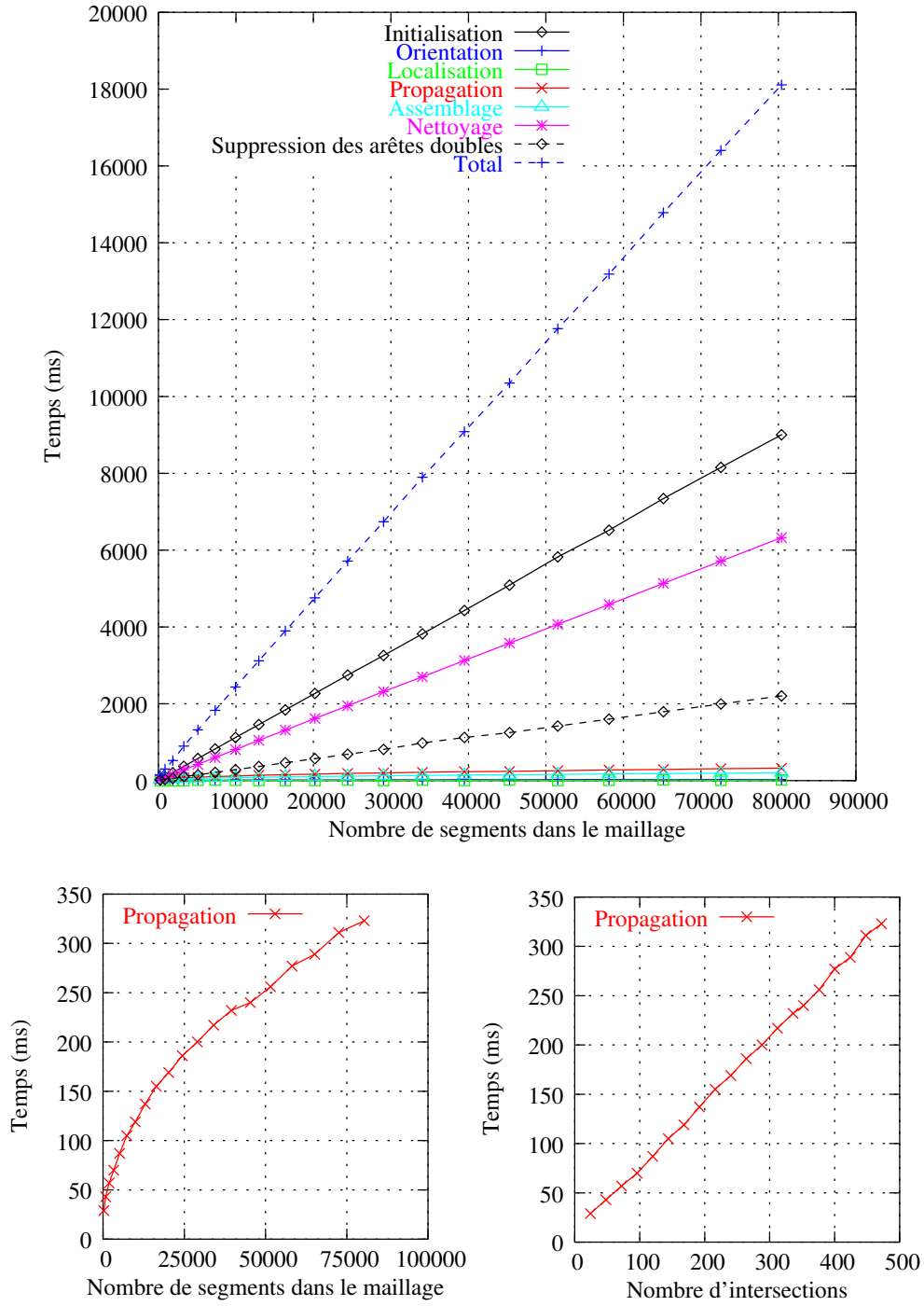


FIG. 3.16 – Évolution des temps de calcul pour le co-raffinement entre un carré et un maillage centrés l'un par rapport à l'autre.

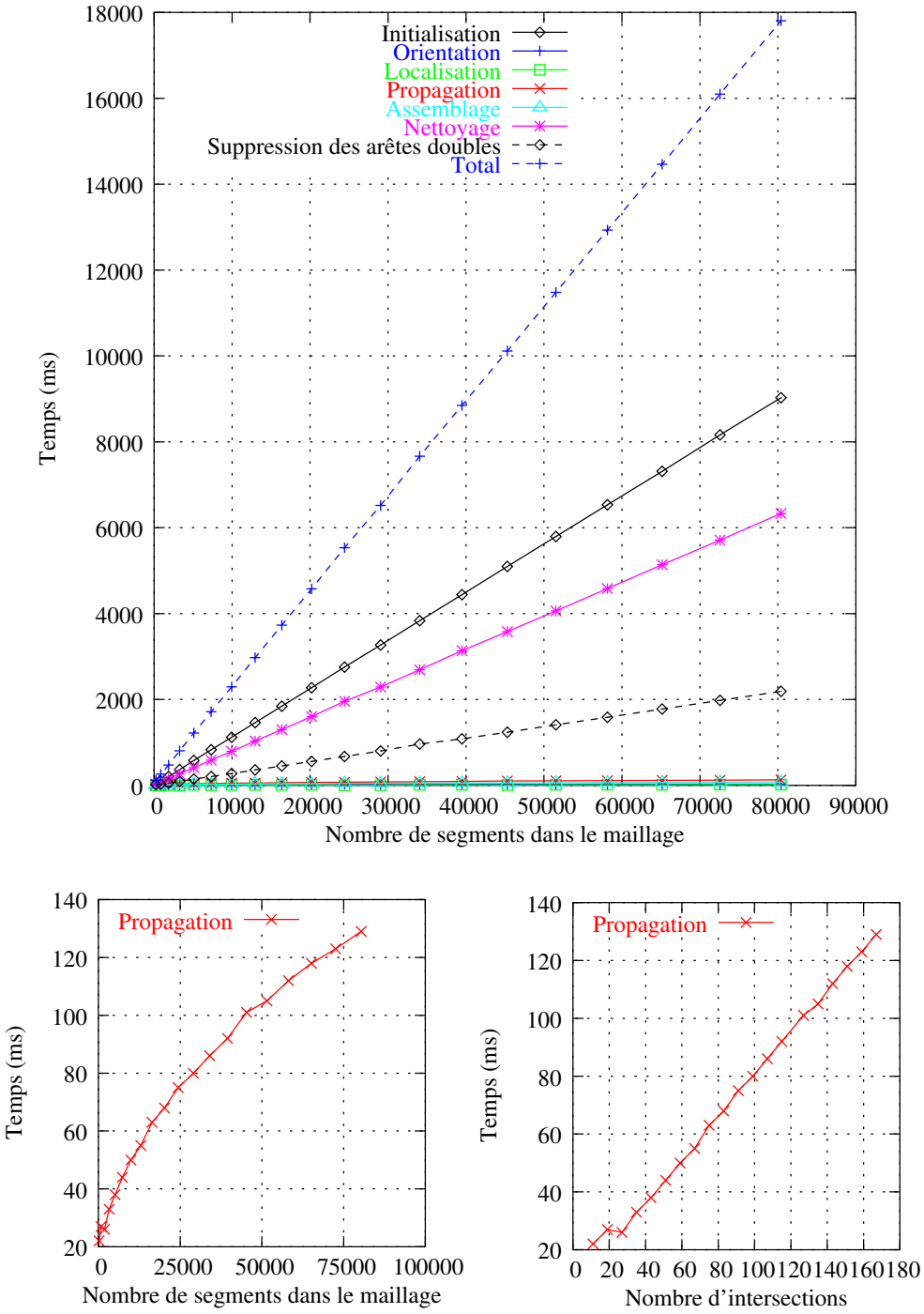
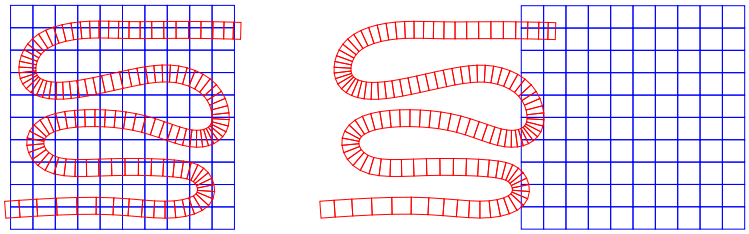


FIG. 3.17 – Évolution des temps de calcul pour le co-raffinement entre un carré et un maillage décalés l'un par rapport à l'autre.



(a) objets centrés (b) objets décalés

FIG. 3.18 – Test de performance du co-raffinement entre un chemin et un maillage.

Pour confirmer les hypothèses précédentes, nous effectuons un nouveau test sur des objets un peu plus complexes qui sont un chemin maillé et un maillage dont la taille varie comme précédemment (cf. FIG. 3.18). Nous obtenons alors les tableaux 3.3 et 3.4 ainsi que les courbes correspondantes sur les figures 3.19 et 3.20. Ces résultats ont été obtenus sur une machine PC équipée d'un processeur Athlon 64 3000+ et de 512 Mo de mémoire vive.

Nous pouvons donc faire les mêmes remarques que précédemment mis à part que le temps de propagation est beaucoup plus important car il correspond à un plus grand nombre d'intersections. Cependant, notons que la durée du parcours par propagation est plus longue lorsque les objets sont décalés que lorsqu'ils sont centrés. Ceci est dû au fait que la majeure partie des tests d'intersection est effectuée avec la face extérieure du maillage qui possède un nombre d'arêtes bien plus grand que les faces internes.

$\#S$	$\#I$	t_i	t_o	t_l	t_p	t_a	t_n	t_s	t_t
220	241	32 ms	$<$ 1 ms	$<$ 1 ms	59 ms	37 ms	48 ms	21 ms	201 ms
840	454	48 ms	$<$ 1 ms	$<$ 1 ms	61 ms	55 ms	56 ms	29 ms	252 ms
1860	672	76 ms	$<$ 1 ms	$<$ 1 ms	97 ms	72 ms	83 ms	38 ms	370 ms
3280	902	118 ms	$<$ 1 ms	$<$ 1 ms	120 ms	93 ms	119 ms	51 ms	504 ms
5100	1126	170 ms	$<$ 1 ms	16 ms	155 ms	110 ms	165 ms	66 ms	685 ms
7320	1338	237 ms	1 ms	1 ms	176 ms	129 ms	215 ms	85 ms	847 ms
9940	1578	310 ms	17 ms	1 ms	201 ms	148 ms	280 ms	103 ms	1 s 62 ms
12960	1801	396 ms	17 ms	17 ms	222 ms	160 ms	346 ms	126 ms	1 s 287 ms
16380	2021	492 ms	17 ms	17 ms	245 ms	184 ms	423 ms	153 ms	1 s 535 ms
20200	2238	605 ms	30 ms	18 ms	255 ms	219 ms	509 ms	179 ms	1 s 817 ms
24420	2467	720 ms	17 ms	18 ms	282 ms	222 ms	596 ms	209 ms	2 s 66 ms
29040	2690	857 ms	18 ms	18 ms	305 ms	243 ms	698 ms	243 ms	2 s 385 ms
34060	2913	1 s 2 ms	18 ms	18 ms	330 ms	264 ms	804 ms	279 ms	2 s 717 ms
39480	3129	1 s 156 ms	18 ms	18 ms	356 ms	276 ms	920 ms	317 ms	3 s 64 ms
45300	3372	1 s 322 ms	18 ms	19 ms	375 ms	304 ms	1 s 50 ms	373 ms	3 s 463 ms
51520	3594	1 s 495 ms	19 ms	19 ms	397 ms	327 ms	1 s 179 ms	403 ms	3 s 842 ms
58140	3801	1 s 693 ms	18 ms	19 ms	422 ms	360 ms	1 s 334 ms	453 ms	4 s 302 ms
65160	4028	1 s 894 ms	19 ms	19 ms	455 ms	343 ms	1 s 472 ms	502 ms	4 s 707 ms
72580	4254	2 s 107 ms	32 ms	20 ms	474 ms	381 ms	1 s 619 ms	547 ms	5 s 181 ms
80400	4477	2 s 328 ms	19 ms	20 ms	489 ms	428 ms	1 s 794 ms	601 ms	5 s 680 ms

Tab. 3.3 – Temps de calcul obtenus pour le co-raffinement entre un chemin et un maillage centrés l'un par rapport à l'autre.

$\#S$	$\#I$	t_i	t_o	t_l	t_p	t_a	t_n	t_s	t_t
220	38	34 ms	$<$ 1 ms	$<$ 1 ms	104 ms	20 ms	28 ms	19 ms	208 ms
840	67	47 ms	$<$ 1 ms	$<$ 1 ms	169 ms	23 ms	40 ms	23 ms	305 ms
1860	95	76 ms	$<$ 1 ms	$<$ 1 ms	234 ms	24 ms	63 ms	31 ms	431 ms
3280	127	118 ms	$<$ 1 ms	$<$ 1 ms	299 ms	28 ms	105 ms	41 ms	593 ms
5100	155	170 ms	$<$ 1 ms	$<$ 1 ms	376 ms	31 ms	128 ms	53 ms	762 ms
7320	183	235 ms	1 ms	1 ms	443 ms	33 ms	176 ms	68 ms	959 ms
9940	209	311 ms	1 ms	1 ms	507 ms	35 ms	222 ms	83 ms	1 s 162 ms
12960	239	390 ms	17 ms	1 ms	572 ms	38 ms	283 ms	103 ms	1 s 406 ms
16380	269	503 ms	17 ms	29 ms	626 ms	40 ms	355 ms	126 ms	1 s 699 ms
20200	296	601 ms	17 ms	17 ms	704 ms	43 ms	431 ms	149 ms	1 s 966 ms
24420	323	719 ms	17 ms	17 ms	756 ms	46 ms	518 ms	177 ms	2 s 252 ms
29040	357	857 ms	17 ms	18 ms	826 ms	50 ms	604 ms	207 ms	2 s 582 ms
34060	387	1 s	18 ms	18 ms	890 ms	52 ms	704 ms	243 ms	2 s 928 ms
39480	415	1 s 157 ms	18 ms	18 ms	955 ms	54 ms	814 ms	274 ms	3 s 293 ms
45300	443	1 s 320 ms	18 ms	18 ms	1 s 24 ms	58 ms	942 ms	311 ms	3 s 694 ms
51520	468	1 s 486 ms	18 ms	18 ms	1 s 88 ms	59 ms	1 s 57 ms	351 ms	4 s 80 ms
58140	495	1 s 696 ms	19 ms	18 ms	1 s 154 ms	61 ms	1 s 190 ms	393 ms	4 s 535 ms
65160	525	1 s 891 ms	19 ms	19 ms	1 s 223 ms	65 ms	1 s 335 ms	437 ms	4 s 991 ms
72580	552	2 s 104 ms	19 ms	19 ms	1 s 289 ms	67 ms	1 s 474 ms	483 ms	5 s 458 ms
80400	581	2 s 341 ms	19 ms	19 ms	1 s 351 ms	70 ms	1 s 625 ms	531 ms	5 s 959 ms

Tab. 3.4 – Temps de calcul obtenus pour le co-raffinement entre un chemin et un maillage décalés l'un par rapport à l'autre.

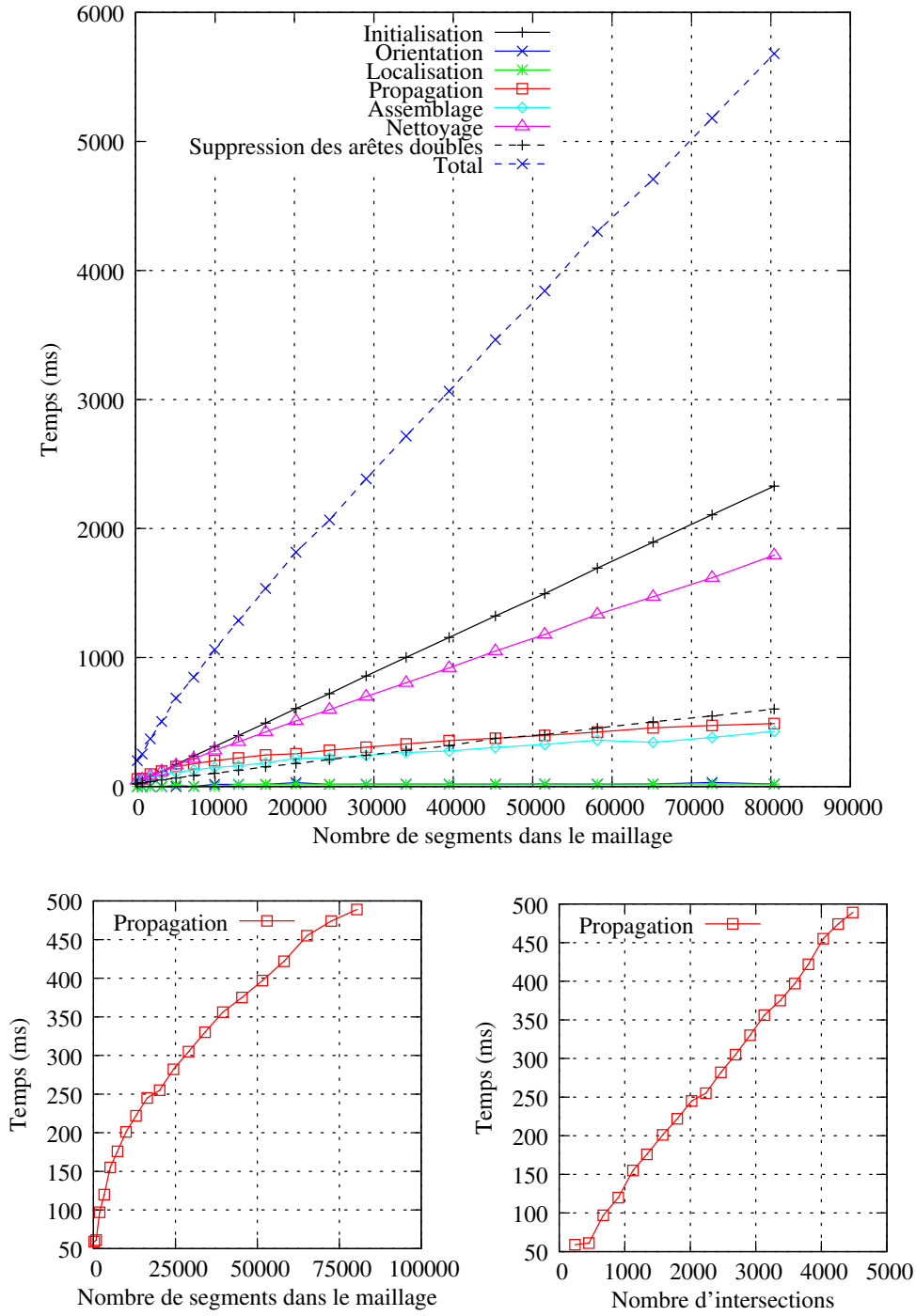


FIG. 3.19 – Évolution des temps de calcul pour le co-raffinement entre un chemin et un maillage centrés l'un par rapport à l'autre.

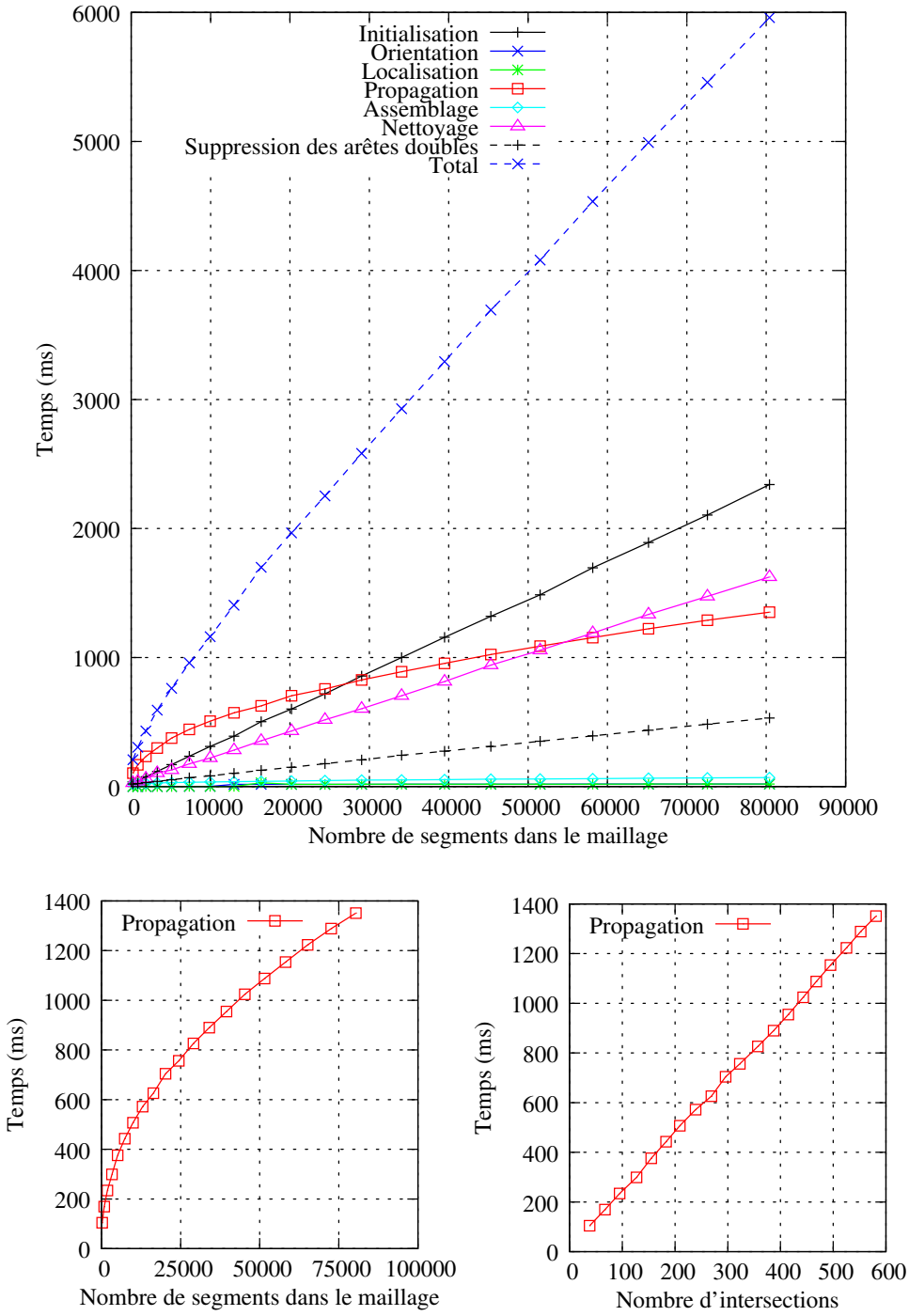
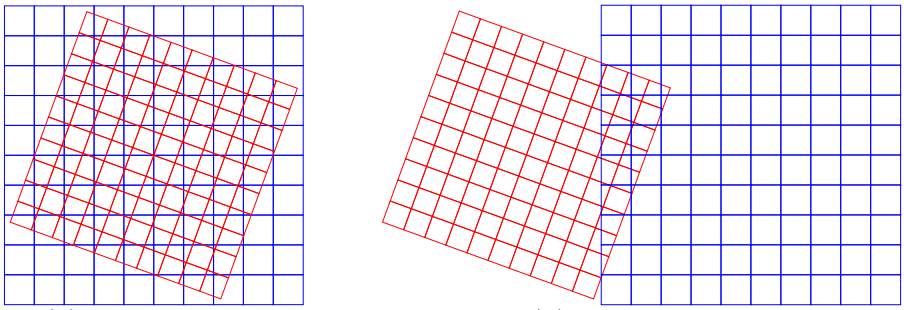


FIG. 3.20 – Évolution des temps de calcul pour le co-raffinement entre un chemin et un maillage décalés l'un par rapport à l'autre.



(a) objets centrés (b) objets décalés

FIG. 3.21 – Test de performance du co-raffinement entre deux maillages identiques.

Pour finir, nous testons deux maillages identiques pour lesquels nous faisons varier en même temps le nombre de leurs segments (cf. FIG. 3.21). Nous obtenons ainsi les tableaux 3.5 et 3.6 ainsi que les courbes des figures 3.22 et 3.23. Ces derniers résultats ont été obtenus sur la même machine que les précédents.

La première chose que nous pouvons observer est que les courbes ont changé de tendance et sont devenues linéaires pour le cas où les maillages ne sont pas déclalés (cf. FIG. 3.22). Ce résultat confirme alors les hypothèses que nous avions énoncées en section 3.1.1. Cependant, nous pouvons observer que lorsque nous décalons les objets (cf. FIG. 3.23), les courbes ne sont plus linéaires et prennent une forme parabolique. Ce phénomène est dû au fait que la quasi totalité des arêtes du premier maillage est testée avec la face extérieure du second maillage dont le nombre de côtés croît proportionnellement à la résolution du maillage.

Au vu de tous les résultats, nous pouvons conclure que l'algorithme réalisé est très efficace pour rechercher des intersections au sein d'un maillage et que la durée de cette recherche est directement proportionnelle au nombre d'intersections trouvées. Cependant, cette méthode donne de moins bons résultats lorsque les objets comportent des faces possédant un très grand nombre d'arêtes. Elle n'est donc pas adaptée pour traiter des objets non maillés car la complexité de l'algorithme tend alors vers $\mathcal{O}(n^2)$.

$\#S$	$\#I$	t_i	t_o	t_l	t_p	t_a	t_n	t_s	t_t
220	209	27 ms	$<$ 1 ms	$<$ 1 ms	47 ms	33 ms	31 ms	20 ms	160 ms
840	805	57 ms	$<$ 1 ms	$<$ 1 ms	124 ms	86 ms	78 ms	37 ms	385 ms
1860	1789	116 ms	1 ms	$<$ 1 ms	295 ms	189 ms	161 ms	64 ms	827 ms
3280	3149	200 ms	1 ms	$<$ 1 ms	511 ms	286 ms	273 ms	104 ms	1 s 378 ms
5100	4905	303 ms	16 ms	$<$ 1 ms	788 ms	447 ms	420 ms	150 ms	2 s 127 ms
7320	7033	430 ms	17 ms	$<$ 1 ms	1 s 133 ms	639 ms	597 ms	214 ms	3 s 32 ms
9940	9545	579 ms	17 ms	$<$ 1 ms	1 s 530 ms	833 ms	808 ms	279 ms	4 s 49 ms
12960	12449	751 ms	17 ms	$<$ 1 ms	2 s 1 ms	1 s 159 ms	1 s 43 ms	363 ms	5 s 336 ms
16380	15729	945 ms	18 ms	$<$ 1 ms	2 s 526 ms	1 s 414 ms	1 s 314 ms	452 ms	6 s 672 ms
20200	19401	1 s 162 ms	18 ms	$<$ 1 ms	3 s 129 ms	1 s 734 ms	1 s 633 ms	556 ms	8 s 235 ms
24420	23449	1 s 405 ms	19 ms	$<$ 1 ms	3 s 787 ms	2 s 91 ms	1 s 959 ms	679 ms	9 s 943 ms
29040	27889	1 s 670 ms	19 ms	$<$ 1 ms	4 s 519 ms	2 s 449 ms	2 s 335 ms	792 ms	11 s 787 ms
34060	32701	1 s 988 ms	19 ms	$<$ 1 ms	5 s 243 ms	2 s 977 ms	2 s 721 ms	917 ms	13 s 868 ms
39480	37909	2 s 291 ms	20 ms	$<$ 1 ms	6 s 79 ms	3 s 458 ms	3 s 150 ms	1 s 78 ms	16 s 79 ms
45300	43497	2 s 635 ms	20 ms	$<$ 1 ms	6 s 986 ms	3 s 979 ms	3 s 599 ms	1 s 219 ms	18 s 442 ms
51520	49453	2 s 975 ms	20 ms	$<$ 1 ms	7 s 931 ms	4 s 375 ms	4 s 99 ms	1 s 378 ms	20 s 781 ms
58140	55805	3 s 350 ms	21 ms	1 ms	8 s 980 ms	5 s 68 ms	4 s 624 ms	1 s 564 ms	23 s 610 ms
65160	62537	3 s 745 ms	21 ms	1 ms	10 s 63 ms	5 s 716 ms	5 s 173 ms	1 s 743 ms	26 s 465 ms
72580	69641	4 s 193 ms	22 ms	1 ms	11 s 215 ms	6 s 182 ms	5 s 777 ms	1 s 944 ms	29 s 336 ms
80400	77149	4 s 656 ms	22 ms	1 ms	12 s 420 ms	6 s 865 ms	6 s 383 ms	2 s 172 ms	32 s 521 ms

TAB. 3.5 – Temps de calcul obtenus pour le co-raffinement entre deux maillages identiques centrés l'un par rapport à l'autre.

$\#S$	$\#I$	t_i	t_o	t_l	t_p	t_a	t_n	t_s	t_t
220	42	25 ms	$<$ 1 ms	$<$ 1 ms	76 ms	18 ms	22 ms	16 ms	160 ms
840	138	54 ms	$<$ 1 ms	$<$ 1 ms	377 ms	27 ms	49 ms	24 ms	535 ms
1860	292	114 ms	1 ms	1 ms	1 s 137 ms	45 ms	109 ms	43 ms	1 s 452 ms
3280	500	209 ms	1 ms	28 ms	2 s 592 ms	68 ms	161 ms	62 ms	3 s 125 ms
5100	772	304 ms	14 ms	28 ms	4 s 964 ms	98 ms	246 ms	104 ms	5 s 761 ms
7320	1090	430 ms	14 ms	28 ms	8 s 482 ms	136 ms	344 ms	124 ms	9 s 561 ms
9940	1472	586 ms	15 ms	15 ms	13 s 340 ms	178 ms	462 ms	162 ms	14 s 760 ms
12960	1906	755 ms	15 ms	28 ms	19 s 807 ms	226 ms	609 ms	206 ms	21 s 649 ms
16380	2398	953 ms	16 ms	16 ms	28 s 92 ms	296 ms	762 ms	261 ms	30 s 398 ms
20200	2948	1 s 158 ms	16 ms	24 ms	38 s 398 ms	363 ms	941 ms	318 ms	41 s 221 ms
24420	3554	1 s 410 ms	16 ms	91 ms	50 s 789 ms	428 ms	1 s 130 ms	381 ms	54 s 249 ms
29040	4213	1 s 703 ms	19 ms	19 ms	1 m 6 s	497 ms	1 s 357 ms	453 ms	1 m 10 s
34060	4931	1 s 984 ms	19 ms	20 ms	1 m 23 s	582 ms	1 s 575 ms	519 ms	1 m 28 s
39480	5707	2 s 275 ms	20 ms	21 ms	1 m 44 s	668 ms	1 s 833 ms	603 ms	1 m 49 s
45300	6541	2 s 620 ms	20 ms	20 ms	2 m 8 s	750 ms	2 s 75 ms	672 ms	2 m 14 s
51520	7423	2 s 952 ms	7 ms	7 ms	2 m 35 s	874 ms	2 s 381 ms	779 ms	2 m 42 s
58140	8370	3 s 385 ms	21 ms	21 ms	3 m 6 s	988 ms	2 s 701 ms	884 ms	3 m 14 s
65160	9366	3 s 762 ms	21 ms	21 ms	3 m 40 s	1 s 104 ms	3 s 14 ms	987 ms	3 m 49 s
72580	10416	4 s 230 ms	22 ms	23 ms	4 m 19 s	1 s 228 ms	3 s 373 ms	1 s 98 ms	4 m 29 s
80400	11536	4 s 664 ms	22 ms	22 ms	5 m 2 s	1 s 372 ms	3 s 723 ms	1 s 214 ms	5 m 13 s

TAB. 3.6 – Temps de calcul obtenus pour le co-raffinement entre deux maillages identiques décalés l'un par rapport à l'autre.

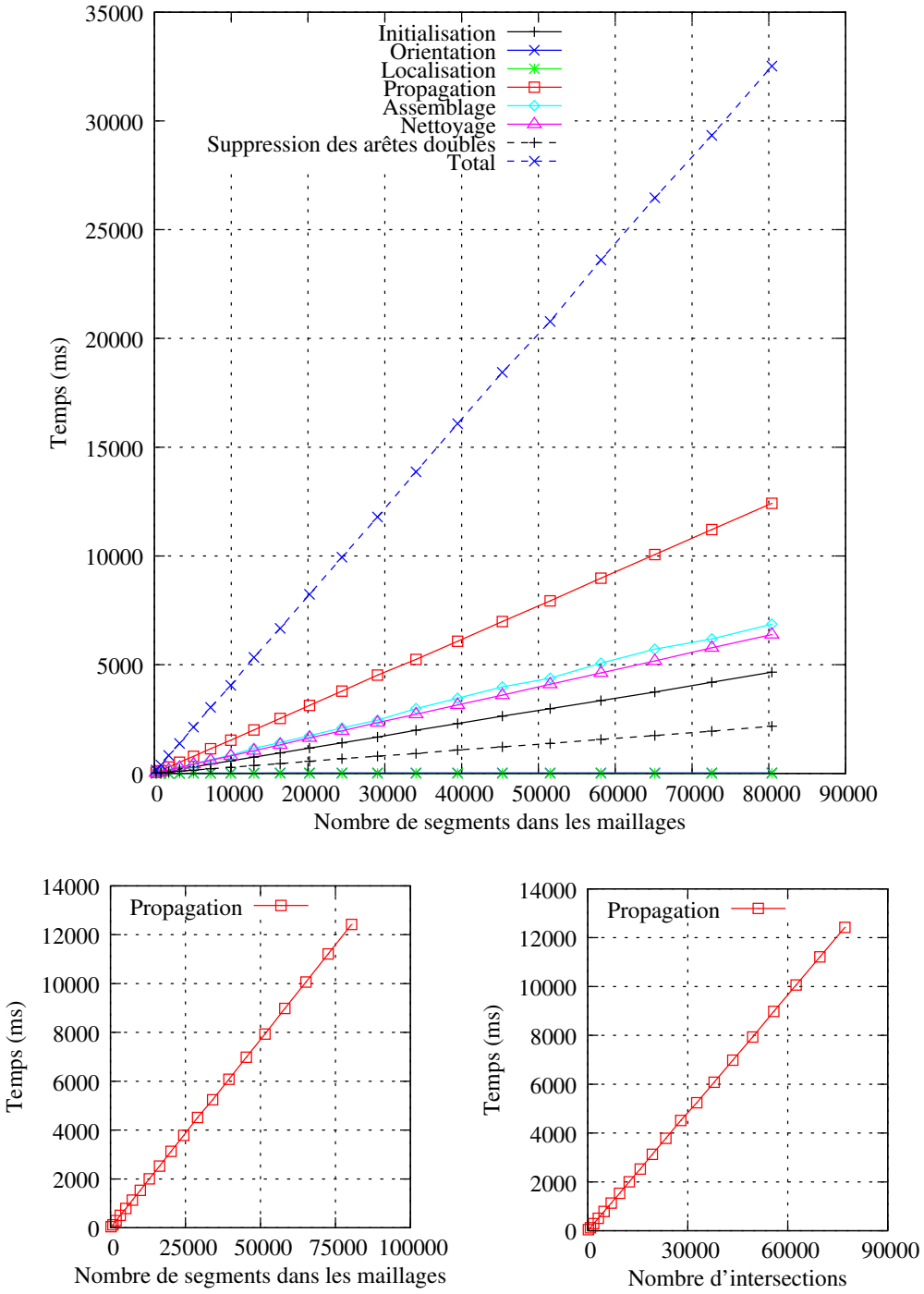


FIG. 3.22 – Évolution des temps de calcul pour le co-raffinement entre deux maillages identiques centrés l'un par rapport à l'autre.

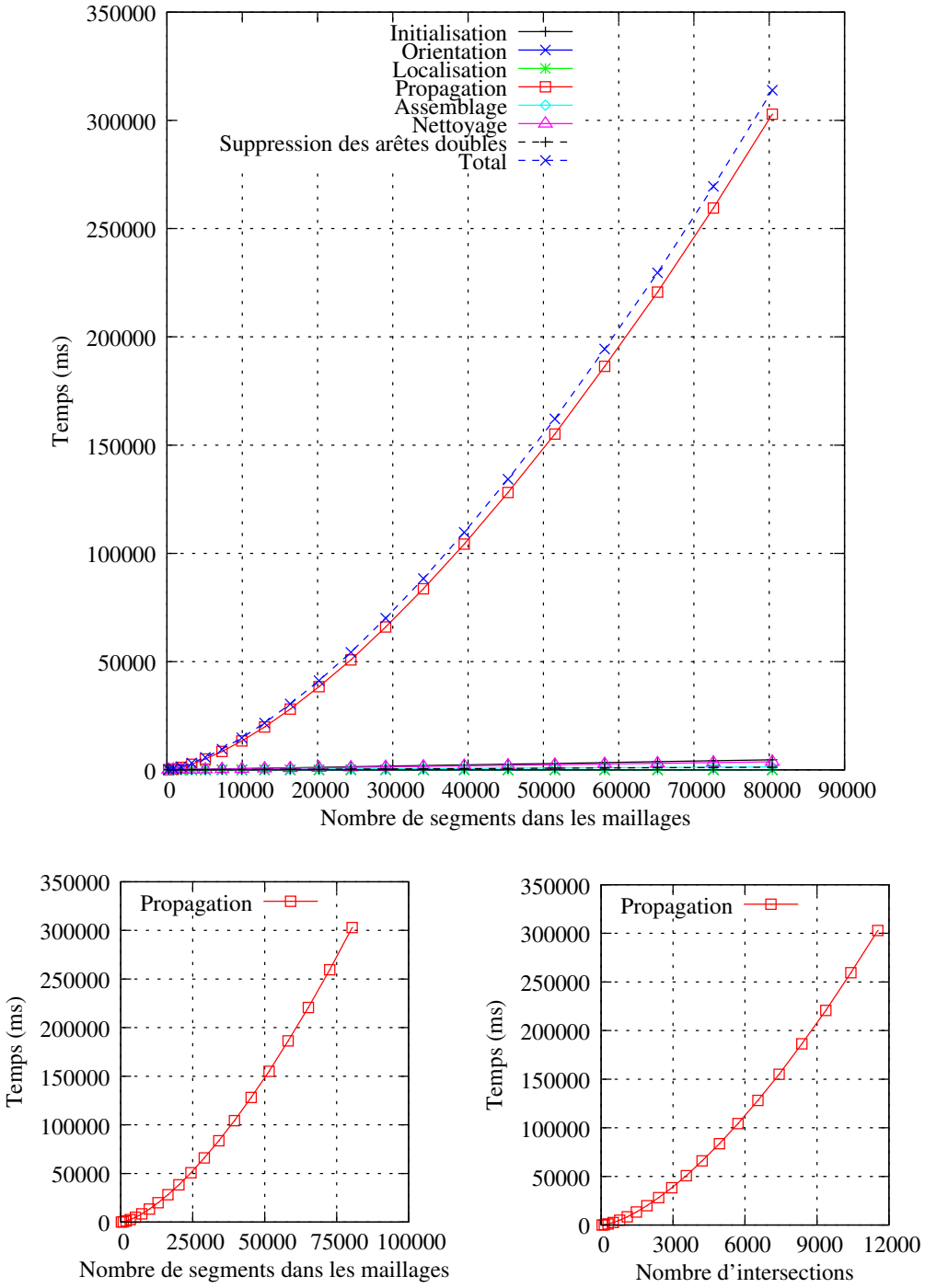


FIG. 3.23 – Évolution des temps de calcul pour le co-raffinement entre deux maillages identiques décalés l'un par rapport à l'autre.

Chapitre 4

Opérations booléennes

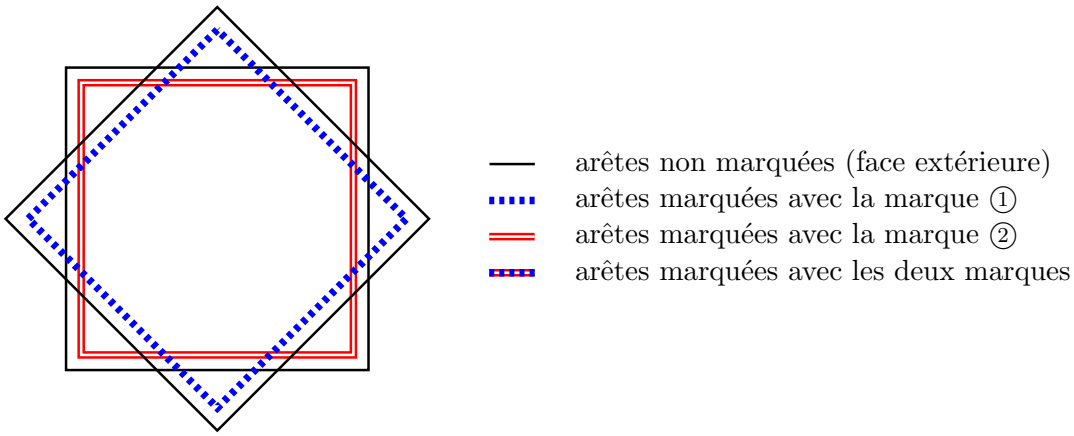
L'opération de co-raffinement décrite dans le précédent chapitre se contente de calculer toutes les intersections existantes entre deux objets sans se soucier de quelles sont les parties appartenant à chacun des objets. Pour extraire les résultats des opérations booléennes (union, soustraction, intersection) à partir du co-raffinement, nous devons connaître à la fin du traitement de quel objet provient chacune des faces générées. Ensuite, il est possible de déterminer à quel résultat appartient chaque face en fonction de sa provenance. Par exemple, si une face provient des deux objets, elle appartient à l'intersection.

Pour reconnaître la provenance de chaque face, il suffit de marquer les faces du premier objet (opérande A) avec une marque (notée ①) et les faces du second (opérande B) avec une autre marque (notée ②). Pour chacun des objets, nous marquons toutes les faces exceptées celles étant considérées comme des trous, comme par exemple la face définissant l'extérieur des objets.

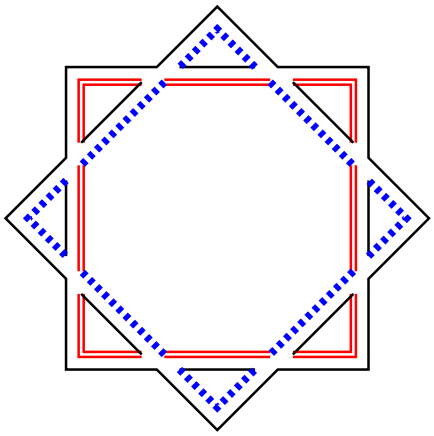
Nous pouvons ensuite appliquer l'opération de co-raffinement sur ces deux objets pour obtenir nos différentes composantes. Cependant, l'algorithme de co-raffinement génère de nouvelles cellules sur les objets afin de modéliser les intersections. Or, ces nouvelles cellules n'étant par défaut pas marquées par ① ou ②, le marquage final des objets n'est pas complet et ne permet donc pas de connaître les différents résultats. Pour pallier ce problème, il est nécessaire que l'algorithme de co-raffinement propage automatiquement ces marques lors de l'ajout de nouvelles cellules. De plus, les composantes obtenues à la suite du co-raffinement sont partiellement marquées, nous devons alors propager les marques afin d'obtenir le résultat final (cf. Fig. 4.1).

Pour le précédent algorithme, cette propagation doit être effectuée dans la fonction `EclaterArête` de l'algorithme 3.7 (page 37) qui permet d'insérer un sommet sur une arête. Pour cela, il suffit d'y rajouter un traitement s'occupant de marquer chaque nouveau brin inséré avec la marque de l'objet si son brin voisin par α_0 l'est aussi. Nous renommons alors cette fonction en `EclaterArêteBis` et l'utilisons aux mêmes endroits que la précédente (cf. Algo. 4.1). Pour finir, lorsque l'algorithme de co-raffinement est terminé, nous propageons les marques ① et ② aux orbites $<\alpha_0, \alpha_1>$ (i.e. aux faces) incidentes à au moins un brin marqué par cette marque.

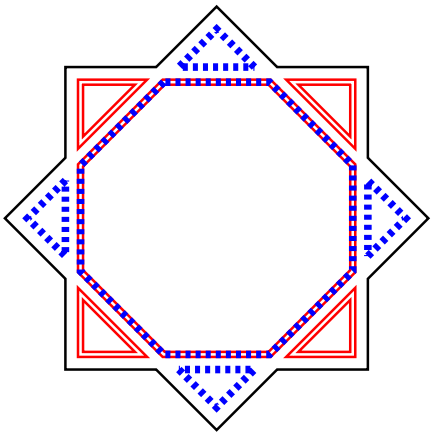
Lorsque ces traitements sont terminés, nous obtenons une décomposition de l'espace en composantes qui peuvent être marquées avec aucune, une ou deux marques. Les brins n'étant pas marqués appartiennent alors à l'union des deux objets, ceux étant marqués par une seule marque



(a) objets en vue semi-éclatée



(b) résultat du co-raffinement



(c) résultat après extension des marques

FIG. 4.1 – Étapes permettant d'obtenir les résultats des opérations booléennes.

ALGO. 4.1 – Fonction `EclaterArêteBis`

Entrées : ▪ un brin b d'une arête à éclater ;
▪ un point P correspondant à la position du nouveau sommet.

Sorties : Un brin du sommet inséré.

début
  $b_1 \leftarrow b$; $b_2 \leftarrow \alpha_2(b)$; $b_3 \leftarrow \alpha_0(b_1)$; $b_4 \leftarrow \alpha_0(b_2)$
  `Découdre`(b_1, α_0) ; `Découdre`(b_2, α_0)
  pour $i \in [5..8]$ **faire** $b_i \leftarrow$ `CréerBrin`
  `Coudre`(b_5, b_6, α_2) ; `Coudre`(b_7, b_8, α_2)
  `Coudre`(b_5, b_7, α_1) ; `Coudre`(b_6, b_8, α_1)
  pour $i \in [1..4]$ **faire**
    `Coudre`(b_i, b_{i+4}, α_0)
    si `EstMarqué`(b_i, ①) **alors** `Marquer`(b_{i+4}, ①)
    si `EstMarqué`(b_i, ②) **alors** `Marquer`(b_{i+4}, ②)
  `Plonger`(b_5, P)
  retourner b_7
fin

appartiennent à une soustraction et ceux étant marqués par les deux marques appartiennent à l'intersection.

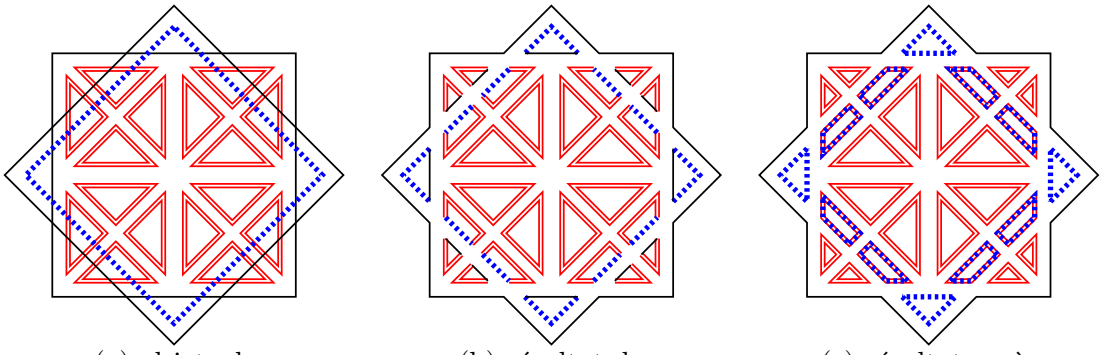
Cependant, si nous utilisons cette technique sur des objets maillés, nous n'obtenons pas le résultat escompté. En effet, si nous regardons le résultat obtenu sur la figure 4.2, nous voyons que seules les composantes ayant été modifiées lors de l'intersection sont prises en compte. Ainsi, les mailles du milieu qui devraient appartenir à l'intersection des deux objets ne sont pas marquées comme il le faudrait.

Pour résoudre ce problème, il est nécessaire de propager les marques à l'intérieur des maillages jusqu'à atteindre une frontière formée d'une arête dont les deux côtés ne possèdent pas les mêmes marquages. L'algorithme 4.2 donne les traitements à effectuer pour propager la marque ① en dimension 2 sur la première opérande. Celui-ci fonctionne de la manière suivante :

1. Nous commençons par stocker dans une file f tous les brins de la première opérande n'étant pas marqués par ① et dont leur image par α_1 l'est. Il s'agit donc de brins se trouvant sur des frontières entre deux résultats différents (cf. FIG. 4.2(b)).

2. Ensuite, nous utilisons la file f pour nous propager à l'intérieur du maillage. Nous effectuons une propagation topologique en récupérant le premier brin b' de la file f et en y insérant ses images par les involutions α_0, α_1 et α_2 si ses dernières ne sont pas marquées par ①. Cependant, nous devons rajouter quelques préconditions avant d'insérer les images du brin b' dans f. Le parcours s'arrête lorsque la file est vide.

3. En ce qui concerne les involutions α_0 et α_1, elles définissent les orbites $<\alpha_0, \alpha_1>$ correspondant aux faces. Sachant qu'une face du maillage ne peut pas appartenir à plusieurs résultats, si un brin de cette face est marqué par ①, les autres doivent l'être aussi. Ainsi,



(a) objets de départ (b) résultat du co-raffinement (c) résultat après extension des marques

FIG. 4.2 – Problèmes posés par les maillages lors de l'extension des marques.

ALGO. 4.2 – Procédure `PropagerPremièreMarque`

Entrées : Un brin b de la première opérande.
Résultat : La marque ① est mise à jour et propagée aux cellules appartenant au même résultat.

début
  $f \leftarrow FileVide$
1 **pour** tous les brins b' du maillage incident à b **faire**
    si `EstMarqué`(b', ①) = `faux` **et** `EstMarqué`($\alpha_1(b')$, ①) = `vrai` **alors**
      `Enfiler`(f, b')
2 **tant que** $f \neq \emptyset$ **faire**
    $b' \leftarrow$ `Défiler`(f)
    si `EstMarqué`(b', ①) = `faux` **alors**
      `Marquer`(b', ①)
3   **si** `EstMarqué`($\alpha_0(b')$, ①) = `faux` **alors** `Enfiler`(f, $\alpha_0(b')$)
      si `EstMarqué`($\alpha_1(b')$, ①) = `faux` **alors** `Enfiler`(f, $\alpha_1(b')$)
4   **si** `EstMarqué`($\alpha_2(b')$, ①) = `faux` **et** (`EstMarqué`(b', ②) = `faux` **ou**
      `EstMarqué`($\alpha_2(b')$, ②) = `vrai`) **alors**
        `Enfiler`(f, $\alpha_2(b')$)
fin

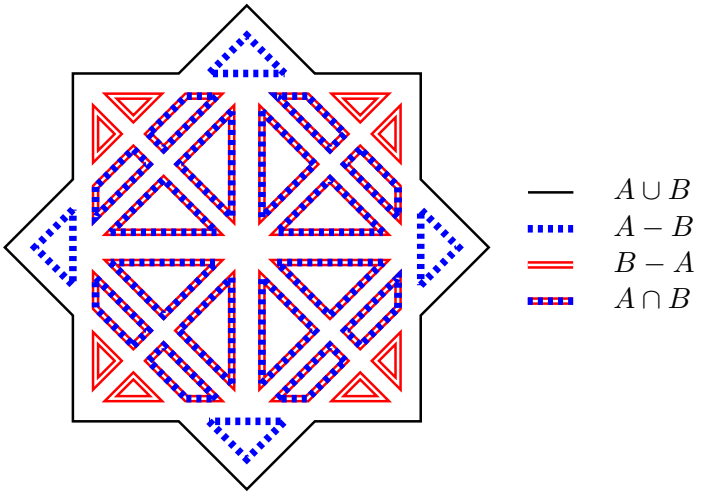


FIG. 4.3 – Résultat des opérations booléennes après une propagation des marques.

pour les images par les involutions α_0 et α_1, il n'y a pas de précondition supplémentaire avant de les ajouter dans la file f.

4. Les involutions α_2 représentent des passages entre deux faces adjacentes. Comme les résultats finaux sont représentés par un ensemble de faces, avant de rajouter une image par α_2 dans la file f, nous devons d'abord vérifier que celle-ci appartient toujours au même résultat. Ceci se fait simplement en comparant les marquages du brin b' et de son image par α_2. Si les deux brins sont marqués de la même manière avec la marque ②, nous restons dans le même résultat et nous pouvons alors ajouter le brin $\alpha_2(b')$ à la file f. Par contre, s'ils sont marqués de manière différente, il s'agit d'une frontière entre deux résultats et nous devons savoir de quel côté de celle-ci nous arrivons. Si le brin b' n'est pas marqué par la marque ②, nous nous trouvons à l'extérieur du deuxième objet et nous pouvons alors franchir la frontière afin d'y rentrer. Dans le cas contraire, nous sommes à l'intérieur du deuxième objet et nous ne pouvons pas en sortir. Comme nous pouvons le voir sur la figure 4.2(b), les frontières sont représentées par des arêtes marquées d'un côté par la marque ② et dont l'autre côté n'est pas marqué.

Lorsque la propagation de la marque ① est effectuée, nous devons alors faire de même pour la marque ②. Nous utilisons alors une procédure `PropagerDeuxièmeMarque` qui effectue les mêmes traitements que la procédure `PropagerPremièreMarque` mis à part qu'elle travaille sur la deuxième opérande et que les numéros des marques sont inversés. Au final, nous obtenons les quatre résultats des opérations booléennes comme nous pouvons le voir sur la figure 4.3. Notons par ailleurs que le principe de marquage expliqué dans ce chapitre est valable en toute dimension et peut être facilement étendu en dimension 3.

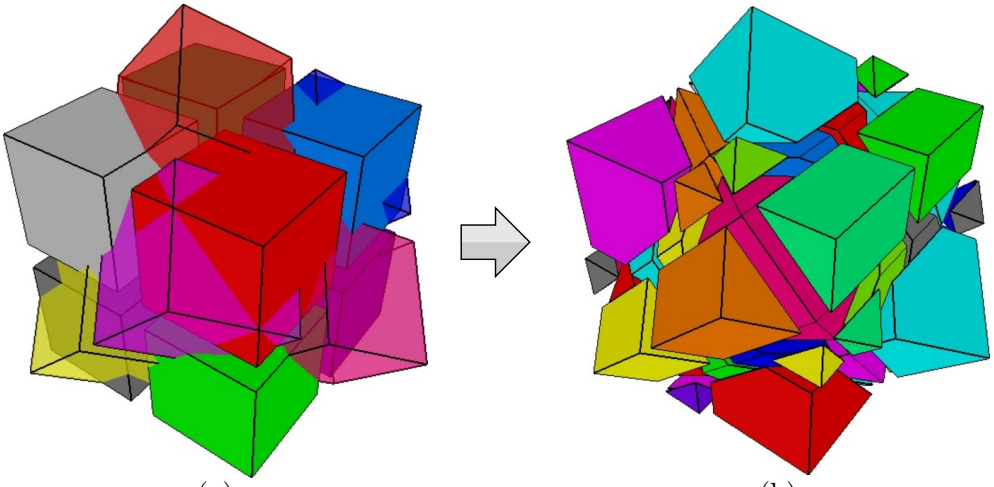
Deuxième partie

Deux algorithmes de co-raffinement en dimension 3

Chapitre 5

Introduction

La définition du co-raffinement en dimension 3 est similaire à celle du co-raffinement en dimension 2 et consiste à gérer une dimension supplémentaire. Son but est donc de calculer la subdivision de l'espace 3D résultant de l'intersection de deux subdivisions 3D originelles (cf. Fig. 5.1).



(a) (b)

Fig. 5.1 – Exemple de co-raffinement entre deux maillages 3D : (a) maillages originaux en vue éclatée ; (b) résultat du co-raffinement.

Les subdivisions en dimension 3 étant formées de volumes dont la surface est composée d'un ensemble de faces, le traitement principal consiste alors à calculer les intersections existantes entre les surfaces de ces volumes et plus précisément entre les faces qui les composent. Les intersections obtenues sont alors représentées sous la forme de segments de coupe.

La plupart des travaux existants dans ce domaine sont essentiellement basés sur la détermination du résultat des opérations booléennes entre deux volumes. Ils ne s'intéressent donc généralement pas aux méthodes permettant de détecter les intersections mais plutôt à celles permettant de les modéliser et de classifier les différentes parties d'un objet par rapport

à l'intérieur ou l'extérieur d'un autre. Ces travaux datent généralement des années 80 et il est assez difficile de les récupérer. De plus, ces méthodes permettent de ne traiter que des objets surfaciques (*i.e.* objets topologiques 2D ayant des plongements en 3D) et permettent de construire un unique volume à partir de deux.

Une première méthode [MT83] consiste à générer les intersections entre chaque face en s'appuyant sur le fait qu'une découpe divise une face en deux parties : un côté se trouvant à l'intérieur de l'objet et l'autre se trouvant à l'extérieur. L'idée principale de ce classement est de dire qu'une arête se trouve à l'intérieur (resp. à l'extérieur) d'un objet si ses deux extrémités le sont aussi. Ainsi, les arêtes ayant une extrémité de chaque côté sont découpées par la surface de l'autre objet. La création des arêtes de coupe se passe alors en deux étapes qui sont l'insertion de sommets sur les arêtes découpées et la liaison de ces sommets par des arêtes.

Une deuxième méthode [MT88, HHK88, KY92, GP96] consiste à calculer pour chaque face d'un objet les droites d'intersections avec les faces de l'autre objet. Le résultat obtenu correspond alors à un arrangement de droites dans le plan de la face. Ces droites sont ensuite intersectées et sont assemblées en utilisant l'orientation des faces les ayant générées afin de déterminer quels sont les morceaux devant être conservés.

Il existe aussi quelques algorithmes permettant de calculer des opérations booléennes en dimension n [PS86, TN87]. Ceux-ci fonctionnent généralement sur des structures moins explicites tels que des graphes d'incidences ou bien des arbres BSP. Leur fonctionnement est souvent basé sur la modélisation des intersections en effectuant une récursion sur la dimension des cellules.

Ici, nous nous intéressons à l'assemblage topologique d'objets en dimension trois et à la construction de la subdivision de l'espace résultant de cet assemblage. Les objets que nous manipulons peuvent donc être des maillages volumiques, c'est-à-dire des objets topologiques 3D ayant des plongements dans un espace 3D. À l'heure actuelle, très peu de travaux existent dans ce domaine. Les auteurs de [Caz97, CD99] présentent des méthodes formelles permettant de résoudre ce problème ainsi que les algorithmes associés. Cependant, ils ne présentent que peu de résultats concrets, qui sont la plupart du temps relativement simples.

Dans cette partie, nous présentons deux algorithmes de co-raffinement 3D basés sur deux techniques différentes. Les deux algorithmes proposent des méthodes permettant de traiter des maillages volumiques dont les faces peuvent être quelconques mais planes. Le but de ces algorithmes n'est pas de résoudre les problèmes d'erreurs numériques pouvant intervenir mais de pouvoir traiter les différentes configurations topologiques se produisant. Le premier d'entre eux s'appuie sur une extension de l'algorithme 2D et utilise la topologie pour générer les lignes de coupe localement sur la surface des objets (cf. chapitre 6). Le second utilise une technique fréquemment utilisée dans les algorithmes d'opérations booléennes sur des objets surfaciques et l'étend au cas des maillages topologiques 3D (cf. chapitre 7).

Chapitre 6

Co-raffinement 3D par suivi de lignes de coupe

Nous présentons ici un premier algorithme permettant de réaliser une telle opération. Nous commençons par donner le principe sur lequel nous nous appuyons dans la section 6.1. Nous expliquons ensuite le raisonnement que nous avons suivi dans la section 6.2. Nous discutons des avantages et des inconvénients de l'algorithme dans la section 6.3. Nous finissons par donner quelques résultats dans la section 6.4.

6.1 Principe

L'algorithme de co-raffinement 2D présenté dans le chapitre 3 peut en théorie être étendu en dimension supérieure. Nous pourrions alors envisager d'élever à la dimension 3 tous les traitements effectués dans ce dernier. Ainsi, l'algorithme consisterait à se propager dans les volumes d'un objet en suivant les arêtes ou les faces d'un autre objet.

Cependant, comme nous l'avons vu précédemment, ce type d'algorithme est le plus efficace dans le cas de maillages denses dont la taille des cellules est faible (cf. section 3.3). Or, en dimension 3, nous voulons être capables de traiter aussi bien des objets volumiques (maillages) que des objets surfaciques (volumes isolés). Dans la plupart des cas, les objets rencontrés sont très souvent composés de volumes dont la surface comporte un très grand nombre de faces. Dans ce cas, la technique de propagation n'est pas adaptée car chaque recherche d'intersection entre un segment ou une face et le bord d'une volume consisterait à tester toutes les faces définissant la surface du volume.

Néanmoins, il est tout de même possible d'utiliser le principe de propagation lors de la création des intersections. En dimension 3, ces dernières possèdent une particularité très intéressante car il s'agit de lignes ou bien de graphes de coupe. En effet, comme l'intersection entre deux faces peut être modélisée à l'aide d'un ou plusieurs segments de droites et que les faces d'un même objet sont connexes, les segments de coupe le sont aussi et forment par conséquent des graphes (cf. Fig. 6.1).

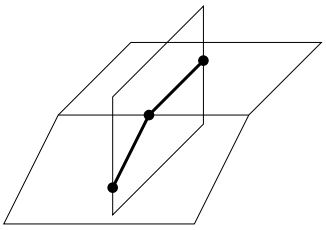


FIG. 6.1 – Exemple de segments de coupe connexes formant un graphe.

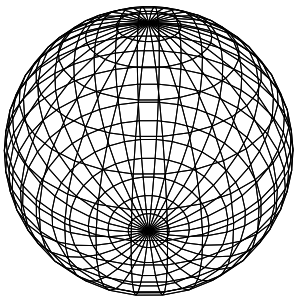


FIG. 6.2 – Exemple d'objet surfacique. Sa surface peut être considérée comme un graphe 2D plongé en dimension 3.

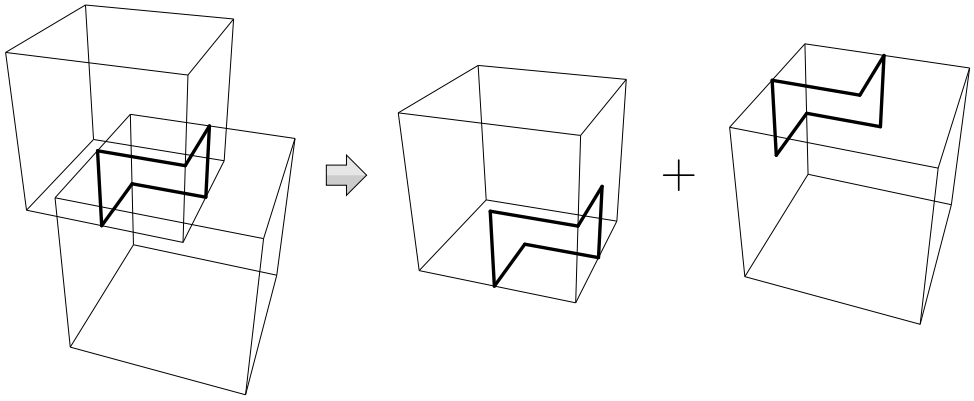


FIG. 6.3 – Exemple d'intersection entre deux cubes. La trace de la découpe forme un graphe commun aux deux objets.

Si nous considérons l'intersection entre deux objets, nous pouvons voir que celle-ci forme des graphes de coupe équivalents sur la surface de chaque objet (cf. Fig. 6.3). Comme la surface des objets est formée de sommets, d'arêtes et de faces, elle peut tout aussi bien être considérée comme un graphe ou un maillage 2D plongé en 3D qui est la plupart du temps dense et dont le degré des faces est faible (cf. Fig. 6.2). De plus, les graphes de coupe se trouvant sur la surface des objets, il est possible d'utiliser un algorithme de co-raffinement 2D par propagation afin de construire rapidement les intersections entre ces deux entités.

En utilisant un algorithme de propagation 2D pour détecter les intersections entre les graphes de coupe et les cellules appartenant à la surface des objets, nous pouvons connaître exactement les faces par lesquelles passent les graphes. Comme un graphe de coupe est commun aux deux objets, il est alors possible de connaître localement quelles sont les faces qui sont en intersection entre les deux objets (*i.e.* faces partageant le même segment du graphe de coupe). De plus, les graphes de coupe provenant de l'intersection entre les faces des objets, nous pouvons aussi, à partir d'un point d'intersection connu (*i.e.* un noeud du graphe de coupe), tester les intersections existantes entre les faces incidentes à celui-ci et déterminer ainsi les arcs du graphe incidents à ce noeud.

L'algorithme présenté dans ce chapitre consiste donc à générer ces graphes de coupe de manière locale en utilisant une méthode de propagation 2D. Nous allons maintenant expliquer le raisonnement que nous avons suivi afin de le réaliser.

6.2 Raisonnement

Pour construire un graphe de coupe, nous devons connaître un premier élément de départ qui peut être soit un sommet, soit un segment. Pour cela, nous devons donc rechercher une première intersection entre nos deux objets. Cependant, pour pouvoir déterminer un segment de coupe, nous devons aussi déterminer ses deux sommets. Le plus simple est alors de rechercher directement un point d'intersection entre les deux objets car les arêtes du graphe de coupe seront déterminées lors de l'étape suivante.

Nous présentons donc tout d'abord comment déterminer un premier point d'intersection entre les deux objets, puis nous expliquons comment générer le graphe de coupe à partir de ce point. Dans la suite, nous considérons que nous générons les intersections entre deux objets notés O_1 et O_2.

6.2.1 Détection d'un premier point d'intersection

Un sommet du graphe de coupe (*i.e.* un point d'intersection entre les deux objets) peut être le résultat d'une intersection entre une cellule de dimension 0 ou 1 (un sommet ou une arête) et une cellule de dimension 0, 1 ou 2 (un sommet, une arête ou une face). Pour simplifier la recherche, nous avons décidé de rechercher les intersections existantes entre les arêtes de O_1 et les cellules de O_2 puis, lorsqu'une intersection est détectée, de tester si celle-ci se produit avec une extremité de l'arête.

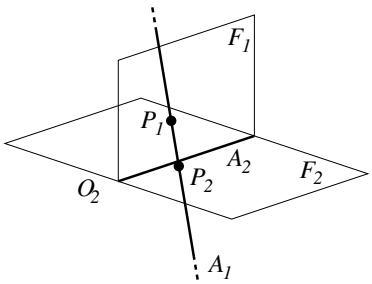


FIG. 6.4 – Contradiction lors de tests d'intersection entre une arête A_1 de l'objet O_1 et des cellules de O_2 : $A_1 \cap F_1 = P_1$; $A_1 \cap F_2 = P_2$; $\|P_1A_2\| > \mathcal{E}$ et $\|P_2A_2\| < \mathcal{E} \Rightarrow A_1 \cap A_2 = ?$

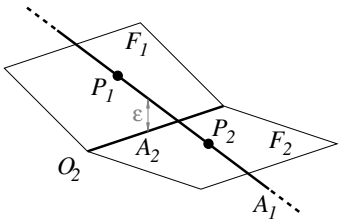


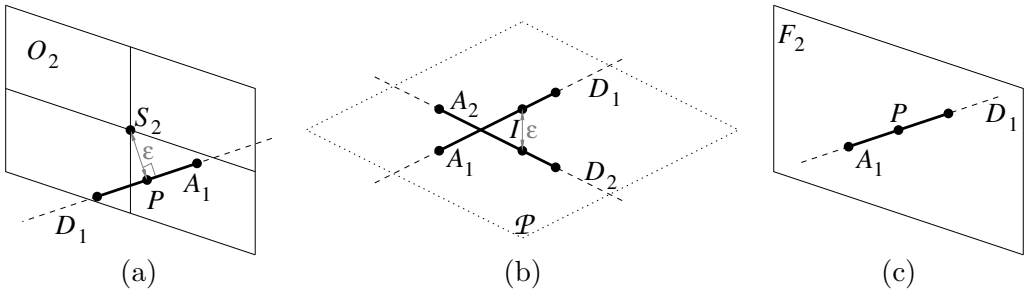
FIG. 6.5 – Exemple de configuration dans laquelle l'arête A_1 est proche de A_2 à $\mathcal{E}$ près alors que les points d'intersection P_1 et P_2 avec les faces F_1 et F_2 ne le sont pas.

Pour effectuer cette recherche, nous parcourons donc toutes les arêtes de O_1 et nous testons si chacune d'entre elles possède une intersection avec une cellule de O_2. Lorsqu'une intersection est détectée, il nous suffit d'exécuter la deuxième phase de l'algorithme afin de construire un nouveau graphe de coupe à partir de ce point. Comme les sommets représentent les bords des arêtes et que les arêtes représentent les bords des faces, une méthode rapide pour effectuer ces tests consiste à rechercher tout d'abord s'il existe une intersection entre l'arête de O_1 et une face F de O_2. Si c'est le cas, nous testons ensuite si cette dernière se produit sur une arête de F puis, si elle se produit sur un sommet de cette arête. Nous obtenons ainsi le résultat souhaité en limitant le nombre de tests.

Cependant, comme nous ne travaillons pas en arithmétique exacte et que les nombres que nous manipulons sont approximés par des flottants, cette technique pose des problèmes numériques. En effet, pour déterminer si un point se trouve sur une arête ou bien si deux points sont confondus, nous devons calculer la distance euclidienne entre les deux entités et tester si cette distance est inférieure à une valeur $\mathcal{E}$ donnée. Ainsi, si nous recherchons une intersection entre une face F_1 de O_1 et les n faces incidentes à une arête A_2 de O_2, nous obtenons au plus un point d'intersection par face, soit n' points ($0 \leq n' \leq n$). Sur ces n' points, n'' points seront à proximité de A_2 ($0 \leq n'' \leq n'$). Dans le cas où $n'' < n$, nous ne pouvons pas conclure sur le type de l'intersection. Les figures 6.4 et 6.5 montrent des exemples dans lesquels une contradiction existe entre le résultat trouvé et celui devant être trouvé.

Pour pallier ce problème, nous utilisons une autre méthode qui consiste à effectuer trois parcours au lieu d'un seul. Nous commençons par rechercher les intersections existantes entre les arêtes de O_1 et les sommets de O_2. Nous notons les intersections trouvées et nous marquons les cellules incidentes (arêtes et faces) afin de ne pas les tester dans les parcours suivants. Ensuite, nous recherchons les intersections existantes entre les arêtes de O_1 et les arêtes non marquées de O_2. Comme précédemment, nous notons ces intersections et nous marquons les faces incidentes. Nous finissons le traitement en recherchant les intersections existantes entre les arêtes de O_1 et les faces non marquées de O_2. Cette technique nous permet ainsi de déterminer plus précisément les types des intersections trouvées. Par exemple, si une intersection avec une face est trouvée, nous sommes sûr que celle-ci ne se produit pas avec un sommet ou bien une arête de cette face.

De plus, cette méthode permet d'accélérer la recherche des intersections en limitant le nombre de tests. En effet, lorsque nous trouvons une intersection avec une arête ou bien une face du



(a) (b) (c)

FIG. 6.6 – Configurations valides lors d'un test d'intersection entre une arête A_1 du premier objet et une cellule du second objet : (a) intersection avec un sommet S_2 ; (b) intersection avec une arête A_2 ; (c) intersection avec une face F_2.

second objet, nous devons générer un nouveau point d'intersection. Ainsi, nous savons que ce dernier correspond au point de départ d'un nouveau graphe de coupe. Par contre, si nous trouvons une intersection avec un sommet (ce qui correspond au premier test effectué), soit ce dernier appartient à un graphe de coupe déjà existant et dans ce cas nous ne devons pas le générer de nouveau, soit il s'agit d'un sommet d'origine de l'objet O_2 et dans ce cas celui-ci correspond à un nouveau point de départ. Le fait de tester en premier les sommets nous permet alors de connaître rapidement les intersections déjà calculées sans avoir besoin de tester les cellules de dimension supérieure.

Pour chaque type de cellule testée, nous devons effectuer des traitements particuliers aussi bien pour détecter s'il y a intersection que pour exécuter la deuxième phase de l'algorithme. Dans la suite, nous considérons que nous testons une arête A_1 de l'objet O_1 avec une cellule de l'objet O_2. Nous notons de plus D_1 la droite support de l'arête A_1 et $\mathcal{E}$ une constante proche de 0. En fonction du type de cellule testée sur l'objet O_2, Nous obtenons les traitements suivants :

- **Sommets :** Nous considérons que A_1 est en intersection avec un sommet S_2 de O_2 si la distance[1] entre D_1 et S_2 est inférieure à $\mathcal{E}$, et que la projection orthogonale P de S_2 sur D_1 se trouve entre les extrémités de A_1 (cf. FIG. 6.6(a)).
  Dans ce cas, si S_2 appartient déjà à un graphe de coupe, nous n'effectuons aucun traitement. Sinon, nous conservons ce sommet.

- **Arêtes :** Nous considérons que A_1 possède une intersection avec une arête A_2 de O_2 si les conditions suivantes sont vérifiées (cf. FIG. 6.6(b)) :
  - la distance[2] entre D_1 et la droite D_2 support de A_2 est inférieure à $\mathcal{E}$,
  - D_1 et D_2 ne sont pas parallèles,
  - le point d'intersection I entre D_1 et D_2 se trouve entre les extrémités de chacune des arêtes A_1 et A_2. Le point I est calculé dans le plan $\mathcal{P}$ formé par les vecteurs directeurs des droites D_1 et D_2.

[1] La distance entre un point et une droite est déterminée en calculant la distance euclidienne entre le point et sa projection orthogonale sur la droite.

[2] La distance entre deux droites correspond à la plus petite distance existante entre deux points de chaque droite. Pour deux droites non-colinéaires, elle peut être déterminée en calculant la distance euclidienne entre un point d'une droite et sa projection orthogonale sur le plan formé par les vecteurs directeurs des droites et passant par un point de l'autre droite.

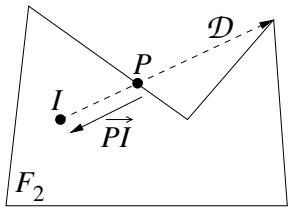


FIG. 6.7 – Méthode utilisée pour détecter si un point de trouve dans les limites d'une face.

Dans ce cas, nous éclatons l'arête A_2 en I et nous obtenons un nouveau sommet noté S_2. Notons ici que S_2 ne peut pas être confondu avec les extrémités de A_2 car ce type de cas est détecté lors du premier parcours.

- **Faces :** Nous considérons que A_1 possède une intersection avec une face F_2 de O_2 si les conditions suivantes sont vérifiées (cf. FIG. 6.6(c)) :
  - il existe un point d'intersection I entre D_1 et le plan support de F_2,
  - I se trouve entre les extrémités de A_1,
  - I se trouve dans les limites de F_2. Pour effectuer ce test, nous traçons une demi-droite $\mathcal{D}$ dans le plan de F_2 entre I et un sommet de F_2, puis nous cherchons la plus proche intersection P avec le bord de F_2. Il est ensuite possible de déterminer si I se trouve dans F_2 en testant si le vecteur $\overrightarrow{PI}$ désigne l'intérieur de la face[3] (cf. FIG. 6.7).

  Dans le cas où l'intersection est valide, nous devons insérer un sommet S_2 dans la face F_2. Comme précédemment, nous savons que ce sommet ne se trouve pas sur un sommet ou une arête de F_2 car ces cas sont détectés lors des premiers parcours.

À la fin de la recherche, nous pouvons nous trouver dans deux cas de figure. Soit nous n'avons pas trouvé d'intersection et dans ce cas, nous pouvons continuer la recherche avec une autre arête de O_1. Soit nous avons détecté une intersection et nous avons à notre disposition un sommet S_2 de l'objet O_2 correspondant au point d'intersection. Nous regardons alors si S_2 est proche (à $\mathcal{E}$ près) d'une des extrémités de A_1. Si ce n'est pas le cas, nous éclatons A_1 en S_2 et obtenons un nouveau sommet S_1. Sinon, nous notons S_1 l'extrémité de A_1 concernée. Nous pouvons ensuite exécuter la deuxième phase de notre algorithme qui permet de créer un graphe de coupe en partant des sommets S_1 et S_2.

6.2.2 Création d'un graphe de coupe

Pour générer un graphe de coupe, nous devons partir d'une intersection existante. Nous supposons donc qu'un premier point d'intersection à été détecté grâce à l'étape précédente et que nous avons à notre disposition deux sommets S_1 et S_2 confondus qui appartiennent respectivement aux objets O_1 et O_2. Ces sommets correspondent au premier noeud N du graphe et le reste de celui-ci est construit incrémentalement à l'aide de la méthode détaillé ci-après.

Comme nous l'avons vu précédemment dans la section 6.1, les arcs du graphe correspondent à des segments de coupe entre les faces des deux objets. Ainsi, les arcs incidents au noeud N

[3]La manière de déterminer un vecteur désignant l'intérieur d'une face est donnée par la l'algortihme 7.4 à la page 98.

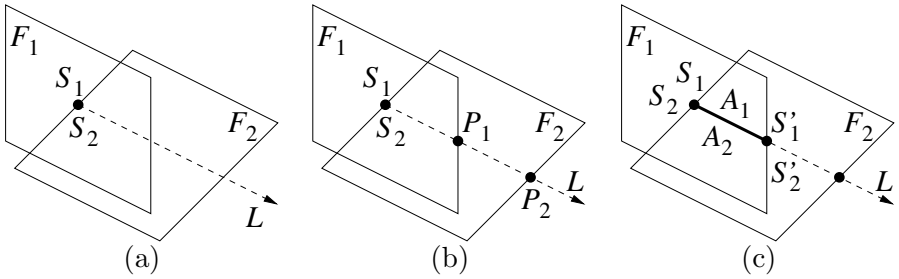


FIG. 6.8 – Les différentes étapes de création d'une ligne de coupe : (a) calcul de la demi-ligne de coupe L entre les faces F_1 et F_2 ; (b) recherche des plus proches points d'intersections P_1 et P_2 entre L et les bords de F_1 et F_2 ; (c) création d'un segment de coupe commun aux deux faces.

correspondent à des intersections entre les faces incidentes aux sommets S_1 et S_2. Pour générer les arcs incidents à N, nous prenons chaque face F_1 incidente à S_1 et nous testons s'il existe une intersection avec chaque face F_2 incidente à S_2 à l'aide des traitements suivants :

1. Nous calculons la demi-ligne de coupe[4] notée L entre F_1 et F_2 et ayant pour origine les sommets S_1 et S_2 (cf. FIG. 6.8(a)). Si L n'est dirigée vers l'intérieur d'aucune des deux faces, nous inversons sa direction.

2. Si la ligne de coupe L n'est toujours dirigée vers l'intérieur d'aucune des deux faces F_1 et F_2, il n'y a pas d'intersection possible (*i.e.* les faces ne se coupent pas). Nous passons alors au couple de faces suivant. Dans le cas contraire, nous calculons les points d'intersection P_1 et P_2 entre L et les bords respectifs des faces F_1 et F_2. Nous conservons uniquement les points les plus proches de S_1 et S_2 (cf. FIG. 6.8(b)).

3. Si P_1 est plus proche que P_2 des sommets S_1 et S_2, nous éclatons le bord de F_1 en P_1 et obtenons un nouveau sommet S'_1. Nous éclatons ensuite F_1 entre les sommets S_1 et S'_1 et obtenons une nouvelle arête A_1. Enfin, nous insérons dans F_2 une arête A_2 confondue avec A_1. Cette arête a pour extremité S_2 et un nouveau sommet S'_2 confondu avec S'_1. Nous pouvons voir toutes ces opérations sur la figure 6.8(c).
   Si P_2 est plus proche que P_1 des sommets S_1 et S_2, nous effectuons le traitement inverse. Si P_1 et P_2 se trouvent à la même distance de S_1 et S_2, nous effectuons le premier traitement pour les deux faces (*i.e.* nous éclatons le bord et insérons une arête).

4. Pour que l'objet final corresponde à une subdivision valide de l'espace (*i.e.* une subdivision sans auto-intersection), nous trions de manière angulaire et relions entre elles les faces incidentes aux arêtes A_1 et A_2. L'algorithme d'interclassement utilisé est équivalent à celui-ci du co-raffinement 2D (cf. section 3.2.1.3) mis à part que le tri est opéré à l'aide des normales aux faces.

Lorsque l'intersection est réalisée et que les arêtes A_1 et A_2 sont insérées, nous récupérons leurs autres extrémités (*i.e.* les sommets S'_1 et S'_2). Nous stockons ces dernières en tant que nouveau couple de sommets dans une file $\mathcal{F}$ contenant l'ensemble des couples de sommets restant

[4] Le vecteur directeur de la demi-ligne de coupe est déterminé en calculant le produit vectoriel entre les normales aux faces F_1 et F_2.

à traiter. Ces sommets ne sont autres que de nouveaux noeuds du graphe de coupe. Bien entendu, si les sommets S'_1 et S'_2 sont des sommets déjà existants dans le graphe, il ne sont pas ajoutés à $\mathcal{F}$.

Lorsque toutes les faces incidentes aux sommets S_1 et S_2 ont été traitées, nous retirons le premier couple de la file $\mathcal{F}$ et recommençons les traitements jusqu'à ce que $\mathcal{F}$ soit vide. Dans ce dernier cas, cela signifie que la création du graphe de coupe est terminée et que nous pouvons revenir dans la première phase de l'algorithme afin de trouver d'autres points d'intersection et de générer les graphes de coupe correspondants.

6.3 Avantages et inconvénients

Le principal avantage de cet algorithme est qu'il permet de générer une ligne de coupe très rapidement car il est basé sur le principe utilisé pour le co-raffinement par propagation 2D. Cependant, contrairement à l'algorithme 2D, la localisation d'un premier point d'intersection n'est pas aussi rapide car elle implique de tester toutes les arêtes du premier objet avec les cellules du second. Comme ce traitement est inévitable et qu'il est très couteux en temps de calcul (en moyenne 90% du temps de calcul global), le fait de pouvoir générer très rapidement un graphe de coupe n'est pas intéressant. Pour optimiser cette recherche, nous avons utilisé des structures accélératrices comme des grilles régulières. Cela nous a permis de baisser grandement les temps de calcul mais pas suffisamment en comparaison d'autres algorithmes déjà existants (cf. section 6.4).

L'algorithme possède de plus quelques lacunes. Comme la recherche des intersections consiste à tester les arêtes du premier objet avec les cellules du second, il est possible (en fonction des objets traités) qu'aucune arête du premier objet ne croise une cellule du second alors que les objets possèdent des intersections (cf. Fig. 6.9). Dans ce cas, aucune intersection n'est calculée et le résultat est faux.

Une autre lacune de cet algorithme, est qu'il ne permet pas de gérer des intersections entre deux faces coplanaires. En effet, l'algorithme est conçu pour travailler localement sur les faces en suivant les lignes de coupe. Or, pour traiter deux faces coplanaires, les traitements à effectuer doivent être globaux tant au niveau de la détection de la coplanarité que de sa gestion (*i.e.* comme les faces sont planes, des parties de faces ne peuvent pas être coplanaires tandis que d'autres ne le sont pas). Une manière de gérer les faces coplanaires serait alors d'effectuer un pré-traitement permettant de les détecter et de les traiter, mais ceci rendrait l'algorithme encore moins efficace qu'il ne l'est.

La localité des traitements pose de plus un problème d'optimisation car comme le bord des faces peut être concave, il est possible que des faces soient traitées plusieurs fois entre elles. Il est aussi très difficile de tester localement si une intersection à déjà été calculée et par conséquant l'unicité des intersections n'est pas complètement garantie.

Étant donné le grand nombre d'inconvénients relatifs à cette méthode, et aussi du fait de la complexité de certaines parties de l'algorithme, nous n'avons pas jugé utile de le décrire sous forme de pseudo-code.

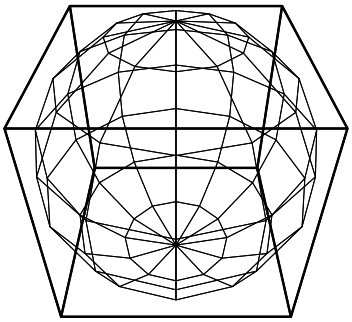


FIG. 6.9 – Exemple d'objets pour lesquels le co-raffinement ne fonctionne pas toujours. Si le premier objet considéré est le cube, ses arêtes ne possèdent aucune intersection avec les faces de la sphère.

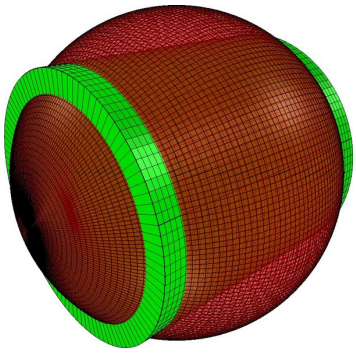


FIG. 6.10 – Objets utilisés pour réaliser les tests de performances sur la résolution de la grille régulière d'optimisation.

6.4 Résultats

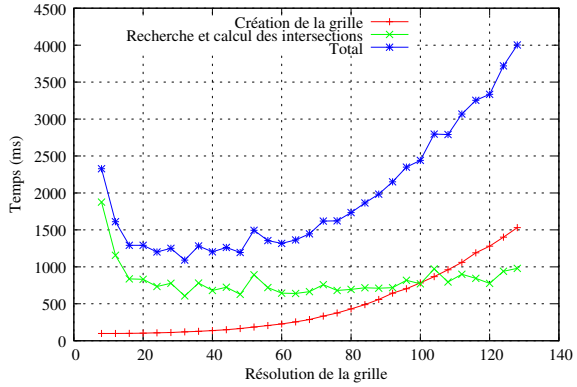
Nous présentons ici quelques résultats obtenus avec l'algorithme décrit dans ce chapitre. Nous commençons par donner une estimation de la résolution de la grille régulière à utiliser pour optimiser la recherche des intersections. Ensuite, pour une résolution de grille fixée, nous donnons les résultats obtenus sur différents objets. Tous les résultats présentés ici ont été obtenus sur une machine PC dotée d'un processeur Athlon 64 3000+ et de 512 Mo de mémoire vive.

6.4.1 Résolution de la grille régulière

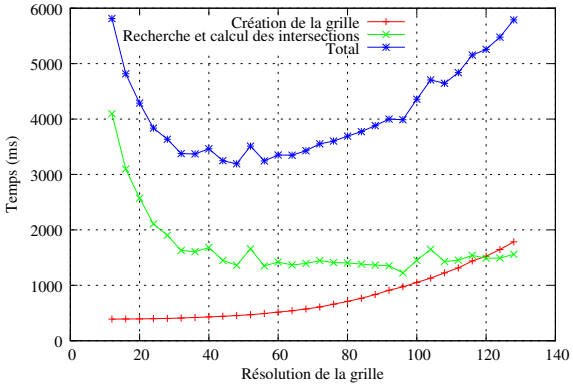
Pour accélerer la recherche des intersections, nous classons les faces du deuxième objet dans une grille régulière 3D. Chaque case de la grille contient la liste des faces dont la boîte englobante chevauche la case. Lorsque nous recherchons une intersection entre une arête de O_1 et une cellule de O_2, nous parcourons alors les cases de la grille correspondant à la boîte englobante de l'arête et nous ne testons que les cellules se trouvant dans ces cases.

Afin d'obtenir les meilleures performances, nous avons effectué des tests dans le but de déterminer une résolution optimale pour la grille. Pour cela, nous avons calculé le co-raffinement entre un cylindre et une sphère (cf. FIG. 6.10) en faisant varier la résolution de la grille. Nous avons réalisé cette expérience avec des objets de résolutions différentes et nous avons obtenu les courbes de la figure 6.11. Nous pouvons voir sur les différentes courbes que l'évolution du temps de calcul global en fonction de la résolution de la grille change selon le nombre de faces présentes sur la sphère. Cependant, nous pouvons aussi voir que les courbes possèdent des minimums pour une résolution de grille comprise entre 40^3 et 80^3. De plus, nous pouvons observer pour les dernières courbes que le temps de calcul global n'évolue plus à partir d'une résolution de 80^3.

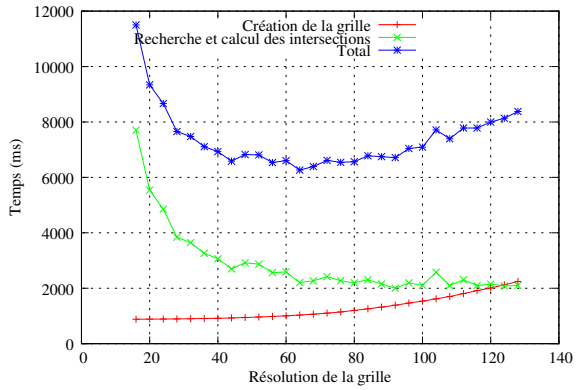
Dans ces conditions, nous avons décidé de prendre une résolution de grille de 64^3 quelle que soit la taille des objets traités. Ce choix n'est pas pénalisant pour les objets ayant une faible résolution étant donnée l'infime différence de temps de calcul existante entre des résolutions de



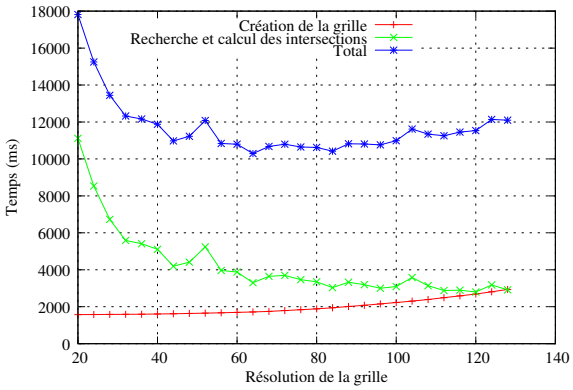
(a) sphère de 840 faces



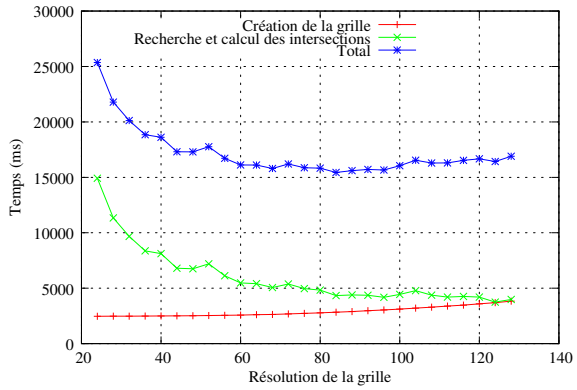
(b) sphère de 3280 faces



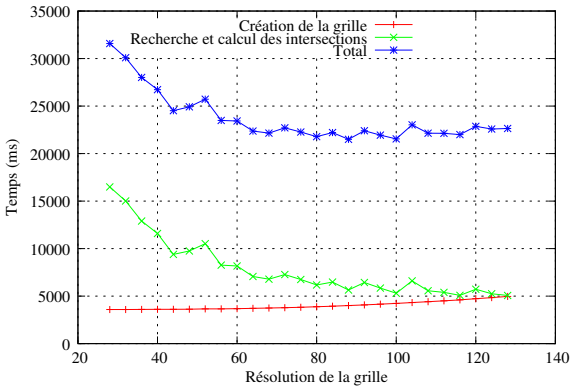
(c) sphère de 7320 faces



(d) sphère de 12960 faces



(e) sphère de 20200 faces



(f) sphère de 29040 faces

FIG. 6.11 – Détermination d'une résolution optimale pour la grille utilisée lors du co-raffinement de volumes. L'objet stocké dans la grille est une sphère dont le nombre de faces varie. Les courbes ci-dessus correspondent aux temps de calcul relevés lors du co-raffinement de la sphère et d'un cylindre.

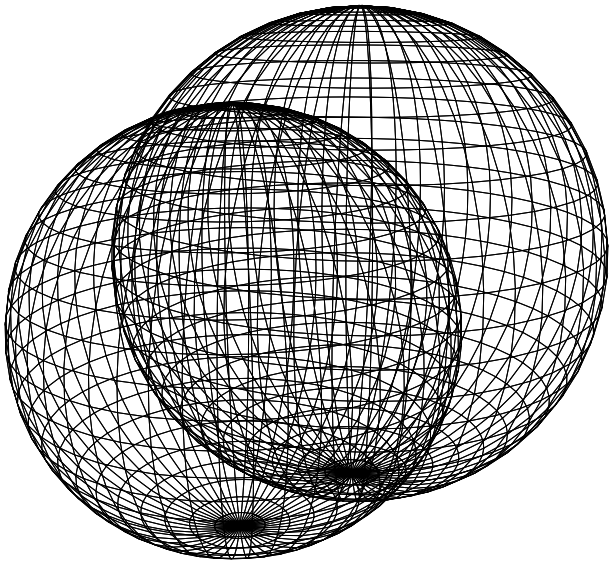


FIG. 6.12 – Objets utilisés pour effectuer des tests de performance.

grille de 40^3 et 64^3. Il en est de même si l'on regarde les différences de temps obtenues sur des gros objets pour des résolutions de 64^3 et 80^3.

6.4.2 Co-raffinement de divers objets

Pour réaliser les tests de performance de notre algorithme, nous avons utilisé des objets classiques, à savoir des sphères (cf. FIG. 6.12). Nous avons fait varier progressivement le nombre de faces sur les sphères et nous avons obtenu les courbes de la figure 6.13 qui montrent l'évolution des temps de calcul. De plus, le tableau 6.1 donne le détail des temps obtenus pour chaque partie de l'algorithme :

$\#F$ Nombre de faces sur chaque sphère.

t_i Durée de l'initialisation des objets.

t_g Durée de la création de la grille d'optimisation.

t_r Durée de la recherche des points de départ pour les lignes de coupe.

t_l Durée de la création des lignes de coupe.

t_m Durée de la mise à jour de la topologie.

t_t Temps de calcul total.

Nous pouvons alors observer que le temps de calcul global reste linéaire en fonction du nombre de faces. Ce phénomène est dû au fait que nous utilisons une grille régulière pour stocker les faces d'une des deux sphères. Ainsi, la recherche des intersections entre les arêtes de la première sphère et les faces de la seconde est optimisée.

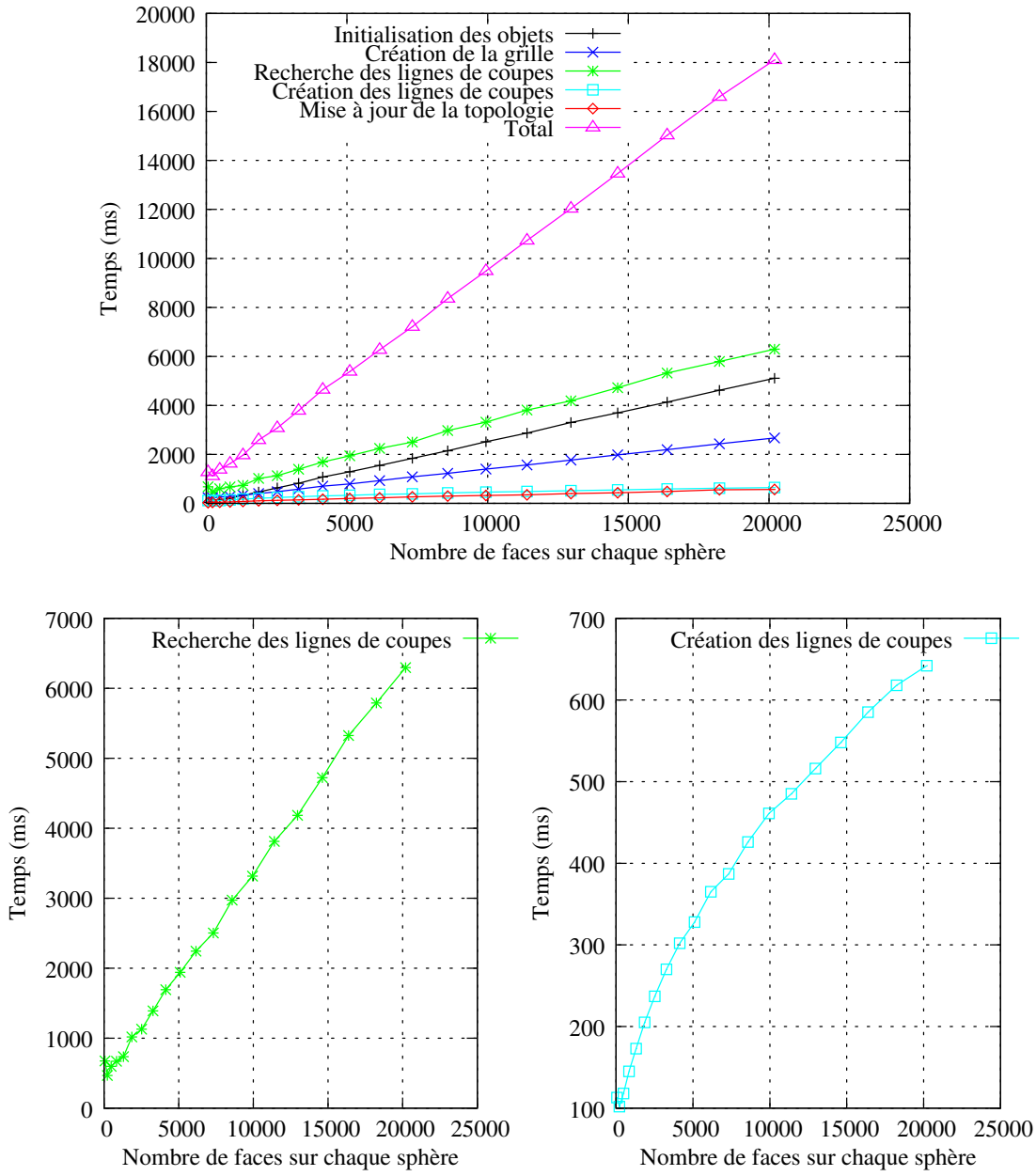


FIG. 6.13 – Évolution des temps de calcul pour le co-raffinement entre deux sphères dont le nombre de faces varie.

$\#F$	t_i	t_g	t_r	t_l	t_m	t_t
60	41 ms	230 ms	674 ms	113 ms	26 ms	1 s 275 ms
220	77 ms	225 ms	468 ms	102 ms	38 ms	1 s 116 ms
480	140 ms	255 ms	599 ms	118 ms	45 ms	1 s 380 ms
840	213 ms	279 ms	670 ms	145 ms	61 ms	1 s 625 ms
1300	327 ms	336 ms	735 ms	173 ms	76 ms	1 s 968 ms
1860	465 ms	402 ms	1 s 19 ms	205 ms	101 ms	2 s 588 ms
2520	629 ms	477 ms	1 s 129 ms	237 ms	123 ms	3 s 76 ms
3280	821 ms	581 ms	1 s 390 ms	270 ms	145 ms	3 s 788 ms
4140	1 s 74 ms	704 ms	1 s 692 ms	302 ms	169 ms	4 s 643 ms
5100	1 s 287 ms	796 ms	1 s 942 ms	328 ms	202 ms	5 s 380 ms
6160	1 s 545 ms	929 ms	2 s 243 ms	365 ms	223 ms	6 s 272 ms
7320	1 s 840 ms	1 s 85 ms	2 s 504 ms	387 ms	263 ms	7 s 209 ms
8580	2 s 150 ms	1 s 227 ms	2 s 974 ms	426 ms	295 ms	8 s 359 ms
9940	2 s 517 ms	1 s 401 ms	3 s 317 ms	461 ms	322 ms	9 s 490 ms
11400	2 s 868 ms	1 s 569 ms	3 s 812 ms	485 ms	349 ms	10 s 732 ms
12960	3 s 305 ms	1 s 766 ms	4 s 184 ms	516 ms	400 ms	12 s 37 ms
14620	3 s 696 ms	1 s 978 ms	4 s 722 ms	548 ms	431 ms	13 s 468 ms
16380	4 s 134 ms	2 s 194 ms	5 s 324 ms	585 ms	481 ms	15 s 30 ms
18240	4 s 617 ms	2 s 433 ms	5 s 789 ms	618 ms	554 ms	16 s 600 ms
20200	5 s 107 ms	2 s 668 ms	6 s 296 ms	642 ms	567 ms	18 s 106 ms

Tab. 6.1 – Temps de calcul obtenus pour le co-raffinement entre deux sphères.

Nous pouvons aussi nous rendre compte que les temps de calcul correspondant à la création des lignes de coupe sont environ dix fois inférieur à ceux de la recherche des points de départ de ces lignes. De plus, sur la courbe montrant uniquement l'évolution du temps de calcul des lignes de coupe, nous pouvons observer le même phénomène que pour les résultats obtenus avec l'algorithme de co-raffinement 2D (cf. section 3.3). En effet, l'allure ce cette courbe est assimilable à celle d'une fonction racine carrée.

6.5 Conclusion

Dans ce chapitre, nous avons présenté un algorithme de co-raffinement 3D basé sur l'algorithme de propagation 2D détaillé au chapitre 3. Cet algorithme permet de générer les intersections existantes entre deux objets en suivant les lignes de coupe communes à ces derniers. Nous avons vu dans les sections 6.3 et 6.4 que cet algorithme pose de nombreux problèmes et qu'il ne permet pas de gérer tous les cas possibles.

Cependant, le principe de propagation utilisé dans cet algorithme reste intéressant pour certaines applications spécifiques. Nous verrons dans la section 10.2.4.3 qu'il peut être utile dans le domaine de la géologie.

Chapitre 7

Co-raffinement 3D par intersection de couples de faces

Pour résoudre les différents problèmes rencontrés dans le précédent algorithme de co-raffinement 3D, nous avons décidé de réaliser un nouvel algorithme dont les traitements effectués sur les faces sont globaux. Ainsi, la procédure de base n'est plus la création d'un segment de coupe local mais consiste en la création de tous les segments de coupe existants entre deux faces. Ceci nous permet alors de résoudre le problème des faces coplanaires ainsi que celui des intersections non détectées (arêtes du premier objet n'étant pas en intersection avec les cellules du second). De plus, les segments de coupes existants entre deux faces étant tous traités en même temps, il nous est possible de déterminer si deux faces ont déjà été intersectées. Ce dernier point permet de garantir l'unicité des intersections calculées.

Nous présentons dans un premier temps dans les sections 7.1 et 7.2 le principe de fonctionnement de cet algorithme ainsi que les hypothèses que nous utilisons. Nous décrivons ensuite en section 7.3 la manière dont il est découpé en expliquant chaque partie en détail. Nous continuons en donnant la description complète de l'algorithme en section 7.4 et la manière dont il a été implémenté sur le modèle des 3-G-Cartes. Nous finissons enfin par présenter quelques résultats dans la section 7.5 et nous concluons en section 7.6.

7.1 Principe

Comme nous l'avons vu précédemment, l'intersection entre deux objets s'effectue en recherchant les segments de coupe existants entre eux. Ces segments de coupe proviennent principalement de l'intersection entre les faces du premier objet et les faces du second. Le principe que nous utilisons ici consiste tout simplement à calculer les intersections existantes entre chaque face F_1 du premier objet O_1 et chaque face F_2 du second objet O_2.

De cette manière, lorsque nous voulons traiter un couple de faces (F_1, F_2), nous regardons tout d'abord si les faces possèdent une intersection et si oui quel est le type de celle-ci. Nous pouvons alors être confrontés soit à des faces coplanaires soit à des faces ayant une intersection

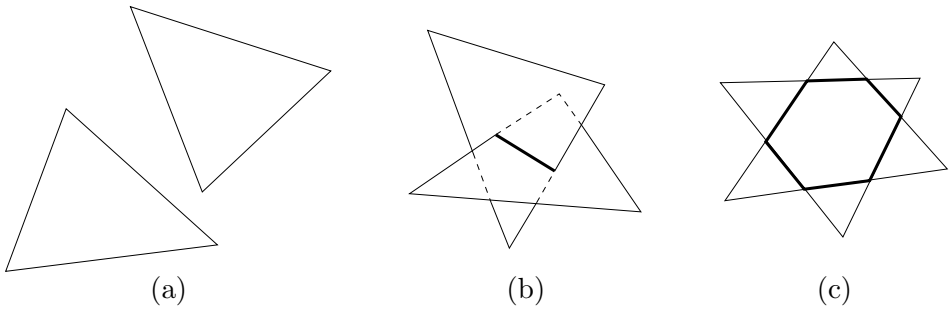


FIG. 7.1 – Exemples de faces : (a) faces non sécantes ; (b) faces sécantes s'intersectant en un segment ; (c) faces coplanaires.

le long d'une ligne de coupe (cf. FIG. 7.1). Nous effectuons donc le traitement approprié à chaque cas et nous obtenons un ensemble de segments résultant de l'intersection des faces.

Comme pour le précédent algorithme, nous devons rendre la subdivision de l'espace valide à la suite du calcul des intersections. Pour cela, nous trions angulairement les faces incidentes à chaque segment de coupe et nous mettons à jour la topologie en conséquence.

Lorsque nous découpons une face par une autre, qu'il s'agisse de faces coplanaires ou non, nous pouvons obtenir une ou plusieurs « faces filles ». Nous devons alors nous souvenir que l'ensemble des faces filles obtenues proviennent de l'intersection de deux « faces mères » et qu'elles ne devront pas être intersectées de nouveau entre elles. Pour cela, nous utilisons une structure de données et un algorithme adapté à ce genre de traitement.

7.2 Hypothèses

Pour modéliser nos objets, nous avons décidé d'utiliser des 3-G-Cartes. Nous supposons que ces objets sont des maillages orientables ne possédant aucun 3-bord. Ceux-ci peuvent cependant être composés de un ou plusieurs volumes qui peuvent être fermés ou non fermés (*i.e.* volumes possédant des 2-bords) mais dont toutes les faces sont fermées (aucun 0-bord ni 1-bord).

Nous supposons de plus que la topologie des objets est correcte et ne comporte ni arêtes doubles (*i.e.* faces fermées composées de seulement deux arêtes) ni faces doubles (*i.e.* volumes fermés composés de seulement deux faces).

En ce qui concerne la géométrie, nous travaillons ici avec des objets plongés linéairement dont seuls les sommets possèdent des informations géométriques, à savoir des coordonnées cartésiennes. Nous considérons que le bord des faces des objets peut être concave mais que tous leurs sommets doivent se trouver sur un même plan. Nous supposons aussi que les objets ne possèdent aucun défaut géométrique comme par exemple des auto-intersections ou des arêtes de longueur nulle.

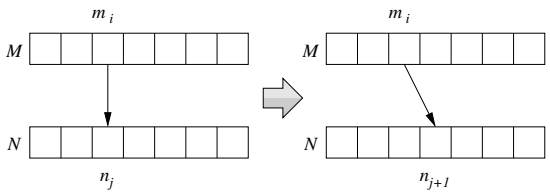


FIG. 7.2 – Fonctionnement de la boucle principale dans le cas où aucune intersection n'est détectée : passage au couple de faces suivant.

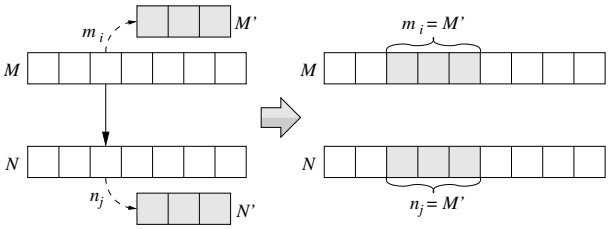


FIG. 7.3 – Fonctionnement de la boucle principale dans le cas où une intersection est calculée : remplacement des éléments intersectés par la liste de leur faces filles.

7.3 Raisonnement

En suivant le principe énoncé précédemment, nous pouvons découper l'algorithme en plusieurs étapes qui sont les suivantes :

1. il y a tout d'abord la boucle principale qui se charge de tester les faces du premier objet avec celle du second de manière à ne calculer qu'une seule fois les intersections ;
2. pour chaque couple de faces, nous testons si celles-ci possèdent une intersection et quel est le type de cette dernière ;
3. en fonction du type d'intersection détecté, nous utilisons le traitement adapté à la réalisation de cette dernière ;
4. nous mettons enfin à jour la topologie autour des arêtes d'intersection afin de rendre la subdivision résultante valide.

Nous expliquons ici chacune de ces parties en détail en commençant par la boucle principale.

7.3.1 Boucle principale

Afin de pouvoir contrôler globalement l'enchaînement des intersections, nous utilisons des listes pour stocker les faces de nos objets. Nous stockons alors les faces de l'objet O_1 dans une liste M et les faces de l'objet O_2 dans une liste N. Pour réaliser toutes les intersections, nous testons alors chaque élément de M avec chaque élément de N. Chaque couple d'élément n'est testé qu'une seule fois et nous obtenons ainsi un algorithme dont la complexité est $\mathcal{O}(m \times n)$, m et n représentant le nombre de faces dans chaque liste.

À chaque étape du parcours des listes, nous traitons un élément m_i de la liste M avec un élément n_j de la liste N. Lors de ce traitement, nous pouvons être confrontés à deux cas de figure. Soit les éléments m_i et n_j ne génèrent pas d'intersection et nous pouvons alors continuer la boucle en passant au traitement des éléments m_i et n_{j+1} (cf. FIG. 7.2). Soit les éléments possèdent une intersection et il est alors possible que les faces correspondant aux éléments m_i et n_j soit découpées en plusieurs morceaux, générant ainsi de nouvelles faces (cf. FIG. 7.4). Dans ce dernier cas, nous devons mettre à jour les listes M et N afin de remplacer les faces intersectées

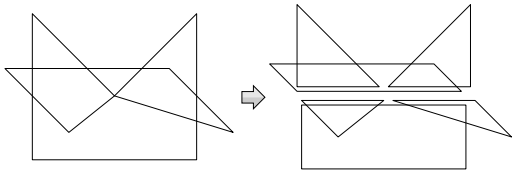


FIG. 7.4 – Exemple d'intersection découpant les deux faces originales en un ensemble de plusieurs faces filles.

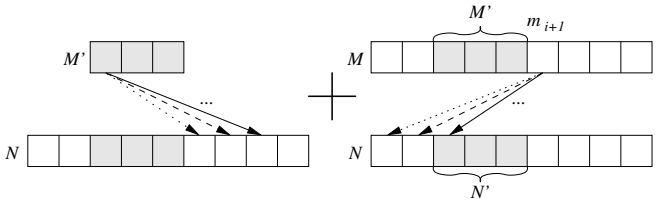


FIG. 7.5 – Parcours résultant d'une modification de la liste suite à la génération d'une intersection entre deux éléments.

par leurs « faces filles » respectives. Nous notons alors M' et N' les listes contenant les faces filles des éléments m_i et n_j (cf. FIG. 7.3). Dans les deux cas, lorsque tous les éléments de N sont testés avec un élément m_i de M, nous passons à l'élément m_{i+1} et le traitons à son tour avec tous les éléments de N.

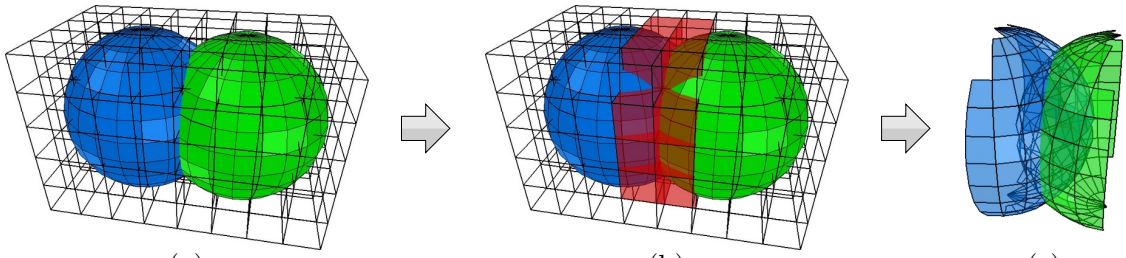
Lorsque nous modifions les listes M et N suite à une intersection entre un élément m_i et un élément n_j, nous devons continuer le parcours en assurant de ne pas recalculer des intersections entre les faces filles générées. Pour cela, nous devons découper notre parcours en deux parties. L'élément m_i pouvant aussi posséder des intersections avec les éléments de N étant situés après n_j, nous traitons dans un premier parcours les éléments de M' (*i.e.* faces filles de m_i) avec les éléments n_k, $k > j$. Puis, dans un second parcours, nous continuons le parcours initial en passant à l'élément suivant de la liste M originale (*i.e.* m_{i+1}) et le traitons avec tous les éléments de N, N' y compris (cf. FIG. 7.5).

À l'aide de cette méthode, nous pouvons assurer que les faces ne seront jamais traitées entre elles plus d'une fois et qu'il en est de même pour leurs faces filles. De ce fait nous assurons que l'algorithme converge et que malgré la croissance des listes de faces, la complexité reste la même. Cependant, comme nous l'avons évoqué précédemment, la complexité d'un tel algorithme est $\mathcal{O}(n^2)$, ce qui peut être gênant pour des objets possédant un très grand nombre de faces. Nous proposons alors une méthode permettant d'accélérer ce parcours.

Dans la plupart des cas d'intersection entre objets, nous pouvons remarquer que seule une infime partie des faces sont modifiées. En effet, comme nous l'avons vu dans le précédent algorithme (cf. chapitre 6), les intersections se produisent le long de lignes de coupe. Ainsi, seules les faces incidentes à ces lignes de coupe sont modifiées lors du traitement. Il est alors inutile de tester toutes les faces d'un objet avec celles de l'autre.

Cependant, nous ne pouvons pas connaître à l'avance où vont se produire exactement les intersections et quelles sont les faces concernées étant donné que c'est le but de l'algorithme actuel. Par contre, comme nous connaissons la géométrie des objets, nous connaissons la répartition de leurs faces dans l'espace. Nous pouvons alors estimer de manière grossière quelles sont les parties de l'espace qui vont être communes aux deux objets.

Pour ce faire, nous discrétisons l'espace à l'aide d'une grille régulière couvrant l'intégralité des deux objets (cf. FIG. 7.6(a)). Ensuite, nous répartissons les faces des objets dans les cellules de la grille en testant simplement si la boîte englobante d'une face chevauche une cellule. Lorsque toutes les faces sont assignées à une ou plusieurs cellules et réciproquement, nous obtenons trois types de répartitions pour chaque cellule. Soit la cellule ne contient aucune face, soit elle



(a) (b) (c)

FIG. 7.6 – Exemple d'utilisation d'une grille régulière pour l'optimisation de la recherche des intersections : (a) création d'une grille régulière couvrant les deux objets à considérer ; (b) détection des cellules de la grille contenant au moins une face de chaque objet (cellules coloriées) ; (c) suppression des faces dont la boîte englobante ne chevauche pas l'une des cases coloriées en (b).

contient des faces d'un seul objet, soit elle contient des faces des deux objets (cellules colorées sur la figure 7.6(b)). Nous pouvons alors affirmer que si une face n'est pas associée à une cellule contenant les faces des deux objets, cette face ne peut en aucun cas posséder une intersection avec une quelconque autre face. Dans ce cas, nous pouvons supprimer des listes M et N toutes les faces vérifiant cette condition (cf. FIG. 7.6(c)).

Cette technique permet en général d'éliminer la majeure parties des faces inutiles, ce qui accélère grandement le calcul. Les résultats obtenus en fonction de la résolution de la grille sont donnés dans la section 7.5 à la page 126.

Nous allons maintenant voir comment détecter si deux éléments possèdent une intersection ainsi que son type si tel est le cas. Ensuite, nous expliquons la technique utilisée pour générer les différents types d'intersections ainsi que la façon dont nous mettons à jour la topologie finale.

7.3.2 Détection d'une intersection

Pour détecter si deux faces sont sécantes ainsi que le type de l'intersection, nous passons par plusieurs traitements successifs. Dans un premier temps, nous testons si les boîtes englobantes des faces se chevauchent. Si ce n'est pas le cas, nous pouvons affirmer que les faces ne possèdent aucune intersection. Dans le cas contraire, nous devons effectuer d'autres tests.

Lorsque nous calculons l'intersection entre deux faces, nous sommes souvent amenés à modifier la topologie des bords des faces en y insérant des sommets. Ces sommets proviennent de l'intersection entre les segments formant le bord d'une face et le plan de l'autre face (cf. FIG. 7.7). Pour détecter plus précisément si deux faces possèdent une intersection, nous devons alors savoir si de tels points d'intersection existent, c'est-à-dire si les segments du bord d'une face possèdent des intersections avec le plan de l'autre face.

Un segment privé de ses extrémités et un plan possèdent un point d'intersection si et seulement si les deux extrémités du segment ne se trouvent pas du même côté du plan. Si nous considérons deux faces F_1 et F_2, nous pouvons alors utiliser cette propriété en polarisant le plan P_2 de F_2 et en testant la position des sommets de F_1 par rapport à P_2. Ce test correspond au calcul de la distance relative D entre les points et l'équation du plan :

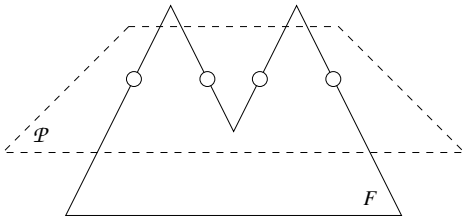


FIG. 7.7 – Exemple d'intersection entre une face F et un plan $\mathcal{P}$. Les cercles vides représentent les points d'intersections entre le bord de F et le plan $\mathcal{P}$.

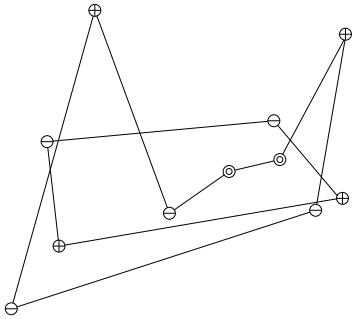


FIG. 7.8 – Classement des sommets de chaque face par rapport au plan de l'autre face. Le signe $\oplus$ (resp. $\ominus$ et $\circledcirc$) désigne les sommets d'une face se trouvant du côté positif du plan de l'autre face (resp. du côté négatif et sur le plan).

DÉF. 5 – Soit $\mathcal{P}$ un plan d'équation $a.x + b.y + c.z + d = 0$ et p un point de coordonnées (p_x, p_y, p_z). La distance relative D entre p et $\mathcal{P}$ est donnée par la formule :

$$D = \frac{a \times p_x + b \times p_y + c \times p_z + d}{\sqrt{a^2 + b^2 + c^2}}$$

La distance obtenue est dite « relative » car elle peut être soit positive, soit négative. Nous pouvons alors connaître de quel côté du plan se trouvent les points simplement grâce à son signe. La valeur absolue de cette distance correspond à la plus courte distance euclidienne entre le point et le plan (*i.e.* distance euclidienne entre un point et son projeté orthogonal sur le plan). Ceci nous permet donc de déterminer si le point est suffisamment proche du plan pour considérer qu'il se trouve dessus. Nous fixons une valeur $\mathcal{E}$ nous indiquant quels sont les points considérés comme étant confondus au plan.

Nous effectuons alors un classement des sommets de chaque face par rapport au plan de l'autre face. Nous obtenons ainsi pour chaque face trois catégories de points (cf. FIG. 7.8) : les points positifs ($D > \mathcal{E}$), les points négatifs ($D < -\mathcal{E}$) et les autres ($-\mathcal{E} \leq D \leq \mathcal{E}$). À l'aide de ce classement, nous pouvons déterminer à la fois si les faces possèdent une intersection et le type de celle-ci :

- Si les sommets d'une des faces sont soit tous positifs, soit tous négatifs, ceci indique que tous les sommets de cette face se trouvent du même côté du plan de l'autre face. Par conséquent, les deux faces ne possèdent aucune intersection.
- Si une des deux faces ne possède aucun sommet positif ni aucun sommet négatif, cela signifie que tous les sommets de cette face se trouve sur le plan de l'autre face. Ainsi, nous pouvons considérer que les faces sont coplanaires.
- Dans tous les autres cas il est possible que les faces possèdent des parties communes correspondant à des intersections devant être calculées. Cependant, ce test ne permet pas d'assurer qu'il existe réellement des intersections entre les deux faces (cf. section 7.3.3).

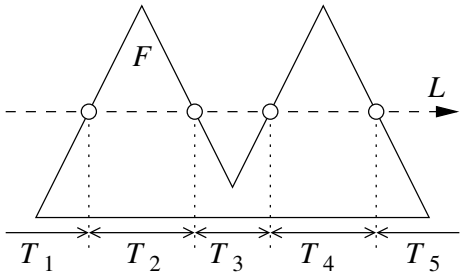


FIG. 7.9 – Exemple de tronçons obtenus sur une ligne de coupe L lorsque celle-ci est intersectée par le bord d'une face F.

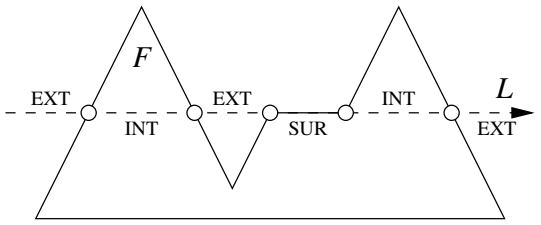


FIG. 7.10 – Les différentes positions des tronçons d'une ligne de coupe par rapport au bord d'une face : EXT pour extérieur, INT pour intérieur et SUR pour sur le bord.

Notons tout de même que dans les tests précédents, deux faces sont considérées coplanaires même si seule une des deux faces vérifie la propriété énoncée. En effet, nous faisons ce choix car par exemple, lors d'une intersection entre une petite face et une grande face, il est tout à fait probable que tous les sommets de la petite face se trouve sur le plan de la grande mais que la réciproque soit fausse à cause des erreurs numériques. Or, dans ce cas, nous devons tout de même considérer que les faces sont coplanaires car une intersection franche pourrait poser des problèmes numériques et par conséquent topologiques.

Les tests cités précédemment nous permettent de connaître dans quel cas de figure nous sommes. Nous pouvons alors appeler les sous programmes associés à chacun de ces cas. Nous avons déjà vu précédemment les traitements à effectuer dans le cas où il n'y a aucune intersection (cf. section 7.3.1). Nous allons donc maintenant voir quels sont ceux correspondants aux intersections entre deux faces sécantes puis entre deux faces coplanaires.

7.3.3 Intersection de faces sécantes

L'intersection entre deux plans non coplanaires est modélisée sous la forme d'une droite que nous appelons « ligne de coupe ». Le vecteur directeur de cette droite correspond simplement au produit vectoriel des vecteurs normaux aux deux plans.

Comme nous travaillons ici avec des faces dont le bord est fermé, cette ligne de coupe peut posséder une ou plusieurs intersections avec les bords des faces considérées et peut ainsi être découpée en plusieurs « tronçons » (cf. FIG. 7.9). Notons que les points d'intersection obtenus entre la ligne de coupe et les bords des faces correspondent aux points d'intersection entre le bord d'une face et le plan de l'autre face. De plus comme nous l'avons évoqué précédemment, seuls les segments ayant une extrémité positive et l'autre négative peuvent être intersectés. Il est donc inutile de procéder à un calcul d'intersection avec les segments ne vérifiant pas cette condition. Cependant, il est aussi possible que la ligne de coupe traverse les faces en passant par des sommets déjà existants. Dans ce cas, ces sommets doivent être considérés au même titre que les autres points d'intersection.

Tous les tronçons de la ligne de coupe obtenus par intersection avec les bords des faces peuvent correspondre à des segments de coupe. Si nous considérons l'intersection entre la ligne

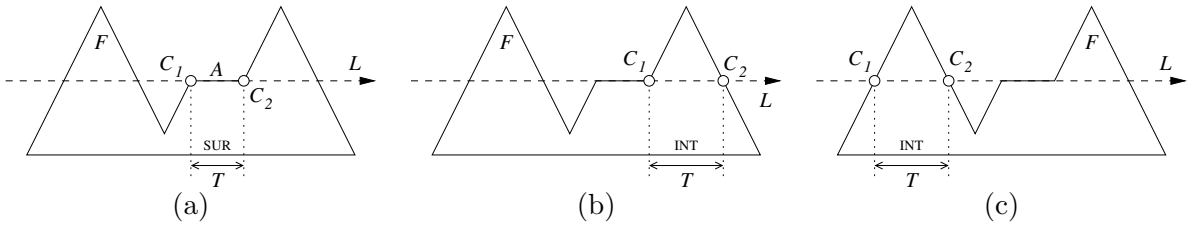


FIG. 7.11 – Les différents cas possible lors de la détection de position d'un tronçon T d'une ligne de coupe L par rapport au bord d'une face F.

de coupe et une seule face, nous pouvons observer que seule une partie des tronçons sont situés à l'intérieur ou sur les limites de la face (cf. FIG. 7.10). Ces tronçons définissent alors des segments de coupe qui ne sont valables que pour l'intersection entre une face et un plan. Si nous calculons l'intersection entre la ligne de coupe et le bord de deux faces, le partitionnement est plus complexe et seuls les tronçons se trouvant à l'intérieur ou sur le bord des deux faces définissent des segments de coupe.

Ainsi, pour déterminer les segments de coupe existants entre deux faces, nous devons procéder en trois étapes :

1. nous devons tout d'abord calculer la ligne de coupe ;
2. nous devons ensuite calculer l'intersection entre la ligne de coupe et le bord de chaque face[1] et trier les points d'intersection selon le vecteur directeur de la ligne de coupe ;
3. nous devons reconnaître quels sont les tronçons de la ligne de coupe communs aux deux faces.

Pour déterminer la position d'un tronçon d'une ligne de coupe L par rapport à une face F, nous considérons tout d'abord un tronçon T ayant pour extrémités les points P_1 et P_2. Ces points proviennent respectivement des intersections entre la ligne de coupe et des cellules C_1 et C_2 de la face. Nous considérons de plus que P_1 se trouve avant P_2 selon la direction de la ligne de coupe. Nous pouvons alors distinguer les différents cas suivants :

- si C_1 et C_2 sont des sommets et qu'ils sont reliés par une arête A, alors T est confondu à A, *i.e.* T se trouve sur le bord de F (cf. FIG. 7.11(a)) ;
- si C_1 est un sommet et que T se trouve dans un secteur angulaire incident à C_1 qui désigne l'intérieur de la face[2], alors T se trouve à l'intérieur de F (cf. FIG. 7.11(b)) ;
- si C_1 est une arête et que T se trouve du côté désignant l'intérieur de la face[2], alors T se trouve dans F (cf. FIG. 7.11(c)) ;
- dans les autres cas, T se trouve hors de la face.

Lorsque les positions des tronçons ont été détectées sur chacune des faces, il suffit de les mettre en commun afin de déterminer les segments de coupe (tronçons se trouvant à l'intérieur

[1] Cette intersection est opérée en calculant les points d'intersection entre le bord d'une face et le plan de l'autre face.

[2] Ces opérations dépendant de la structure de données utilisée, elles sont détaillées dans la partie « description de l'algorithme » en section 7.4

ou sur le bord des deux faces). Lorsque ceux-ci sont désignés, nous devons ensuite les insérer sur chacune des deux faces en modifiant leur topologie. Pour ce faire, nous traitons les tronçons de la ligne de coupe en suivant la direction de cette dernière.

L'insertion d'une arête dans une face s'effectue à l'aide de trois opérations de base qui sont : l'insertion d'un sommet sur une arête, l'insertion d'un sommet dans une face et l'éclatement d'une face entre deux sommets. Ces opérations ne sont pas toujours toutes utilisées, cela dépendant des cas d'intersection rencontrés.

Dans les sections suivantes, nous commençons par présenter la méthode utilisée pour représenter l'inclusion d'éléments dans une face. Nous expliquons ensuite comment insérer un segment de coupe sur une face en découpant le processus en deux étapes : l'insertion de ses extrémités notées E_1 et E_2, et l'insertion de l'arête en elle même notée A. Dans la suite, nous notons T le tronçon de la ligne de coupe traité et P_1 et P_2 ses extrémités. De plus, l'insertion que nous détaillons concerne seulement une arête pour une seule face notée F.

7.3.3.1 Inclusion d'éléments dans une face

Lorsque nous réalisons des intersections, nous sommes souvent amenés à insérer des segments en plein milieu d'une face sans aucune liaison avec le bord de celle-ci (cf. Fig. 7.12(a)). Cependant, nous devons nous souvenir à quelles faces appartiennent ces segments et par conséquent, la structure de données utilisée doit permettre de stocker cette information. Il existe deux manières de stocker cette information avec chacune leurs avantages et leurs inconvénients.

La première méthode consiste à stocker dans chaque face une liste décrivant les segments ou faces qu'elle contient. Ainsi, lorsque nous voulons rajouter un segment sur une face, nous l'ajoutons simplement à la liste. Cependant, lorsque la face est découpée en plusieurs parties suite à une intersection, nous devons aussi découper la liste en conséquence afin que chaque segment se retrouve dans la bonne face fille. Ceci implique donc d'effectuer des tests d'appartenance de segments à une face et comme le bord des faces avec lesquelles nous travaillons n'est pas toujours convexe, ce test peut être couteux en temps de calcul (cf. test d'appartenance d'un point à une face expliqué en section 6.2.1).

La deuxième solution consiste à représenter explicitement la relation d'appartenance d'un segment à une face en reliant celui-ci au bord de la face à l'aide d'une « arête fictive ». Cette arête fictive est ensuite considérée comme n'importe quelle autre arête et peut aussi être intersectée. Ainsi, lorsqu'une face est découpée en plusieurs morceaux, les arêtes fictives peuvent aussi être découpées et la nouvelle relation d'appartenance découle directement de la nouvelle topologie calculée (cf. Fig. 7.13). Notons d'ailleurs, que lorsque une arête fictive est découpée, une des deux parties résultantes doit être supprimée. En effet, par définition, une arête fictive ne doit pas découper une face en deux et chaque côté d'une telle arête doit désigner une même face (cf. Fig. 7.13(b)). Cependant, bien que la topologie soit simple à mettre à jour lors d'un calcul d'intersection, l'ajout d'une arête fictive dans une face est plus compliqué. Lorsque nous insérons une nouvelle arête fictive, nous devons nous assurer que cette dernière définisse une topologie et une géométrie correcte afin de ne pas obtenir d'auto-intersections avec le bord de la face (cf. Fig. 7.12(b)).

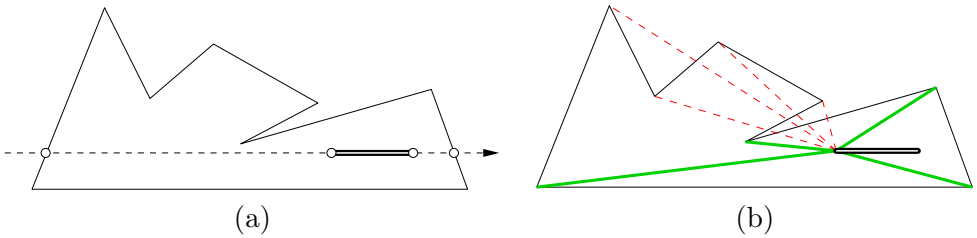


FIG. 7.12 – Inclusion d'un segment au sein d'une face à l'aide d'une arête fictive : (a) la ligne en pointillés représente la ligne de coupe, les cercles représentent des points d'intersection et le segment double correspond à un segment de coupe devant être ajouté sur la face ; (b) les lignes en pointillés définissent des arêtes fictives géométriquement non valides et les lignes en gras aux arêtes fictives valides.

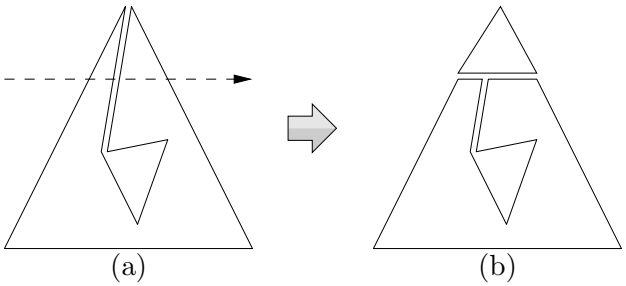


FIG. 7.13 – Exemple d'arête fictive découpée lors d'un calcul d'intersection : (a) la ligne en pointillés correspond à la ligne de coupe et la ligne double à une arête fictive ; (b) topologie résultant du calcul d'intersection (la partie haute de l'arête fictive à été supprimée).

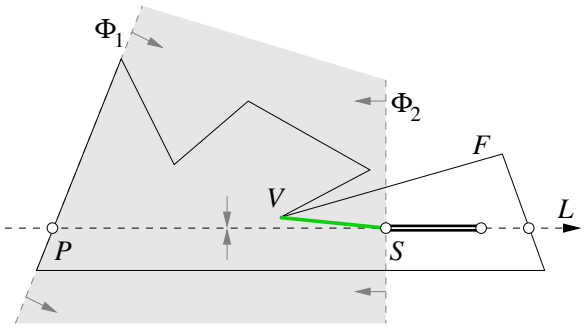


FIG. 7.14 – Exemple d'insertion d'une arête fictive dans une face. La ligne en pointillés correspond à la ligne de coupe et le segment double au segment de coupe devant être inséré dans la face. La partie grisée montre la zone considérée pour la recherche du sommet le plus proche de la ligne de coupe et le segment en gras correspond à l'arête fictive trouvée par la méthode.

Pour gérer le problème d'inclusion, nous avons décidé d'utiliser la deuxième méthode qui consiste à ajouter des arêtes fictives. Nous avons alors défini une méthode permettant d'insérer une arête fictive A dans une face F en s'assurant que A ne possède aucune intersection avec le bord de F. Pour cela, nous n'effectuons aucun test d'intersection entre les arêtes potentielles et le bord de la face, mais nous utilisons les propriétés liées au calcul d'intersection en cours. En effet, comme l'insertion d'arêtes fictives se fait généralement lors d'un calcul d'intersection, nous utilisons ceci afin de déterminer une zone de la face dans laquelle nous sommes sûrs de ne pas recouper son bord.

Lorsque nous voulons ajouter un sommet S au milieu d'une face F, nous savons que S se trouve sur une ligne de coupe L. De plus, nous savons que L possède au moins un autre point d'intersection P avec le bord de F correspondant à l'entrée dans la face, que P se trouve juste avant S sur L et qu'il n'y aucun autre point d'intersection entre P et S (cf. FIG. 7.14). Ceci signifie donc qu'il est en théorie possible d'ajouter une arête entre S et P sans obtenir aucune intersection avec le bord de F. Ceci n'est pourtant pas souhaitable car cela entraînerait l'ajout d'un sommet topologique au niveau de P, ce qui compliquerait nettement les objets d'origine et pourrait provoquer des erreurs numériques.

Cependant, nous pouvons déduire de cette propriété qu'il existe une zone dont les bords sont parallèles à L, qui est comprise entre P et S et dans laquelle il n'existe aucune arête de F. Cette zone peut d'ailleurs être bornée par les sommets de F les plus proches de L. Ainsi, ces sommets peuvent être utilisés pour l'insertion d'une arête fictive reliant le sommet S.

Pour déterminer les points valides, nous commençons alors par définir la zone de recherche à l'aide de deux inéquations. La première inéquation notée Φ_1 passe par le point P et est parallèle à l'arête de F sur laquelle se trouve P. Le côté positif de Φ_1 est déterminé par la direction de la ligne de coupe L (*i.e.* sa normale est orientée dans le même sens que L). La deuxième inéquation notée Φ_2 passe par le sommet S, est perpendiculaire à L et possède la même orientation que Φ_1. Ainsi, pour trouver un sommet convenant à l'insertion d'une arête fictive, nous recherchons un sommet valide V de la face F tel que $\Phi_1 \leq V \leq \Phi_2$ et que la distance entre V et L soit la plus petite possible (cf. FIG. 7.14).

7.3.3.2 Insertion des extrémités d'une arête

Pour insérer les extrémités d'une arête sur une face, nous devons séparer nos traitements en deux cas : soit T se trouve à l'intérieur de F, soit T est confondu avec un segment du bord de F.

Dans le premier cas, nous considérons que la ligne de coupe rentre dans la face F par une cellule C_1 et en sort via une cellule C_2. Nous obtenons alors deux cas possibles :

- Soit P_1 (resp. P_2) provient de l'intersection entre la ligne de coupe et C_1 (resp. C_2). Dans ce cas, nous éclatons C_1 (resp. C_2) en P_1 (resp. P_2) s'il s'agit d'une arête et nous obtenons ainsi l'extrémité E_1 (resp. E_2). S'il s'agit d'un sommet, nous récupérons ce dernier.
- Soit P_1 (resp. P_2) ne provient pas de l'intersection entre la ligne de coupe et C_1 (resp. C_2). Dans ce cas, cela signifie que le point P_1 (resp. P_2) provient de l'intersection avec une cellule de l'autre face. Comme nous n'avons ici aucun élément de F auquel raccrocher notre extrémité, nous devons insérer un sommet au milieu de la face. Ceci se fait grâce à l'ajout d'une arête fictive (cf. section 7.3.3.1) et E_1 (resp. E_2) correspond à son extrémité.

Dans le second cas, la ligne de coupe longe une arête A de la face. Nous notons alors S_1 et S_2 les sommets de cette arête. Si P_1 (resp. P_2) n'est pas situé entre S_1 et S_2, nous assignons S_1 (resp. S_2) à E_1 (resp. E_2). Dans le cas contraire, nous insérons sur A un sommet correspondant à E_1 (resp. E_2) ayant pour coordonnées P_1 (resp. P_2).

7.3.3.3 Insertion d'une arête entre deux sommets

Lorsque nous avons inséré les extrémités des segments de coupe sur les faces, il ne reste plus qu'à insérer une arête. Cependant, deux cas peuvent ici se présenter. Si le tronçon T considéré est confondu avec une arête de la face, il existe alors déjà une arête reliant les sommets E_1 et E_2 et il est donc inutile d'en rajouter une nouvelle. Dans le cas contraire, nous insérons une arête entre les sommets E_1 et E_2 qui peut dans la plupart des cas diviser la face en deux.

Le principe d'insertion d'un segment de coupe que nous avons présenté ici est en réalité plus optimisé afin de limiter l'insertion d'arêtes fictives. En effet, lorsque nous ajoutons un segment au milieu d'une face, chacune de ses extrémités ne doit pas être reliée par une arête fictive pour deux raisons : 1) une seule arête fictive suffit à inclure le segment dans la face ; 2) une deuxième arête fictive découperait la face en deux, ce qui n'est par définition pas autorisé.

Pour ce faire, nous insérons d'abord le premier sommet E_1 à l'aide d'une arête fictive puis directement le segment de coupe en le reliant à E_1. L'autre extrémité du segment de coupe sera alors E_2.

Il en est de même lorsque nous devons insérer une arête « pendante » (*i.e.* une arête dont un sommet touche le bord de la face et dont l'autre est à l'intérieur de la face). Comme celle-ci est déjà reliée au bord de la face, nous n'ajoutons pas d'arête fictive pour relier son autre extrémité.

7.3.4 Intersection de faces coplanaires

L'intersection entre deux faces coplanaires correspond à un co-raffinement 2D entre les bords des faces. Cependant, nous ne pouvons pas utiliser directement l'algorithme étudié dans le

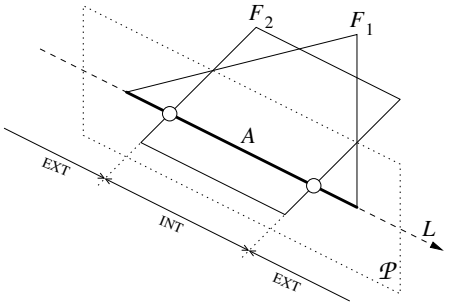


FIG. 7.15 – Exemple de recherche de points d'intersection entre les bords de deux faces coplanaires F_1 et F_2. Le plan $\mathcal{P}$ est colinéaire à l'arête A et perpendiculaire aux faces F_1 et F_2. Il génère une ligne de coupe L colinéaire à A ainsi que deux points d'intersection avec le bord de F_2. La ligne de coupe L est alors découpée en trois tronçons.

chapitre 3 car celui-ci fusionne les maillages originaux afin d'obtenir une nouvelle subdivision 2D.

Dans notre cas, nous voulons que l'algorithme général fonctionne en deux étapes principales (cf. section 7.1) qui sont la création des segments de coupe et la mise à jour de la topologie (liaison des objets entre eux). Nous ne pouvons donc pas fusionner les faces à cette étape et nous devons conserver l'intégralité des faces originales. Pour cela, nous devons seulement rajouter sur chaque face des segments permettant de les coudre entre elles lors de la deuxième phase de l'algorithme.

Comme la méthode de co-raffinement 2D présentée au chapitre 3 est plus adaptée au cas de maillages, et que nous voulons traiter ici seulement deux faces, nous utilisons une autre technique. Cette technique se découpe en deux étapes dont la première consiste à calculer les points d'intersection existants entre les arêtes des deux faces. Nous décrivons cette opération dans la section 7.3.4.1.

La deuxième étape consiste à parcourir le bord d'une face F en partant d'un point d'intersection jusqu'à un autre. Ensuite, pour chaque arête A parcourue, nous détectons si A se trouve dans les limites de l'autre face F' et si c'est le cas, nous insérons dans F' une arête A' confondue à A. Cette deuxième opération est expliquée dans la section 7.3.4.2.

7.3.4.1 Calcul des points d'intersection entre les arêtes des deux faces

Pour générer les points d'intersection entre les arêtes de deux faces F_1 et F_2, nous réutilisons la même méthode que celle expliquée précédemment pour le cas des faces sécantes (cf. section 7.3.3). Pour cela, nous considérons chaque arête A de F_1 (resp. F_2) comme un plan $\mathcal{P}$ perpendiculaire à F_2 (resp. F_1) et nous recherchons toutes les intersections entre $\mathcal{P}$ et les arêtes de F_2 (resp. F_1). Nous obtenons alors une ligne de coupe L confondue avec A (cf. FIG. 7.15) et découpée en plusieurs tronçons qui peuvent être soit à l'intérieur, soit à l'extérieur ou sur le bord de F_2 (resp. F_1). Nous pouvons ensuite générer topologiquement sur F_2 (resp. F_1) et sur A tous les points d'intersection de L se trouvant entre ou sur les extrémités de A.

Lorsque toutes les arêtes d'une face ont été traitées et que plusieurs points d'intersection ont été générés, il est inutile de traiter les arêtes de l'autre face car les mêmes intersections seraient

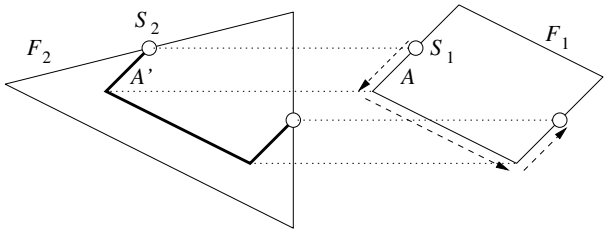


FIG. 7.16 – Exemple de création d'arêtes entre deux points d'intersection. Les deux faces F_1 et F_2 originalement superposées ont été décalées horizontalement afin de représenter les opérations effectuées. Les arêtes en gras sur la face F_2 représentent les segments ajoutés.

détectées. Par contre, si aucun point d'intersection n'a été trouvé, cela signifie que les bords des deux faces ne se chevauchent pas. Nous pouvons alors être confrontés à trois cas possibles : soit F_1 se trouve entièrement dans F_2, soit F_2 se trouve entièrement dans F_1, soit F_1 et F_2 se trouvent l'une à côté de l'autre.

Pour différencier les trois cas précédents, il suffit tout simplement de tester lors de la recherche des points d'intersection entre une arête A de F_1 et le bord de F_2 si les extrémités de A se trouvent sur un même tronçon T de la ligne de coupe qui lui est associée, et si T désigne l'intérieur de F_2. Si toutes les arêtes de la face F_1 vérifient cette condition, alors F_1 se trouve à l'intérieur de F_2. Si ce n'est pas le cas, nous devons alors refaire le traitement en testant les arêtes de F_2 avec le bord de F_1. Si toutes les arêtes de F_2 vérifient la condition précédente, alors F_2 se trouve à l'intérieur de F_1. Dans le cas contraire, les faces ne se chevauchent pas et aucune intersection ne doit être calculée.

Dans les deux premiers cas, nous devons insérer dans la plus grande face F_g une face identique à la face incluse F_i. Cependant, pour que l'étape suivante puisse réaliser cette insertion, les deux faces doivent avoir au moins un point d'intersection en commun. Pour cela, nous ajoutons alors à F_g un sommet confondu à un de ceux de F_i. Cette insertion se fait à l'aide d'une arête fictive en calculant une ligne de coupe entre deux sommets proches de F_g et F_i (cf. section 7.3.3.1).

7.3.4.2 Insertion de segments entre les points d'intersection

Lorsque tous les points d'intersection sont calculés, nous obtenons un découpage des bords des faces en plusieurs tronçons. Chaque tronçon du bord d'une face se trouve entre deux points d'intersection et peut être situé soit à l'intérieur, soit à l'extérieur ou sur le bord de l'autre face. Pour déterminer la position d'un tronçon par rapport à l'autre face, il suffit de connaître la position d'une des arêtes composant ce tronçon. Cette information étant calculée lors de l'étape précédente, il suffit tout simplement de la récupérer.

Pour chacune des faces F_1 et F_2, nous procédons ensuite de la manière suivante. Nous partons d'un point d'intersection I qui est modélisé sous la forme d'un sommet S_1 sur la face F_1 et d'un sommet S_2 pour la face F_2. Nous parcourons chaque tronçon de F_1 (resp. F_2) incident à I se trouvant à l'intérieur de F_2 (resp. F_1) et ce jusqu'à atteindre un autre point d'intersection. Pour chaque arête A de ce tronçon, nous insérons alors une copie A' de A sur la face F_2 (resp. F_1) et nous la relions à S_2 (resp. S_1). Nous notons ensuite S_1 et S_2 les extrémités de A et A', puis

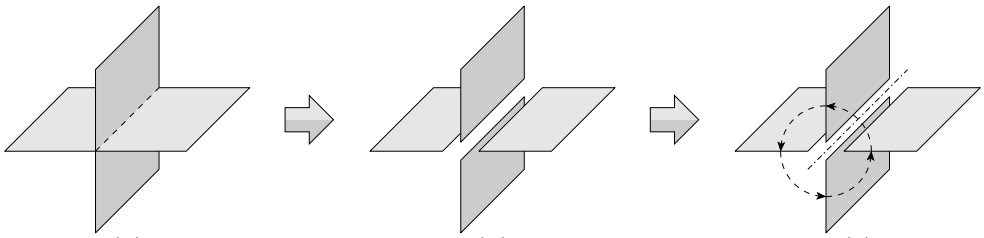


FIG. 7.17 – Mise à jour de la topologie autour des segments d'intersection à l'aide d'un tri angulaire : (a) faces originales avant l'intersection ; (b) face découpées ; (c) tri angulaire autour de l'arête d'intersection.

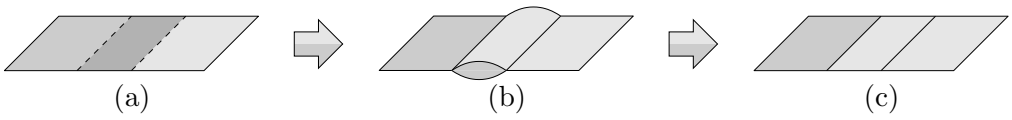


FIG. 7.18 – Correction de la topologie suite à l'intersection de deux faces coplanaires : (a) faces coplanaires ayant une partie commune ; (b) face double apparaissant suite au calcul d'intersection et au tri angulaire ; (c) suppression d'une des deux faces communes.

nous continuons le parcours. Lorsque nous atteignons un nouveau point d'intersection I', nous relions les sommets S_1 et S_2 (*i.e.* les extrémités de la dernière arête parcourue et de sa copie) aux sommets correspondant à I'. Cette opération est représentée sur la figure 7.16 à l'aide d'un exemple simple.

7.3.5 Mise à jour de la topologie

Lorsque tous les segments d'intersection sont générés entre deux objets, il reste encore à rendre la subdivision de l'espace valide. Pour cela, nous corrigeons la topologie autour des arêtes d'intersection en triant angulairement les faces qui leur sont incidentes (cf. FIG. 7.17). Ce processus est identique à celui utilisé dans l'algorithme de co-raffinement 2D (cf. section 3.2.1.3), excepté que les normales aux faces sont utilisées à la place des directions des segments.

Lorsque cette étape est effectuée, nous pouvons nous retrouver avec des artefacts dans la topologie résultante et ce notamment à cause d'éventuelles intersections entre des faces coplanaires. Ces artefacts sont alors représentés sous la forme de faces doubles (ou volumes plats composés de deux faces identiques) et doivent être supprimés (cf. FIG. 7.18). Pour cela, nous repérons les faces communes lors du processus d'intersection de faces coplanaires et nous les fusionnons en supprimant l'une d'entre elles. Nous mettons ensuite à jour la topologie autour des arêtes auxquelles elle était incidente. L'algorithme correspondant est détaillé en section 7.4.8.

7.4 Description de l'algorithme

Nous décrivons dans cette section les principales parties de l'algorithme de co-raffinement 3D et plus particulièrement la manière dont ce dernier est implanté sur le modèle des 3-G-Cartes.

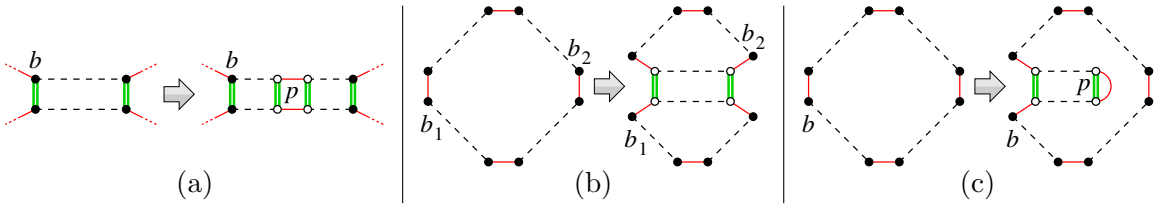


FIG. 7.19 – Les trois principales opérations d'insertion. Les brins blancs correspondent à la cellule nouvellement ajoutée. (a) insertion d'un sommet sur une arête ; (b) découpage d'une face par une arête ; (c) insertion d'une arête pendante dans une face.

Nous commençons en définissant plusieurs opérations topologiques et géométriques utiles par la suite puis nous détaillons les différentes étapes de l'algorithme.

7.4.1 Opérations d'insertion

Nous donnons ici les opérations d'insertion de cellules topologiques dont nous avons besoin dans la description de l'algorithme. Ces opérations concernent l'ajout d'un sommet sur une arête, la scission d'une face par une arête, l'ajout d'une arête pendante dans une face et l'ajout d'un sommet dans une face. Pour chaque opération, nous donnons le prototype de la fonction correspondante ainsi que son fonctionnement :

`ÉclaterArête(`b : `Brin`, p : `Point)` $\rightarrow$ `Brin` Cette fonction permet d'éclater l'arête incidente à b en y ajoutant un sommet topologique plongé en p (cf. FIG. 7.19(a)). Elle retourne le brin du nouveau sommet correspondant à $\alpha_1(\alpha_0(b))$.

`ÉclaterFace(`b_1, b_2 : `Brin)` $\rightarrow$ `Brin` Cette fonction permet d'éclater la face incidente aux brins b_1 et b_2 (cf. FIG. 7.19(b)). Elle insère une arête topologique entre b_1 et b_2 de manière à obtenir $\alpha_2(\alpha_1(b_1)) = \alpha_0(\alpha_1(b_2))$. Le brin retourné correspond à $\alpha_2(\alpha_1(b_1))$.

`InsérerArête(`b : `Brin`, p : `Point)` $\rightarrow$ `Brin` Cette fonction permet d'insérer une arête dans la face incidente au brin b. La nouvelle arête est cousue à b d'un côté et est plongée par p de l'autre (cf. FIG. 7.19(c)). Les brins de l'extrémité plongée sont cousus par α_1 afin de ne pas obtenir de 1-bord dans la face. La fonction retourne le brin de la nouvelle arête correspondant à $\alpha_2(\alpha_1(b))$.

`InsérerSommet(`b : `Brin`, p : `Point)` $\rightarrow$ `Brin` Cette fonction est équivalente à la fonction `InsérerArête` excepté pour deux choses. Le brin retourné appartient au sommet non cousu et correspond à $\alpha_0(\alpha_1(b))$. La nouvelle arête est marquée à l'aide d'une marque Ⓕ permettant d'identifier les arêtes fictives ajoutées par l'algorithme.

7.4.2 Opérations géométriques

Au cours de l'algorithme, nous avons aussi besoin de quelques fonctions géométriques nous permettant principalement de déterminer l'appartenance d'un vecteur à un secteur angulaire. Pour chaque opération, nous donnons le prototype de la fonction, son fonctionnement ainsi que l'algorithme correspondant :

ALGO. 7.1 – Fonction `DansSecteur`

Entrées :
- le vecteur V à tester ;
- deux vecteurs V_1 et V_2 désignant le secteur ;
- le vecteur normal V_n indiquant l'orientation du secteur.

Sorties : `vrai` dans le cas où V est compris entre V_1 et V_2 et `faux` dans le cas contraire.

début
  $a \leftarrow (V_1 \wedge V) \cdot V_n$
  $b \leftarrow (V_2 \wedge V) \cdot V_n$
  si $(V_1 \wedge V_2) \cdot V_n < 0$ **alors retourner** $a \geq 0$ **ou** $b \leq 0$
  sinon retourner $a \geq 0$ **et** $b \leq 0$
fin

`DansSecteur(`V, V_1, V_2, V_n : `Vecteur) → booléen` Cette fonction est similaire à celle utilisée dans l'algorithme 2D (cf. section 3.2.1.3) et consiste à tester si un vecteur V se trouve dans un secteur angulaire modélisé par deux vecteurs V_1 et V_2. Cette fonction prend en paramètre un vecteur V_n supplémentaire permettant d'indiquer la normale du secteur angulaire. Cette fonction est décrite dans l'algorithme 7.1.

ALGO. 7.2 – Fonction `DansSecteur`

Entrées :
- le vecteur V à tester ;
- un brin b désignant un secteur angulaire sur un sommet topologique ;
- le vecteur normal V_n indiquant l'orientation du secteur.

Sorties : `vrai` dans le cas où V se trouve dans le secteur et `faux` dans le cas contraire.

début
  si `SurOrbite(`$\alpha_1(b)$, $<\alpha_2, \alpha_3>(b)$`)` **ou**
    `DansSecteur(`V, `Vecteur(`b`)`, `Vecteur(`$\alpha_1(b)$`))` **alors retourner** `vrai`
  sinon retourner `faux`
fin

`DansSecteur(`V : `Vecteur`, b : `Brin`, V_n : `Vecteur) → booléen` Cette fonction permet d'appeler la méthode précédente en récupérant les vecteurs V_1 et V_2 depuis un secteur angulaire incident à un brin b. De plus, elle permet d'éviter des erreurs liées aux arêtes pendante qui définissent des secteurs vides. Pour cela, elle utilise la topologie pour détecter si le secteur angulaire provient d'une telle arête (cf. ALGO. 7.1).

`TrouverSecteur(`b : `Brin`, P : `Plan`, Ⓜ : `Marque`, v : `Vecteur) → Brin` Cette fonction permet de déterminer le secteur angulaire incident à un sommet (représenté ici par un brin b) qui contient un vecteur v. Pour cela, elle parcours l'orbite du sommet et fait appel aux fonctions précédentes (cf. ALGO. 7.3). Elle prend de plus en paramètre une marque permettant de limiter les brins à tester autour du sommet. Nous expliquons l'intérêt de cette marque en section 7.4.4.

`IntérieurFace(`b : `Brin`, P : `Plan) → Vecteur` Cette fonction diffère des trois précédentes et permet de détermier un vecteur désignant l'intérieur d'une face. Pour cela, elle calcule

ALGO. 7.3 – Fonction `TrouverSecteur`

Entrées :
- un brin b désignant un sommet S d'une face F ;
- le plan P support de la face F ;
- une marque Ⓜ permettant de limiter les brins à tester ;
- le vecteur v à tester.

Sorties : Un brin désignant le secteur angulaire incident à S contenant v. Si aucun existe, le brin vaut `nil`.

```
début
    sect ← nil
    marquer un brin sur deux de l'orbite <α1, α2, α3>(b) avec Ⓣ
    pour chaque brin b_s ∈ <α1, α2, α3>(b) faire
        si EstMarqué(b_s, Ⓣ) alors
            Démarquer((b_s), Ⓣ)
            si sect = nil et EstMarqué(b_s, Ⓜ) et DansSecteur(v, b_s, Normale(P)) alors
                sect ← b_s
    retourner sect
fin
```

ALGO. 7.4 – Fonction `IntérieurFace`

Entrées :
- un brin b désignant une arête A sur une face F ;
- le plan P support de F et dont la normale définit l'orientation de celle-ci.

Sorties : Le vecteur normal à l'arête A sur le plan P qui désigne la direction correspond à l'intérieur de la face.

```
début
    retourner Normale(P) ∧ Vecteur(b)
fin
```

simplement le produit vectoriel entre la normale à la face et le vecteur directeur d'une arête de la face (cf. ALGO. 7.4).

7.4.3 Boucle principale

Pour réaliser la boucle principale de l'algorithme, nous utilisons deux listes notées L_1 et L_2 associées respectivement aux objets O_1 et O_2. Ces listes sont des listes doublement chaînées dont chaque élément contient les informations suivantes :

- `face` : un pointeur sur un brin d'une face. Les arêtes et les sommets de la face correspondante peuvent être retrouvés en parcourant l'orbite $<\alpha_0, \alpha_1>$.
- `plan` : l'équation du plan de la face. Ce plan est dupliqué pour chacune des faces filles provenant de l'éclatement de la face. Ceci permet dans un premier temps d'éviter de recalculer l'équation du plan de la face à chaque fois que sa géométrie est modifiée. Dans un deuxième temps, cela permet aussi de conserver l'équation originale afin de limiter au maximum les erreurs numériques lorsque les faces filles sont trop petites.

– `boîte` : la boîte englobante de la face. Ceci permet de tester rapidement si deux faces ne possèdent pas d'intersection en comparant simplement les coordonnées des sommets des boîtes de chaque face. Cette information est recalculée à chaque fois que la face est découpée en plusieurs faces filles.

Pour représenter les portions de listes devant être mises en commun pour la recherche des intersections, nous utilisons pour chacune d'entre elles deux pointeurs indiquant le début et la fin de la portion. Nous avons donc en tout quatre pointeurs que nous notons deb_1 et fin_1 pour la liste L_1 et deb_2 et fin_2 pour la liste L_2. Les pointeurs de fin fin_i représentent en fait l'élément suivant le dernier élément devant être parcouru, ceci afin de simplifier les conditions d'arrêt des boucles. Au début de l'algorithme, les pointeurs deb_i sont positionnés sur les premiers éléments des listes tandis que les pointeurs fin_i reçoivent la valeur `nil` pour indiquer qu'il s'agit d'un élément n'existant pas.

L'algorithme se compose alors de deux boucles imbriquées. La boucle externe permet de parcourir les éléments de L_1 entre deb_1 et fin_1 et la boucle interne parcourt quant à elle les éléments de L_2 entre deb_2 et fin_2. À chaque intersection réalisée, nous obtenons deux listes L'_1 et L'_2 contenant les faces filles obtenues.

Lorsque des faces filles sont générées, nous stockons dans une file $\mathcal{F}$ les quadruplés de pointeurs permettant d'effectuer les parcours des listes L'_1 et L'_2. Une fois la boucle principale terminée, nous récupérons les premiers quadruplés de $\mathcal{F}$ et nous réitérons le processus jusqu'à ce que $\mathcal{F}$ soit vide.

L'algorithme 7.5 montre l'ensemble des opérations à effectuer. Les explications suivantes se réfèrent aux lignes numérotées dans l'algorithme :

1. nous commençons par initialiser la file $\mathcal{F}$ en y insérant les premières valeurs des pointeurs deb_i et fin_i comme nous l'avons vu précédemment ;
2. nous défilons la file $\mathcal{F}$ jusqu'à ce qu'elle soit vide et affectons à chaque tour les éléments défilés aux pointeurs deb_i et fin_i ;
3. nous démarrons la boucle externe qui parcourt les éléments de la liste L_1 à l'aide d'un pointeur nommé it_1 ;
4. nous démarrons la boucle interne qui parcourt les éléments de la liste L_2 à l'aide d'un pointeur nommé it_2 ;
5. si les boîtes des faces pointées par it_1 et it_2 se chevauchent, nous calculons l'intersection entre ces deux faces à l'aide de la fonction `IntersectionFaces` (cf. section 7.4.4). Nous obtenons alors deux listes L'_1 et L'_2 contenant les faces filles. Si l'intersection entre les deux faces existe et que par conséquent les listes L'_1 et L'_2 ne sont pas vides, nous effectuons les traitements suivants :
   a. nous stockons dans la file $\mathcal{F}$ le quadruplet de pointeurs permettant de parcourir et de calculer les intersections entre les éléments de la liste L'_1 et ceux de L_2 ;
   b. si les listes filles possèdent plus d'un élément, nous mettons à jour (fonction `MàJ`) les listes L_1 et L_2 en remplaçant les faces pointées par it_1 et it_2 par les faces filles contenues dans L'_1 et L'_2 ;

c. nous arrêtons le parcours de la boucle interne et nous passons à l'élément suivant de la boucle externe.

ALGO. 7.5 – Boucle principale

Données : Deux listes L_1 et L_2 contenant les faces des objets O_1 et O_2.
Résultat : Les deux listes L_1 et L_2 mises à jour suite au calcul des intersections entre les faces des objets O_1 et O_2.

```
début
1   F ← Enfiler(∅, (Tête(L1), nil, Tête(L2), nil))
2   tant que F ≠ ∅ faire
        (deb1, fin1, deb2, fin2) ← Défiler(F)
        it1 ← deb1
3       tant que it1 ≠ fin1 faire
            stop ← faux
            it2 ← deb2
4           tant que it2 ≠ fin2 et stop = faux faire
                si boîte(it1) ∩ boîte(it2) ≠ ∅ alors
5                   (L'1, L'2) ← IntersectionFaces(face(it1), face(it2),
                                                   plan(it1), plan(it2))
                    si L'1 ≠ ∅ et L'2 ≠ ∅ alors
a                       deb'1 ← it1 ; fin'1 ← Suivant(L1, it1)
                        deb'2 ← Suivant(L2, it2) ; fin'2 ← fin2
                        si deb'2 ≠ fin'2 alors Enfiler(F, (deb'1, fin'1, deb'2, fin'2))
b                       si Taille(L'1) > 1 alors L1 ← MàJ(L1, it1, L'1)
                        si Taille(L'2) > 1 alors L2 ← MàJ(L2, it2, L'2)
c                       stop ← vrai
                        it1 ← fin'1
                it2 ← Suivant(L2, it2)
            si stop = faux alors it1 ← Suivant(L1, it1)
fin
```

7.4.4 Détection d'une intersection

Avant de pouvoir calculer une intersection entre deux faces, nous devons tout d'abord détecter si cette intersection existe réellement et quelle est son type. Comme nous l'avons vu précédemment (cf. section 7.3.2), cette détection se réalise à l'aide d'un tri des sommets d'une face par rapport au plan de l'autre face.

La fonction `ClasserSommets` décrite dans l'algorithme 7.6 permet d'effectuer ce classement. Elle prend en entrée un brin de la face F dont les sommets doivent être classés, le plan par rapport auquel les sommets doivent être testés, et une marque Ⓜ avec laquelle les brins de la face F sont marqués. Le classement est effectué en marquant les sommets à l'aide de deux marques : $\oplus$ pour les sommets se trouvant du côté positif du plan et $\ominus$ pour ceux se trouvant

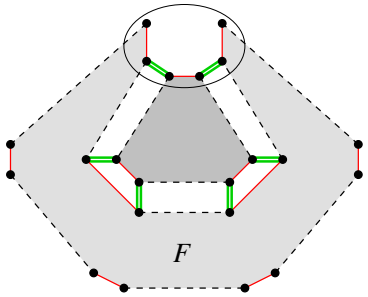


FIG. 7.20 – Exemple de face. Le sommet encerclé est associé à quatre brins de la face F.

du côté négatif. À la fin du classement, la fonction retourne en sortie le nombre total n_s de sommets appartenant à la face, le nombre n_p de sommets se trouvant du côté positif du plan, et le nombre n_n de sommets se trouvant du côté négatif.

La marque Ⓜ est utilisée lors du marquage d'un sommet par ⊕ ou ⊖. Elle permet grâce à l'utilisation de la fonction `MarquerSi` de marquer uniquement les brins d'un sommet qui appartiennent à la face F. En effet, comme les faces que nous manipulons peuvent être de forme quelconque, il est possible qu'un sommet soit associé à plus de deux brins (cf. FIG. 7.20). Dans ce cas, comme nous ne voulons pas traiter plusieurs fois les mêmes sommets lors du parcours de la face, nous marquons l'ensemble des brins de la face appartenant au sommet à classer. Pour se souvenir des sommets déjà traités, nous utilisons de plus une marque notée Ⓣ.

La fonction précédente est ensuite appelée par `IntersectionFaces` (cf. ALGO. 7.7) afin de déterminer le type de l'intersection devant être réalisée. Ce choix est effectué uniquement grâce aux nombres de sommets retournés par la fonction `ClasserSommets`. Les traitements effectués sont les suivants :

1. Nous marquons les brins des faces F_1 et F_2 à l'aide d'une marque Ⓜ. Cette marque sera ensuite passée à toutes les sous-fonctions en ayant besoin.
2. Nous classons les sommets de la face F_1 par rapport au plan P_2 de la face F_2. Nous récupérons ainsi un triplet d'entiers (s_1, p_1, n_1) correpondant respectivement au nombre total de sommets sur F_1, au nombre de sommets de F_1 se trouvant du côté positif de P_2 et au nombre de sommets de F_1 se trouvant du côté négatif de P_2. Nous effectuons le même classement pour la face F_2 et le plan P_1 et nous obtenons de la même manière le triplet (s_2, p_2, n_2).
3. Si l'une des deux faces ne possède aucun sommet se trouvant du côté positif ou négatif (*i.e.* $p_i = n_i = 0$), ceci indique que tous ses sommets sont suffisamment proches du plan de l'autre face pour considérer que les deux faces sont coplanaires. Nous démarquons alors les brins classés par la fonction `ClasserSommets` puis nous exécutons la fonction `IntersectionFacesCoplanaires` dont les traitements sont dédiés à ce cas de figure. Cette fonction retourne l'ensemble ω des arêtes d'intersection générées. Chaque élément de cet ensemble est un couple de brins (b_1, b_2) désignant une même arête d'intersection sur chacune des faces F_1 et F_2.
4. Si les deux faces possèdent des sommets positifs ou négatifs mais pas essentiellement l'un ou l'autre (*i.e.* $p_i \neq 0$ ou $n_i \neq 0$ et $p_i < s_i$ et $n_i < s_i$), nous

ALGO. 7.6 – Fonction `ClasserSommets`

Entrées :
- un brin b_f de la face F dont les sommets doivent être classés ;
- le plan P par rapport auquel les sommets doivent être testés ;
- la marque Ⓜ avec laquelle les brins de F sont marqués.

Sorties :
- le nombre de sommets dans la face ;
- le nombre de sommets se trouvant du côté positif ;
- le nombre de sommets se trouvant du côté négatif.

Résultat : Les sommets de la face F sont classés et marqués par l'une des marques $\oplus$, $\ominus$.

début
  $(n_s, n_p, n_n) \leftarrow (0, 0, 0)$
  $b \leftarrow b_f$
  répéter
    si `EstMarqué`$(b, \text{Ⓣ}) =$ `faux` **alors**
      `MarquerSi`$(<\alpha_1, \alpha_2>(b), \text{Ⓣ}, \text{Ⓜ})$
      $d \leftarrow$ `Distance`$(P,$ `Point`$(b))$
      si $d < -\mathcal{E}$ **alors** `MarquerSi`$(<\alpha_1, \alpha_2>(b), \ominus, \text{Ⓜ})$
      sinon si $d > \mathcal{E}$ **alors** `MarquerSi`$(<\alpha_1, \alpha_2>(b), \oplus, \text{Ⓜ})$
    $b \leftarrow \alpha_1(\alpha_0(b))$
  jusqu'à $b = b_f$
  `Démarquer`$(<\alpha_0, \alpha_1>(b_f), \text{Ⓣ})$
  retourner (n_s, n_p, n_n)
fin

pouvons supposer que les deux faces possèdent une intersection et nous faisons alors appel à la fonction `IntersectionFacesSécantes` pour la générer. Comme pour `IntersectionFacesCoplanaires`, cette fonction retourne l'ensemble ω des arêtes d'intersection générées.

5. Si les conditions précédentes ne sont pas vérifiées, cela signifie que tous les sommets d'au moins l'une des faces se trouve du même côté du plan de l'autre face (*i.e.* $p_i = s_i$ ou $n_i = s_i$). Il n'y a alors aucune intersection possible et nous devons démarquer les brins classés par la fonction `ClasserSommets`.

6. Lorsque l'intersection est réalisée, nous devons récupérer les faces filles générées à l'aide de l'ensemble des arêtes d'intersection ω :

   a. Si ce dernier est vide, il n'y a pas eu d'intersection et par conséquent, la face n'a pas pu être découpée en plusieurs faces filles. Nous stockons alors les brins d'origine des faces F_1 et F_2 dans les listes de faces filles L_1 et L_2.

   b. Dans le cas contraire, nous ajoutons le contenu de ω à l'ensemble global des arêtes d'intertsection que nous notons Ω. Nous devons ensuite retrouver l'ensemble des faces filles générées par les arêtes de coupe. Nous faisons donc appel à la fonction `FacesFilles` dont le but est de récupérer toutes les faces marquées par Ⓜ et qui sont incidentes aux arêtes de coupes se trouvant dans ω. Le détail de cette fonction est donné dans l'algorithme 7.8.

7.4.5 Intersection de faces sécantes

Comme nous l'avons vu dans la section 7.3.3, l'intersection entre deux faces sécantes s'opère en plusieurs étapes qui sont les suivantes :

1. Calcul de la ligne de coupe : celle-ci est en fait modélisée par un simple vecteur permettant d'indiquer une direction de coupe.
2. Calcul des points d'intersection entre le bord de chaque face et le plan de l'autre face : ces points d'intersection sont ensuite stockés et triés selon la direction de coupe précédemment calculée.
3. Détection des segments d'intersection communs aux deux faces : ceci se fait à l'aide du tri précédemment obtenu.
4. Insertion de sommets et arêtes sur les faces afin de modéliser l'intersection.

Pour réaliser l'algorithme correspondant aux traitements précédents, nous l'avons découpé en deux étapes qui sont la recherche des points d'intersection (étape 2) et la création des segments de coupe (étape 4). La détection des segments de coupe communs (étape 3) est elle aussi découpée en deux et intégrée aux deux parties précédentes.

ALGO. 7.7 – Fonction `IntersectionFaces`

Entrées :
- un brin b_1 de la première face F_1 ;
- un brin b_2 de la seconde face F_2 ;
- le plan P_1 de F_1 ;
- le plan P_2 de F_2.

Sorties : Deux listes de brins contenant les faces filles de F_1 et de F_2.

début

1 `Marquer`($<\alpha_0, \alpha_1>(b_1)$, Ⓜ) ; `Marquer`($<\alpha_0, \alpha_1>(b_2)$, Ⓜ)

2 $(s_1, p_1, n_1) \leftarrow$ `ClasserSommets`(b_1, P_2, Ⓜ)
$(s_2, p_2, n_2) \leftarrow$ `ClasserSommets`(b_2, P_1, Ⓜ)

3 **si** $p_1 = 0$ **et** $n_1 = 0$ **alors**
- **si** $p_2 > 0$ **alors** `Démarquer`($<\alpha_0, \alpha_1>(b_2)$, ⊕)
- **si** $n_2 > 0$ **alors** `Démarquer`($<\alpha_0, \alpha_1>(b_2)$, ⊖)
- $\omega \leftarrow$ `IntersectionFacesCoplanaires`(b_1, b_2, P_1, P_2, Ⓜ)

sinon si $p_2 = 0$ **et** $n_2 = 0$ **alors**
- **si** $p_1 > 0$ **alors** `Démarquer`($<\alpha_0, \alpha_1>(b_1)$, ⊕)
- **si** $n_1 > 0$ **alors** `Démarquer`($<\alpha_0, \alpha_1>(b_1)$, ⊖)
- $\omega \leftarrow$ `IntersectionFacesCoplanaires`(b_1, b_2, P_1, P_2, Ⓜ)

4 **sinon si** $p_1 < s_1$ **et** $n_1 < s_1$ **et** $p_2 < s_2$ **et** $n_2 < s_2$ **alors**
- $\omega \leftarrow$ `IntersectionFacesSécantes`(b_1, b_2, P_1, P_2, Ⓜ)

5 **sinon**
- `Démarquer`($<\alpha_0, \alpha_1>(b_1)$, ⊕) ; `Démarquer`($<\alpha_0, \alpha_1>(b_1)$, ⊖)
- `Démarquer`($<\alpha_0, \alpha_1>(b_2)$, ⊕) ; `Démarquer`($<\alpha_0, \alpha_1>(b_2)$, ⊖)

6 **si** $\omega = \emptyset$ **alors**

a
- $L_1 \leftarrow \{b_1\}$; $L_2 \leftarrow \{b_2\}$
- `Démarquer`($<\alpha_0, \alpha_1>(b_1)$, Ⓜ) ; `Démarquer`($<\alpha_0, \alpha_1>(b_2)$, Ⓜ)

sinon

b
- $\Omega \leftarrow \Omega \cup \omega$
- $(L_1, L_2) \leftarrow$ `FacesFilles`(ω, Ⓜ)

retourner (L_1, L_2)

fin

ALGO. 7.8 – Fonction `FacesFilles`

Entrées :
- un ensemble ω contenant des couples de brins désignant une même arête d'intersection sur deux faces F_1 et F_2 ;
- une marque Ⓜ avec laquelle les brins des faces F_1 et F_2 sont marquées.

Sorties : Deux listes de brins contenant les faces filles de F_1 et de F_2.

début
  $L_1 \leftarrow \emptyset$; $L_2 \leftarrow \emptyset$
  pour chaque $(b_1, b_2) \in \omega$ **faire**
    si `EstMarqué(`b_1,Ⓜ`)` **alors**
      `Démarquer(`$<\alpha_0, \alpha_1>(b_1)$, Ⓜ`)`
      `Enfiler(`L_1, b_1`)`
    si `EstMarqué(`$\alpha_2(b_1)$,Ⓜ`)` **alors**
      `Démarquer(`$<\alpha_0, \alpha_1>(\alpha_2(b_1))$, Ⓜ`)`
      `Enfiler(`L_1, $\alpha_2(\alpha_0(b_1))$`)`
    si `EstMarqué(`b_2,Ⓜ`)` **alors**
      `Démarquer(`$<\alpha_0, \alpha_1>(b_2)$, Ⓜ`)`
      `Enfiler(`L_2, b_2`)`
    si `EstMarqué(`$\alpha_2(b_2)$,Ⓜ`)` **alors**
      `Démarquer(`$<\alpha_0, \alpha_1>(\alpha_2(b_2))$, Ⓜ`)`
      `Enfiler(`L_2, $\alpha_2(\alpha_0(b_2))$`)`
  retourner (L_1, L_2)
fin

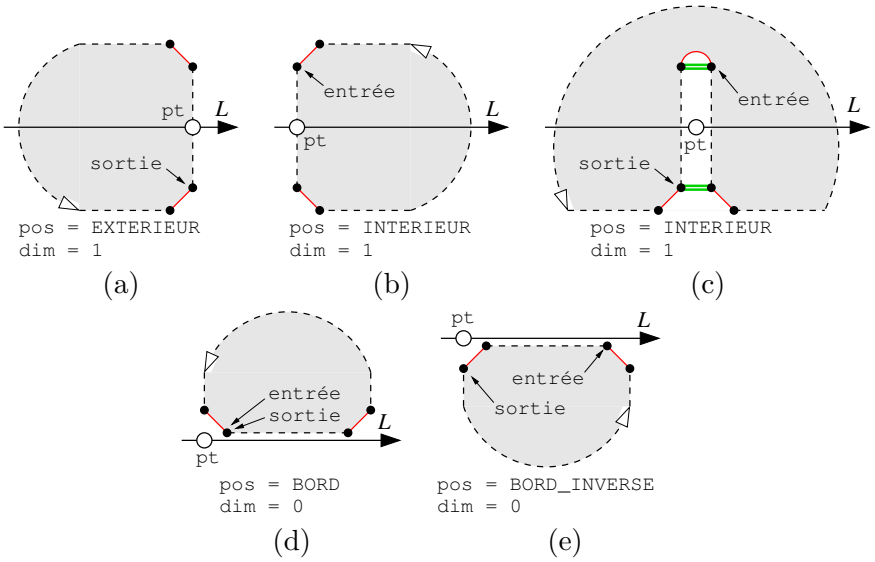


FIG. 7.21 – Les différents cas d'intersections possibles avec les segments du bord d'une face. La flèche blanche en pointillés indique l'orientation de la face et la flèche noire désignée L correspond à la direction de coupe. Pour chaque cas, les informations associées aux points d'intersections sont données.

7.4.5.1 Recherche des points d'intersection

Cette première partie de l'algorithme a pour but de déterminer les extrémités des hypothétiques segments de coupe. Elle consiste en le calcul géométrique des points d'intersection entre le bord d'une face F et un plan P (support d'une autre face). Aucun sommet topologique n'est donc inséré à cette étape.

Pour ce faire, nous supposons qu'un classement des sommets de F par rapport à P a été opéré à l'aide de l'algorithme 7.6 et que certains d'entre eux sont par conséquent marqués par les marques $\oplus$ et $\ominus$. Ainsi les points d'intersection que nous cherchons sont :

- les sommets de F n'étant marqués ni par $\oplus$, ni par $\ominus$;
- les points d'intersection entre les arêtes de F ayant un sommet marqué par $\oplus$ et l'autre par $\ominus$.

Comme nous l'avons vu précédemment (cf. section 7.3.3), les sommets d'intersection trouvés permettent de découper la ligne de coupe en plusieurs tronçons pouvant correspondre à des segments de coupe. Pour réaliser l'étape suivante concernant la création des segments de coupe, nous devons d'abord déterminer la position des tronçons par rapport au bord de la face. Nous associons alors d'autres informations aux points d'intersection trouvés. Ces informations sont répertoriées dans une structure notée `PointInter` dont les champs sont les suivants :

- `point` : un vecteur contenant les coordonnées du point d'intersection.
- `position` : la position par rapport aux limites de la face du tronçon se trouvant après le point courant sur la ligne de coupe. Ce champ peut recevoir quatre valeurs qui sont `INTERIEUR`, `EXTERIEUR`, `BORD`, `BORD_INVERSE`. Les deux dernières valeurs sont dues au fait

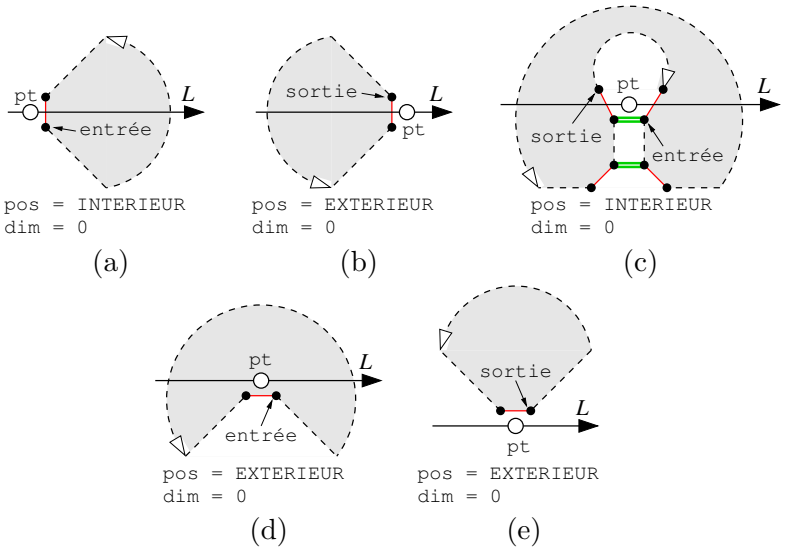


FIG. 7.22 – Les différents cas d'intersections possibles avec les sommets du bord d'une face. La flèche blanche en pointillés indique l'orientation de la face et la fleche noir désignée L correspond à la direction de coupe. Pour chaque cas, les informations associées aux points d'intersections sont données.

que les faces sont orientées et qu'un segment se trouvant sur le plan de coupe peut avoir un sens de parcours identique à celui de la direction de coupe ou bien il peut être opposé.

- `entrée` et `sortie` : deux brins appartenant à la cellule support du point d'intersection. Les valeurs de ces champs dépendent de la valeur du champ `position`. Par exemple, pour un tronçon se trouvant à l'intérieur de la face, le champ `entrée` du point précédent designera le secteur angulaire par lequel on entre dans la face et le champ `sortie` du point suivant désignera le secteur angulaire par lequel nous sortons de la face (cf. FIG. 7.21(a-b)). Pour un tronçon se trouvant sur un bord de la face, les champs `entrée` et `sortie` du point précédent désigneront respectivement le segment du bord concerné et le secteur angulaire par lequel nous atteignons ce point (cf. FIG. 7.21(d-e)).
- `dimension` : la dimension de la cellule support du point d'intersection (0 pour un sommet, 1 pour une arête).

Tous les cas pouvant être rencontrés ainsi que les valeurs correspondantes sont répertoriés sur la figure 7.21 pour les segments et sur la figure 7.22 pour les sommets. L'algorithme 7.9 décrit la manière d'établir un tel classement et se découpe comme suit :

1. La première partie de l'algorithme consiste à classer les sommets de la face F étant en intersection avec le plan de coupe. Ces sommets correspondent à ceux n'étant marqués par aucune des marques $\oplus$ et $\ominus$. Nous stockons dans chaque sommet concerné une structure `PointInter` que nous renseignons comme indiqué sur la figure 7.22 et chaque structure est ensuite ajoutée à une liste L. Notons qu'il est possible de tester plusieurs fois le même sommet lors du parcours de la face mais pour des secteurs angulaires différents. Les informations sont alors remplies en plusieurs étapes.

ALGO. 7.9 – Fonction `RechercherIntersections`

Entrées :
- un brin b_f de la face F à tester ;
- le plan P_f de F ;
- le plan de coupe P_c ;
- la direction de coupe D_c ;
- une marque Ⓜ avec laquelle les brins des faces F_1 et F_2 sont marquées.

Sorties : La liste triée selon D_c des points d'intersection existants entre le bord de la face F et le plan de coupe P_c.

```
début
    L ← ∅
    b ← b_f
1   répéter
        si EstMarqué(b, ⊕) = faux et EstMarqué(b, ⊖) = faux alors
            I ← Récupérer(<α1, α2, α3>(b), PointInter)
            si I = nil alors
                I ← PointInter(Point(b), EXTERIEUR, nil, nil, 0)
                Attribuer(<α1, α2, α3>(b), I)
                Enfiler(L, I)
            si position(I) = EXTERIEUR et DansSecteur(Dc, b, Normale(Pf)) alors
                position(I) ← INTERIEUR
                entrée(I) ← b
            si DansSecteur(−Dc, b, Normale(Pf)) alors
                sortie(I) ← b
        b ← α1(α0(b))
    jusqu'à b = b_f
2   répéter
        I1 ← Récupérer(<α1, α2, α3>(b), PointInter)
        I2 ← Récupérer(<α1, α2, α3>(α0(b)), PointInter)
        si I1 ≠ nil et I2 ≠ nil alors
            si Vecteur(point(I1), point(I2)).Dc > 0 alors
                position(I1) ← BORD
                entrée(I1) ← b
                si sortie(I1) = nil alors sortie(I1) ← b
            sinon
                position(I2) ← BORD_INVERSE
                entrée(I2) ← b
                si sortie(I2) = nil alors sortie(I2) ← α1(α0(b))
        b ← α1(α0(b))
    jusqu'à b = b_f
```

ALGO. 7.9 – Fonction `RechercherIntersections` *(suite et fin)*

```
3   répéter
      si Récupérer(<α1, α2, α3>(b), PointInter) ≠ nil alors
        Retirer(<α1, α2, α3>(b), PointInter)
      sinon si EstMarqué(b, ⊕) et EstMarqué(α0(b), ⊖) ou
               EstMarqué(b, ⊖) et EstMarqué(α0(b), ⊕) alors
        p ← Point (b)
        v ← Vecteur (b)
        I ← PointInter(Intersection(Pc, p, v), EXTERIEUR, nil, nil, 1)
        Enfiler(L, I)
a       si (Normale(Pf) ∧ v).Dc < 0 alors
          sortie(I) ← b
        sinon
          position(I) ← INTERIEUR
          entrée(I) ← b
b       si α2(b) ≠ b alors
          b' ← nil
          pour b'' ∈ <α2, α3>(b) faire
            si b'' ≠ b et EstMarqué(b'', Ⓜ) alors b' ← α0(b'')
          si b' ≠ nil alors
            si position(I) = INTERIEUR alors
              sortie(I) ← b'
            sinon
              position(I) ← INTERIEUR
              entrée(I) ← b'
      Démarquer(<α0, α2, α3>(b), ⊕)
      Démarquer(<α0, α2, α3>(b), ⊖)
      b ← α1(α0(b))
    jusqu'à b = bf
4   Trier L selon Dc
    retourner L
  fin
```

2. La seconde partie de l'algorithme parcourt une deuxième fois la face afin de repérer les segments ayant pour extrémités des sommets détectés à l'étape précédente. Ces sommets sont trouvés en regardant simplement si une structure `PointInter` leur est associée.

   Chaque segment détecté correspond alors à un segment longeant la ligne de coupe. Les informations associées aux extrémités de ces segments sont mises à jour en fonction de la direction de ces derniers par rapport à la direction de coupe (cf. FIG. 7.21(d-e)).

3. La troisième partie de l'algorithme s'occupe de détecter et calculer les points d'intersection entre le plan de coupe et les segments de la face. Nous parcourons alors une nouvelle fois la face et nous en profitons pour y enlever toutes les informations stockées lors des étapes précédentes (marques $\oplus$ et $\ominus$ structures `PointInter`), ce afin de rendre la face vierge pour les prochaines intersections.

   Les segments de la face étant intersectés sont ceux ayant une extrémité marquée par $\oplus$ et l'autre par $\ominus$. Pour chacun de ces segments, nous calculons son point d'intersection avec le plan de coupe. Nous ajoutons ensuite à la liste L une nouvelle structure `PointInter` dont les champs sont remplis à l'aide des deux traitements suivants :

   a. Nous regardons si la ligne de coupe est dirigée vers l'intérieur ou l'extérieur de la face. Le vecteur désignant l'intérieur de la face se détermine simplement en effectuant un produit vectoriel entre la normale à la face et le vecteur porteur du segment parcouru (se dernier détermine le sens de parcours de la face). Une fois ce vecteur obtenu, il suffit de regarder s'il pointe dans la même direction que la direction de coupe à l'aide d'un simple produit scalaire. Les informations associées au point d'intersection sont alors équivalentes à celles détaillées sur les figures 7.21(a-b).

   b. Nous testons ensuite si ce segment est une arête fictive en cherchant si un autre brin de l'orbite $<\alpha_2, \alpha_3>$ appartient à la face (*i.e.* un brin marqué par la marque Ⓜ). Si tel est le cas, les informations du point d'intersection sont mises à jour comme sur la figure 7.21(c).

4. Pour finir, nous trions la liste L de telle manière que le vecteur formé par deux points d'intersection successifs dans L soit orienté dans le même sens que la direction de coupe.

7.4.5.2 Création des segments de coupe

Après avoir recherché les intersections entre le bord de chaque face et le plan de l'autre face, nous obtenons deux listes triées contenant les points d'intersection. Nous pouvons alors déduire de ces deux listes la position des segments de coupe par rapport aux bords des faces. En particulier, les segments de coupe correspondent aux parties communes des listes se trouvant à l'intérieur ou sur le bord des faces.

Pour détecter ces parties communes, nous parcourons les deux listes en parallèle de manière à traiter les points des deux listes dans leur ordre d'apparition sur la ligne de coupe. À chaque segment de coupe détecté, nous mettons à jour la topologie en insérant des sommets et arêtes sur les faces.

L'algortihme 7.10 décrit tous les traitements à effectuer pour réaliser l'intersection entre deux faces sécantes. Celui-ci commence par calculer la direction de coupe D_c entre les deux faces F_1 et

ALGO. 7.10 – Fonction `IntersectionFacesSécantes`

Entrées :
- un brin b_1 de la première face F_1 ;
- un brin b_2 de la seconde face F_2 ;
- le plan P_1 de F_1 ;
- le plan P_2 de F_2 ;
- une marque Ⓜ avec laquelle les brins des faces F_1 et F_2 sont marquées.

Sorties : Un ensemble contenant des couples de brins désignant une même arête d'intersection sur chaque face F_1 et F_2.

début
- $D_c \leftarrow$ `Normale(`F_1`)` $\wedge$ `Normale(`F_2`)`
- $L_1 \leftarrow$ `RechercherIntersections(`b_1, P_1, P_2, D_c, Ⓜ`)`
- $L_2 \leftarrow$ `RechercherIntersections(`b_2, P_2, P_1, D_c, Ⓜ`)`

1 $\omega \leftarrow \emptyset$
- $I_1^-, I_2^- \leftarrow$ `nil`
- $I_1^+, I_2^+ \leftarrow$ `Tête(`L_1`)`, `Tête(`L_2`)`
- $pos_1, pos_2, pos_1^-, pos_2^- \leftarrow$ `EXTERIEUR`
- $config^-, config^+ \leftarrow 0$
- $som_1, som_2 \leftarrow$ `nil`
- $seg_1, seg_2 \leftarrow$ `nil`

2 **tant que** $I_1^+ \neq$ `nil` **et** $I_2^+ \neq$ `nil` **faire**
- $v \leftarrow$ `Vecteur(point(`I_1^+`)`, `point(`I_2^+`))`

3 **si** $|v| \leq \mathcal{E}$ **alors** $config^+ \leftarrow 3$
sinon si $v \cdot D_c < 0$ **alors** $config^+ \leftarrow 1$
sinon $config^+ \leftarrow 2$

4 **si** $pos_1 \neq$ `EXTERIEUR` **et** $pos_2 \neq$ `EXTERIEUR` **alors**
- **pour** $i \in [1, 2]$ **faire**
  - **suivant** la valeur de pos_i **faire**
    - **cas où** $pos_i =$ `BORD` **alors** exécuter sous-code n°1
    - **cas où** $pos_i =$ `BORD_INVERSE` **alors** exécuter sous-code n°2
    - **cas où** $pos_i =$ `INTERIEUR` **alors** exécuter sous-code n°3
  - **si** `EstMarqué(`seg_i, Ⓕ`)` **alors** `Démarquer(`$<\alpha_0, \alpha_2, \alpha_3>(seg_i)$, Ⓕ`)`
- $\omega \leftarrow \omega \cup \{(seg_1, seg_2)\}$

5 **pour** $i \in [1, 2]$ **faire**
- **si** $config^+ = i$ **ou** $config^+ = 3$ **alors**
  - $I_i^- \leftarrow I_i^+$
  - $I_i^+ \leftarrow$ `Suivant(`L_i, I_i^+`)`
  - $pos_i^- \leftarrow pos_i$
  - $pos_i \leftarrow$ `position(`I_i^-`)`
  - $som_i \leftarrow$ `nil`

- $config^- \leftarrow config^+$

retourner ω

fin

F_2 puis fait appel à la fonction de recherche d'intersection décrite précédemment (cf. ALGO. 7.9). Il récupère ainsi deux listes L_1 et L_2 contenant les points d'intersection existants entre le bord de chaque face et le plan de l'autre face et les utilise de la manière suivante :

1. Nous initialisons plusieurs variables utiles dans la suite de l'algorithme. Ces variables correspondent aux éléments suivants :

   I_1^-, I_1^+ Points d'intersection consécutifs dans la liste L_1. I_1^- correspond au point le moins en avant dans la liste et I_1^+ au point le plus en avant.

   I_2^-, I_2^+ Idem que pour les variables précédentes mais pour la liste L_2.

   pos_1, pos_1^- Les positions respectives par rapport au bord de la face F_1 du tronçon courant et du tronçon précédent dans la liste L_1. Plus précisément, pos_1 correspond à la valeur de `position(`I_1^-`)` et pos_1^- à l'ancienne valeur de `position(`I_1^-`)`.

   pos_2, pos_2^- Idem que pour les variables précédentes mais pour la liste L_2.

   $config^-$, $config^+$ La configuration des points d'intersection définissant les extrémités du tronçon courant. La variable $config^-$ correspond à la première extrémité tandis que $config^+$ correspond à la seconde. Ces variables peuvent prendre pour valeur un entier compris entre 1 et 3. Cette valeur permet alors d'indiquer si une extrémité provient d'un point d'intersection sur la première face (valeur 1), sur la deuxième face (valeur 2) ou bien sur les deux faces en même temps (valeur 3).

   som_1, som_2 Les derniers sommets topologiques ajoutés sur chacune des faces.

   seg_1, seg_2 Les segments d'intersection modélisés sur chacune des faces.

2. Tant que les listes L_1 et L_2 possèdent des points d'intersection, nous effectuons une boucle qui va se charger de défiler tous les points d'intersection des deux listes et d'effectuer les traitements permettant de générer les segments de coupe.
3. Nous regardons la position des points d'intersection courant par rapport à la direction de coupe et nous en déduisons la configuration de la seconde extrémité du tronçon.
4. Si le tronçon courant se trouve à l'intérieur du bord des deux faces, nous pouvons, pour chacune des deux faces, effectuer les traitrements permettant de générer un segment de coupe en fonction de la position du tronçon par rapport au bord de celle-ci. Pour cela, nous faisons appel aux algorithmes 7.11, 7.12 et 7.13.

   Les segments de coupe ainsi générés sont ensuite ajoutés à la liste ω répertoriant tous les segments de coupe existants entre les deux faces.
5. Lorsque le traitement du tronçon actuel est fini, nous récupérons les éléments suivants dans les listes L_1 et L_2 en fonction de la configuration de la seconde extrémité du tronçon.

L'algorithme 7.11 décrit les traitements à effectuer pour modéliser un segment d'intersection sur une face dans le cas ou celui-ci intervient sur le bord de cette dernière et que l'arête concernée est orientée dans le même sens que la direction de coupe. L'algorithme effectue les traitements suivants :

ALGO. 7.11 – Fonction `IntersectionFacesSécantes` – sous-code n°1

```
   cas où pos_i = BORD alors
1  |  seg_i ← entrée(I_i^-)
2  |  si config^- ≠ i et config^- ≠ 3 alors
a  |  |  si pos^-_(3−i) ≠ EXTERIEUR alors
   |  |  |  seg_i ← som_i
b  |  |  sinon
   |  |  |  seg_i ← ÉclaterArête(seg_i, point(I^-_(3−i)))
   |  |  |  entrée(I_i^-) ← seg_i
3  |  si config^+ ≠ i et config^+ ≠ 3 alors
   |  |  ÉclaterArête(seg_i, point(I^+_(3−i)))
4  |  som_i ← α1(α0(seg_i))
```

1. Comme le tronçon courant est confondu avec le bord de la face, le segment de coupe existe déjà et il n'y pas besoin d'en générer un nouveau.
2. Si le premier sommet du tronçon correspond uniquement à une intersection avec le bord de l'autre face, cela veut dire que nous devons rajouter un sommet sur le segment courant. Cependant, nous pouvons être confrontés à deux cas possibles :
   a. Si la position du tronçon précédent était à l'intérieur de l'autre face, comme le premier sommet du tronçon courant ne correspond pas à une intersection sur la face courante, un segment de coupe a alors obligatoirement été calculé pour le précédent tronçon. Dans ce cas, le sommet à insérer existe déjà et correspond à la seconde extrémité du précédent segment de coupe généré, à savoir som_i.
   b. Dans le cas contraire, nous éclatons l'arête concernée sur la face courante en y insérant un point correspondant à l'intersection avec le bord de l'autre face. De plus, nous mettons à jour le sommet d'entrée dans la face courante pour qu'il soit réutilisé au cas où les tronçons suivants correspondraient à des intersections avec le même segment du bord de la face.
3. Si la seconde extrémité du tronçon correspond uniquement à une intersection avec le bord de l'autre face, nous devons comme précédemment rajouter un sommet sur l'arête concernée. Ici, il n'est pas nécessaire de mettre à jour le brin correspondant au sommet d'entrée dans la face, étant donné que le cas est géré par le traitement expliqué en 2.a.
4. Au final, nous récupérons la seconde extrémité du segment de coupe et la stockons dans som_i.

L'algorithme 7.12 décrit les traitements à effectuer pour modéliser un segment d'intersection sur une face dans le cas ou celui-ci intervient sur le bord de cette dernière et que l'arête concernée est orientée dans le sens contraire de la direction de coupe. L'algorithme effectue les traitements suivants :

1. Comme précédemment, l'intersection s'effectuant sur le bord de la face, le segment de coupe existe déjà et il n'y pas besoin d'en générer un nouveau.

ALGO. 7.12 – Fonction `IntersectionFacesSécantes` – sous-code n°2

```
   cas où pos_i = BORD_INVERSE alors
1  |  som_i ← entrée(I_i^-)
2  |  si config^- ≠ i et config^- ≠ 3 alors
   |  |  si pos^-_(3-i) = EXTERIEUR alors
   |  |  |  ÉclaterArête(som_i, point(I^-_(3-i)))
3  |  si config^+ ≠ i et config^+ ≠ 3 alors
   |  |  som_i ← ÉclaterArête(som_i, point(I^+_(3-i)))
4  |  seg_i ← som_i
```

2. Si la première extrémité du tronçon correspond uniquement à une intersection avec le bord de l'autre face, nous devons ajouter un sommet sur l'arête concernée. Cependant, l'ajout du sommet se fait uniquement si le tronçon précédent n'a pas déjà généré un segment de coupe et par conséquent le dit sommet.
3. Si la seconde extrémité du tronçon correspond uniquement à une intersection avec le bord de l'autre face, nous ajoutons un sommet sur l'arête concernée.
4. Finalement, nous récupérons la seconde extrémité du segment de coupe et la stockons dans la variable som_i. Notons par ailleurs que les traitements effectués dans cet algorithme considèrent que l'arête est orientée dans le sens contraire de la droite de coupe. Ainsi, les brins correspondant aux secteurs d'entrée dans la face ne sont pas altérés et n'ont pas besoin d'être mis à jour.

L'algorithme 7.13 décrit les traitements à effectuer pour modéliser un segment d'intersection sur une face dans le cas ou celui-ci intervient à l'intérieur des limites de la face. Cet algorithme est plus complexe que les précédents au vu des nombreux cas à gérer. Il se décompose en deux parties principales correspondant à la modélisation de deux sommets A et B qui sont les extrémités du segment de coupe devant être ajouté. Les traitements effectués sont les suivants :

1. Si la première extrémité du tronçon correspond à une intersection avec le bord de la face courante, nous devons modéliser le sommet A sur la cellule concernée. Si cette cellule est déjà un sommet, nous récupérons seulement un brin de celui-ci. Dans le cas contraire, il s'agit d'une arête et nous y insérons un nouveau sommet.
2. Si la première extrémité du tronçon correspond uniquement à une intersection avec le bord de l'autre face, nous devons ajouter le sommet A à l'intérieur des limites de la face courante. Cependant, nous pouvons différencier ici trois cas de figures :
   a. Si le tronçon précédemment traité ne se trouvait pas à l'extérieur de l'autre face, cela signifie qu'un segment de coupe a été modélisé pour ce tronçon. Ainsi, le sommet que nous voulons insérer existe déjà et correspond à l'extrémité du dernier segment de coupe, c'est-à-dire som_i.
   b. Dans le cas contraire et si la deuxième extrémité du tronçon courant correspond uniquement à une intersection avec le bord de l'autre face, cela signifie que le segment

ALGO. 7.13 – Fonction `IntersectionFacesSécantes` – sous-code n°3

cas où $pos_i =$ `INTERIEUR` **alors**

1 **si** $config^- = i$ **ou** $config^- = 3$ **alors**
  si `dimension`$(I_i^-) = 0$ **alors** $A \leftarrow$ `entrée`(I_i^-)
  sinon $A \leftarrow$ `ÉclaterArête(entrée`(I_i^-), `point`$(I_i^-))$
2 **sinon**
a **si** $pos^-_{(3-i)} \neq$ `EXTERIEUR` **alors** $A \leftarrow som_i$
b **sinon si** $config^+ \neq i$ **et** $config^+ \neq 3$ **alors**
    si $som_i \neq$ `nil` **alors** $A \leftarrow$ `entrée`(I_i^-)
    sinon $A \leftarrow som_i$
    $A \leftarrow$ `InsérerSommet(`A, `point`$(I^-_{(3-i)}))$
c **sinon** $A \leftarrow$ `nil`
3 **si** $config^+ = i$ **ou** $config^+ = 3$ **alors**
a **si** `dimension`$(I_i^+) = 0$ **alors**
    si `sortie`$(I_i^+) \neq$ `nil` **alors** $B \leftarrow$ `sortie`(I_i^+)
    sinon $B \leftarrow$ `entrée`(I_i^+)
   sinon
    $B \leftarrow$ `ÉclaterArête(sortie`(I_i^+), `point`$(I_i^+))$
b **si** $A \neq nil$ **alors**
    $seg_i \leftarrow$ `ÉclaterFace(`A, B`)`
   sinon
    $seg_i \leftarrow \alpha_2(\alpha_0($`InsérerArête(`$B$, `point`$(I^-_{(3-i)}))))$
    $A \leftarrow seg_i$
c **si** `position`$(I_i^+) =$ `INTERIEUR` **alors**
    si `dimension`$(I_i^+) = 1$ **alors**
     `dimension`$(I_i^+) \leftarrow 0$
     `entrée`$(I_i^+) \leftarrow \alpha_1(\alpha_0($`entrée`$(I_i^+)))$
    sinon si (`Vecteur(entrée`$(I_i^+))$ $\wedge D_c$).`Normale`$(F_i) < 0$ **alors**
     `entrée`$(I_i^+) \leftarrow \alpha_2(\alpha_0(seg_i))$
4 **sinon**
  $seg_i \leftarrow$ `InsérerArête(`A, `point`$(I^+_{(3-i)}))$
  $B, som_i \leftarrow \alpha_1(\alpha_0(seg_i))$

de coupe que nous modéliserons ici ne sera relié à aucun bord de la face courante. Nous devons alors insérer un sommet dans la face qui sera relié à un bord de celle-ci grâce à une arête fictive.

Pour ne pas compliquer l'écriture de l'algorithme nous faisons appel ici uniquement à la fonction `InsérerSommet` en récupérant soit l'extrémité du dernier segment de coupe inséré, soit un sommet proche de la dernière cellule intersectée. Cependant, le choix fait ici concernant le sommet auquel relier l'arête fictive n'est pas suffisant et la procédure à suivre concernant ce problème est expliquée en détail en section 7.3.3.1.

c. Si les deux cas précédents ne sont pas vérifiés, cela signifie que le segment de coupe qui devra être rajouté sera relié au bord de la face courante via sa deuxième extrémité. Nous n'ajoutons alors ici aucun sommet afin d'éviter l'insertion d'arêtes fictives inutiles et nous ajouterons le segment de coupe directement lors du traitement de la seconde extrémité.

3. Si la deuxième extrémité du tronçon correspond à une intersection avec le bord de la face courante, nous devons modéliser le sommet B sur la cellule concernée. Ensuite, lorsque nous avons les deux sommets A et B, nous pouvons les relier à l'aide d'une arête correspondant au segment de coupe. Tout ceci s'effectue en trois étapes qui sont les suivantes :

   a. Nous commençons par modéliser le sommet B en regardant la dimension de la cellule intersectée. Si la cellule est déjà un sommet, nous récupérons uniquement un brin de celui-ci. Notons ici que nous effectuons un test afin de différencier deux cas possibles représentés sur les figures 7.22(a) et 7.22(b) se trouvant à la page 107. Dans le cas où la cellule intersectée est une arête, nous insérons un sommet sur celle-ci.

   b. Dans un deuxième temps, nous regardons si le sommet A a bien été généré lors des précédentes étapes. Si c'est le cas, nous pouvons insérer une arête entre les sommets A et B. Dans le cas contraire, cela signifie que le sommet A n'a pas été modélisé afin d'éviter l'insertion d'une arête fictive inutile. Nous pouvons alors ajouter une arête dans la face à l'aide de la méthode `InsérerArête` et en la reliant au sommet B.

   c. Pour finir, nous devons mettre à jour les informations concernant la deuxième extrémité du tronçon dans le cas où le tronçon suivant se trouve à l'intérieur de la face. Ainsi, si la deuxième extrémité du tronçon correspond à une intersection avec une arête, nous la mettons à jour en lui affectant le nouveau sommet généré. Par contre, s'il s'agit déjà d'un sommet, nous devons modifer le secteur d'entrée dans le cas où celui-ci serait erroné. Le cas correspondant est montré sur la figure 7.23 et peut être détecté en calculant l'orientation du segment incident au secteur d'entrée par rapport à la ligne de coupe.

4. Si la deuxième extrémité du tronçon correspond uniquement à une intersection avec le bord de l'autre face, cela signifie que le second sommet du segment de coupe ne se trouvera pas sur le bord de la face courante mais à l'intérieur des limites de celle-ci. Nous n'avons pas besoin ici d'ajouter un nouveau sommet car celui-ci sera automatiquement créé en générant le segment de coupe. Nous appelons alors simplement la fonction `InsérerArête` prévue à cet effet.

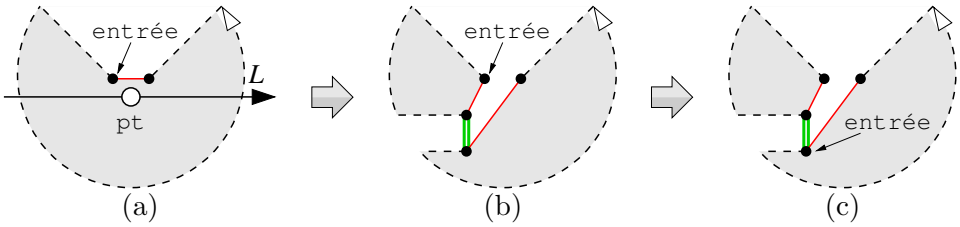


FIG. 7.23 – Cas de figure pouvant provoquer la mise à jour des informations associées à un point d'intersection : (a) la topologie du sommet avant la génération du segment de coupe ; (b) le même sommet après avoir inséré le segment de coupe ; (c) modification du pointeur sur brin désignant le secteur angulaire d'entrée associé au sommet.

7.4.6 Intersection de faces coplanaires

Comme nous l'avons vu dans la section 7.3.4, l'intersection de faces coplanaires utilise des techniques similaires à l'intersection de faces sécantes. Le processus précédent est alors répété pour chaque arête d'une face afin de déterminer les points d'intersection avec les arêtes de l'autre face. Ensuite, lorsque tous les points d'intersection sont déterminés, nous parcourons le bord des faces et générons les segments de coupe sur chacune des deux faces.

L'algorithme 7.14 décrit la manière de procéder pour réaliser les deux étapes précédentes. En plus d'un ensemble ω contenant les segments d'intersection, il utilise un ensemble γ contenant les points d'intersection entre les arêtes des deux faces. De plus, afin de différencier et de mémoriser les faces doubles générées, il utilise une marque Ⓓ pour les marquer. Son fonctionnement est le suivant :

1. La première étape de l'algorithme consiste à générer les points d'intersection entre les arêtes des deux faces. Cependant, il est possible que les faces ne possèdent aucune intersection et nous devons alors détecter si une face est incluse dans l'autre. Pour ce faire, nous utilisons une boucle pour parcourir les deux faces afin de déterminer la position de chacune de leur arête par rapport au bord de l'autre face. Cette boucle est interrompue dès qu'un point d'intersection est stocké dans l'ensemble γ (cf. étape 6).

   Au début de la boucle, nous initialisons un booléen *inter* à `vrai`. Ce booléen est mis à jour lors des traitements suivants et permet de savoir si la face courante se trouve à l'intérieur de l'autre face.

2. Pour chaque face, nous parcourons chacune de ses arêtes et nous calculons un plan de coupe lui correspondant. Ce plan passe par les deux sommets de l'arête et est perpendiculaire à la face courante (cf. section 7.3.4.1). Il est ensuite utilisé pour classer les sommets de l'autre face à l'aide de la fonction `ClasserSommets`.

3. Nous devons ensuite calculer les points d'intersection existants en fonction du classement précédemment effectué. Auparavant, nous initialisons un second booléen $inter'$ permettant de spécifier si l'arête courante se trouve à l'intérieur ou non de l'autre face.

   Si les sommets se trouvent tous du même côté du plan, l'arête se trouve en dehors de l'autre face et aucun traitement n'est nécessaire. Nous démarquons alors les sommets de

ALGO. 7.14 – Fonction `IntersectionFacesCoplanaires`

Entrées :
- un brin b_1 de la première face F_1 ;
- un brin b_2 de la seconde face F_2 ;
- le plan P_1 de F_1 ;
- le plan P_2 de F_2 ;
- une marque Ⓜ avec laquelle les brins des faces F_1 et F_2 sont marquées.

Sorties : Un ensemble contenant des couples de brins désignant une même arête d'intersection sur chaque face F_1 et F_2.

début
- $\omega \leftarrow \emptyset$
- $\gamma \leftarrow \emptyset$

1 **pour** $i \in [1,2]$ **faire**
- $inter \leftarrow$ `vrai`
- $b_a \leftarrow b_i$

2 **répéter**
- $s_1 \leftarrow b_a$
- $b_a \leftarrow \alpha_1(\alpha_0(b_a))$
- $D_c \leftarrow$ `Vecteur(`s_1`)`
- $P_a \leftarrow$ `Plan(Normale(`P_i`)` $\wedge$ D_c, `Point(`s_1`))`
- $(n_s, n_p, n_n) \leftarrow$ `ClasserSommets(`$b_{(3-i)}$, P_a, Ⓜ`)`

3 $inter' \leftarrow$ `faux`

si $n_p = n_s$ **alors** `Démarquer(`$<\alpha_0, \alpha_1>(b_{(3-i)})$, $\oplus$`)`

sinon si $n_n = n_s$ **alors** `Démarquer(`$<\alpha_0, \alpha_1>(b_{(3-i)})$, $\ominus$`)`

sinon exécuter sous-code n°1

4 **si** $inter' =$ `faux` **alors** $inter =$ `faux`

jusqu'à $b_a = b_i$

5 **si** $inter =$ `vrai` **alors**
- $s_2 \leftarrow$ `InsérerSommet(`$b_{(3-i)}$, `Point(`b_i`))`
- **si** $i = 1$ **alors** $\gamma \leftarrow \gamma \cup \{(b_i, s_2)\}$
- **sinon** $\gamma \leftarrow \gamma \cup \{(s_2, b_i)\}$

6 **si** $\gamma \neq \emptyset$ **alors** arrêter la boucle

7 **pour chaque** $(s_1, s_2) \in \gamma$ **faire** exécuter sous-code n°2

8 **pour chaque** $(s_1, s_2) \in \gamma$ **faire**
- **pour** $i \in [1,2]$ **faire**
  - **pour chaque** brin $b \in <\alpha_1, \alpha_2, \alpha_3>(s_i)$ **faire**
    - **si** `EstMarqué(`b, Ⓓ`)` $\neq$ `EstMarqué(`$\alpha_1(b)$, Ⓓ`)` **alors**
      - `Marquer(`$<\alpha_0, \alpha_1, \alpha_2>(b)$, Ⓓ`)`

retourner ω

fin

l'autre face des marques $\oplus$ et $\ominus$ pour passer à la prochaine arête. Si les sommets ne se trouvent pas tous du même côté du plan, il est possible qu'il y ait des intersections. Nous exécutons alors le sous-code n°1 (cf. Algo. 7.15) qui permet de calculer les points d'intersection entre l'arête courante et le bord de l'autre face.

4. Une fois l'arête courante traitée, nous mettons à jour le booléen *inter* en fonction de la position de l'arête courante.

5. Lorsque toutes les arêtes de la face ont été traitées, nous savons si cette dernière se trouve incluse ou non dans l'autre face. Si c'est le cas, nous devons insérer dans l'autre face un sommet correspondant à un sommet de la face courante. Cette opération permet de pouvoir effectuer la seconde partie de l'algorithme, c'est-à-dire la création des segments de coupe.
   Comme précédemment pour l'algorithme d'intersection de faces sécantes, l'insertion d'un sommet au sein d'une face n'est pas une opération anodine. En effet, nous devons vérifier lors de cette étape qu'aucune arête des deux faces n'est intersectée par l'arête fictive permettant de relier le sommet au bord de la face. Cependant, pour simplifier l'algorithme, nous ne détaillons pas ici ce processus qui est similaire au précédent.
   Une fois qu'un sommet est inséré dans l'autre face, les deux sommets correspondants sont insérés dans l'ensemble γ. Notons ici que l'ordre des brins dans le couple ajouté n'est pas le même en fonction de la face traitée. Ceci est fait de manière à ce que le premier élément du couple soit un brin de la première face et le second un brin de la seconde.

6. Lorsqu'une face a été entièrement traitée, nous arrêtons la boucle dans le cas où l'ensemble des points d'intersection γ n'est pas vide. En effet, il est inutile de parcourir la seconde face car les points d'intersection trouvés seront les mêmes. Cependant, si γ est vide, cela signifie que le bord des faces ne se croisent pas et que la première face n'est pas incluse dans la seconde. Nous devons alors traiter la deuxième face afin de déterminer si elle est incluse dans la première ou si elle se trouve à l'extérieur.

7. Après avoir calculé tous les points d'intersection, nous modélisons les segments de coupe en exécutant le sous-code n°2 (cf. Algo. 7.18).

8. Pour finir, nous parcourons les faces incidentes au sommets d'intersection et nous propageons la marque Ⓓ afin que les faces filles partiellement marquées le soient totalement.

L'algorithme 7.15 décrit la manière de procéder pour calculer les points d'intersection existants entre une arête d'une face et le bord de l'autre face. Les sommets de l'arête sont désignés par des brins s_1 et b_a et la face par un brin $b_{(3-i)}$. De plus D_c désigne la ligne de coupe correspondant au vecteur directeur de l'arête. Le fonctionnement de l'algorithme est le suivant :

1. Nous commençons par construire les listes de points d'intersection. La liste L_1 correspond à l'intersection avec l'arête courante et contient alors deux éléments correspondant à ses deux sommets. La liste L_2 correspond aux points d'intersection entre la ligne de coupe D_c et les arêtes de l'autre face.

2. Nous faisons ensuite la même chose que pour l'algorithme d'intersection de faces sécantes (cf. section 7.4.5.2) : nous initialisons des variables permettant le parcours des deux listes en parallèle puis nous effectuons une boucle jusqu'à ce qu'il n'y ait plus de points dans

ALGO. 7.15 – Fonction `IntersectionFacesCoplanaires` – sous-code n°1

début

1 $L_1 \leftarrow \emptyset$
`Enfiler`(L_1, `PointInter(Point(`s_1`), BORD,` s_1, s_1, 0))
`Enfiler`(L_1, `PointInter(Point(`b_a`), EXTERIEUR,nil,` b_a, 0))
$L_2 \leftarrow$ `RechercherIntersections`($b_{(3-i)}$, $P_{(3-i)}$, P_a, D_c, Ⓜ)
2 $I_1^-, I_2^- \leftarrow$ `nil`
$I_1^+, I_2^+ \leftarrow$ `Tête`(L_1), `Tête`(L_2)
$pos_1, pos_2 \leftarrow$ `EXTERIEUR`
$config^-, config^+ \leftarrow 0$
tant que $I_1^+ \neq$ `nil` **et** $I_2^+ \neq$ `nil` **faire**
  $v \leftarrow$ `Vecteur(point(`I_1^+`), point(`I_2^+`))`
  si $|v| \leq \mathcal{E}$ **alors** $config^+ \leftarrow 3$
  sinon si $v \cdot D_c < 0$ **alors** $config^+ \leftarrow 1$
  sinon $config^+ \leftarrow 2$
3 **si** $pos_1 \neq$ `EXTERIEUR` **alors**
a **si** $inter' =$ `vrai` **et** ($config^- \neq 1$ **ou** $config^+ \neq 1$) **alors** $inter' \leftarrow$ `faux`
b **si** $config^- \neq 2$ **et** ($pos_2 =$ `INTERIEUR` **ou** ($pos_2 \neq$ `EXTERIEUR` **et**
    `IntérieurFace(`s_1, P_i`) · IntérieurFace(entrée(`I_2^-`),` $P_{(3-i)}$`) >` 0)) **alors**
    `Marquer(`s_1, Ⓓ`)`
c **si** $pos_2 =$ `INTERIEUR` **alors** exécuter sous-code n°1.a
  sinon exécuter sous-code n°1.b
d **si** $config^+ = 2$ **et** (`position(`I_2^+`)` $\neq$ `INTERIEUR` **ou**
    (`position(`I_2^+`)` $\neq$ `EXTERIEUR` **et**
    `IntérieurFace(`s_0, P_i`) · IntérieurFace(entrée(`I_2^+`),` $P_{(3-i)}$`) >` 0)) **alors**
    `Marquer(`s_1, Ⓓ`)`
4 **sinon si** $I_1^- =$ `nil` **alors**
  si $pos_2 =$ `INTERIEUR` **alors** $inter' \leftarrow$ `vrai`
  sinon $inter' \leftarrow$ `faux`
5 **pour** $j \in [1, 2]$ **faire**
  si $config^+ = j$ **ou** $config^+ = 3$ **alors**
    $I_j^- \leftarrow I_j^+$
    $I_j^+ \leftarrow$ `Suivant(`L_j, I_j^+`)`
    $pos_j \leftarrow$ `position(`I_j^-`)`
  $config^- \leftarrow config^+$

fin

les listes. À chaque début de boucle, nous calculons la configuration du second sommet du tronçon courant.

3. Si le tronçon courant se trouve entre les deux sommets de l'arête courante, nous testons la position du tronçon par rapport au bord de l'autre face afin de détecter les points d'intersection. Nous effectuons alors différents traitements en fonction des cas possibles :

   a. Si les sommets du tronçon courant ne correspondent pas aux sommets de l'arête courante, cela signifie qu'il existe une intersection avec une cellule de l'autre face et par conséquent, que l'arête courante ne se trouve pas entièrement incluse à l'intérieur de l'autre face.

   b. Si le premier sommet du tronçon courant correspond à un sommet de l'arête courante et que le tronçon se trouve à l'intérieur de l'autre face ou bien sur le bord et que les intérieurs des deux faces sont orientées dans le même sens, alors cela signifie que l'arête se trouve à l'intérieur de l'autre face et qu'elle définira par conséquence le bord d'une face double. Nous marquons alors la première moitié de l'arête avec la marque Ⓓ.

   c. Si le tronçon courant se situe à l'intérieur de l'autre face, nous exécutons le sous-code n°1.a dédié au traitement des intersections franches. Sinon, nous exécutons le sous-code n°1.b dédié au traitement des intersections entre arête colinéaires.

   d. Si l'étape précédente génère un sommet sur l'arête et que le tronçon suivant se trouve à l'intérieur de l'autre face, nous effectuons les mêmes traitements que pour l'étape b afin de marquer l'arête située derrière le nouveau sommet.

4. Dans le cas où le tronçon ne se trouve pas entre les deux sommets de l'arête courante et qu'aucun des sommets de cette dernière n'a été extrait de la liste L_1, ceci signifie que nous nous situons avant l'arête. Nous mettons alors à jour la variable $inter'$ en fonction de la position du tronçon par rapport au bord de l'autre face.

5. Nous terminons la boucle en passant aux éléments suivants dans les listes L_1 et L_2.

L'algorithme 7.16 décrit les traitements à effectuer pour insérer des sommets sur l'arête courante de la première face dans le cas où elle possède des intersections franches avec le bord de la seconde face. Le fonctionnement de l'algorithme est le suivant :

1. Si le premier sommet du tronçon courant correspond à la fois au premier sommet de l'arête courante et à une intersection avec une arête de l'autre face, nous insérons un sommet sur cette dernière puis nous ajoutons le couple de sommet à l'ensemble γ. Notons ici que nous ne gérons pas le cas où le premier sommet du tronçon correspondrait uniquement à une intersection avec le bord de la deuxième face. En effet, ce cas est traité par l'étape suivante qui s'occupe de rajouter le sommet correspondant.

2. Si le second sommet du tronçon courant correspond à une intersection avec le bord de la deuxième face, nous devons insérer un sommet sur le bord de cette dernière ainsi que sur l'arête courante si cela est nécessaire. Pour cela, nous effectuons les traitements suivants :

   a. Nous commençons pas récupérer un brin appartenant à la cellule intersectée sur la seconde face.

ALGO. 7.16 – Fonction `IntersectionFacesCoplanaires` – sous-code n°1.a

début
1 **si** $config^- = 3$ **et** `dimension`$(I_2^-) = 1$ **alors**
  si $pos_2 =$ `EXTERIEUR` **alors** $s_2 \leftarrow$ `ÉclaterArête(sortie(`I_2^-`), Point(`s_1`))`
  sinon $s_2 \leftarrow$ `ÉclaterArête(entrée(`I_2^-`), Point(`s_1`))`
  si $i = 1$ **alors** $\gamma \leftarrow \gamma \cup \{(s_1, s_2)\}$
  sinon $\gamma \leftarrow \gamma \cup \{(s_2, s_1)\}$
2 **si** $config^+ \neq 1$ **alors**
a **si** `position`$(I_2^+) =$ `INTERIEUR` **alors** $s_2 \leftarrow$ `entrée`(I_2^+)
  sinon $s_2 \leftarrow$ `sortie`(I_2^+)
b **si** $config^+ = 2$ **alors**
    $s_1 \leftarrow$ `ÉclaterArête(`s_1`, Point(`I_2^+`))`
    si `dimension`$(I_2^+) = 1$ **alors** $s_2 \leftarrow$ `ÉclaterArête(`s_2`, Point(`I_2^+`))`
  sinon
    $s_1 \leftarrow \alpha_1(\alpha_0(s_1))$
    si `dimension`$(I_2^+) = 1$ **alors** $s_2 \leftarrow$ `ÉclaterArête(`s_2`, Point(`b_a`))`
c **si** $i = 1$ **alors** $\gamma \leftarrow \gamma \cup \{(s_1, s_2)\}$
  sinon $\gamma \leftarrow \gamma \cup \{(s_2, s_1)\}$
fin

b. Si le second sommet du tronçon correspond uniquement à une intersection avec le bord de la deuxième face, nous insérons un sommet sur l'arête courante ainsi que sur la seconde face. Dans le cas contraire, nous insérons uniquement un sommet sur le bord de la seconde face si cela est nécessaire.

c. Pour finir, nous ajoutons le couple de sommets correspondant à l'ensemble des points d'intersection γ.

L'algorithme 7.17 décrit les traitements à effectuer pour insérer des sommets sur l'arête courante de la première face dans le cas où elle est située sur le bord de la seconde face. Le fonctionnement de l'algorithme est le suivant :

1. Si le premier sommet du tronçon courant correspond uniquement à un sommet de l'arête courante, nous insérons un sommet sur le bord de l'autre face et mettons à jour les informations concernant ce sommet. Dans le cas contraire, le sommet existe déjà sur l'autre face et nous n'avons plus qu'à le récupérer.

2. Si le premier sommet du tronçon correspond au premier sommet de l'arête courante, le point d'intersection n'a pas pu être traité lors des étapes précédentes. Nous l'ajoutons alors à l'ensemble γ.

3. Si le second sommet du tronçon courant correspond uniquement à la deuxième extrémité de l'arête courante, nous devons insérer un sommet sur l'arête de la deuxième face et mettre à jour les informations associées à ce point d'intersection. Dans le cas contraire, nous devons insérer un sommet sur l'arête courante de la première face.

ALGO. 7.17 – Fonction `IntersectionFacesCoplanaires` – sous-code n°1.b

début

1 **si** $config^- = 1$ **alors**
  $s_2 \leftarrow$ `ÉclaterArête(entrée(`I_2^-`), Point(`s_1`))`
  si $pos_2 =$ `BORD` **alors** `entrée(`I_2^-`)` $\leftarrow s_2$
sinon $s_2 \leftarrow$ `sortie(`I_2^-`)`

2 **si** $config^- \neq 2$ **alors**
  si $i = 1$ **alors** $\gamma \leftarrow \gamma \cup \{(s_1, s_2)\}$
  sinon $\gamma \leftarrow \gamma \cup \{(s_2, s_1)\}$

3 $seg_1 \leftarrow s_1$
si $config^+ = 1$ **alors**
  $s_1 \leftarrow \alpha_1(\alpha_0(s_1))$
  $s_2 \leftarrow$ `ÉclaterArête(entrée(`I_2^-`), Point(`b_a`))`
  si $pos_2 =$ `BORD_INVERSE` **alors** `entrée(`I_2^-`)` $\leftarrow s_2$
sinon
  si $config^+ = 2$ **alors** $s_1 \leftarrow$ `ÉclaterArête(`s_1`, Point(`I_2^+`))`
  sinon si $config^+ = 3$ **alors** $s_1 \leftarrow \alpha_1(\alpha_0(s_1))$
  si `position(`I_2^+`)` $=$ `INTERIEUR` **alors** $s_2 \leftarrow$ `entrée(`I_2^+`)`
  sinon $s_2 \leftarrow$ `sortie(`I_2^+`)`

4 $seg_2 \leftarrow$ `entrée(`I_2^-`)`
si $i = 1$ **alors**
  $\gamma \leftarrow \gamma \cup \{(s_1, s_2)\}$
  $\omega \leftarrow \omega \cup \{(seg_1, seg_2)\}$
sinon
  $\gamma \leftarrow \gamma \cup \{(s_2, s_1)\}$
  $\omega \leftarrow \omega \cup \{(seg_2, seg_1)\}$
si `EstMarqué(`seg_1, $\textcircled{f}$`)` **alors** `Démarquer(`$<\alpha_0, \alpha_1, \alpha_3>(seg_1)$, $\textcircled{f}$`)`
si `EstMarqué(`seg_2, $\textcircled{f}$`)` **alors** `Démarquer(`$<\alpha_0, \alpha_1, \alpha_3>(seg_2)$, $\textcircled{f}$`)`

fin

4. Pour finir, nous rajoutons aux ensembles γ et ω les sommets et segments d'intersection générés pendant cette étape. Si les segments d'intersection trouvés correspondent à des arêtes fictives, ceux-ci sont démarqués de la marque Ⓕ.

ALGO. 7.18 – Fonction `IntersectionFacesCoplanaires` – sous-code n°2

```
    pour i ∈ [1,2] faire
        marquer un brin sur deux de l'orbite <α1,α2,α3>(s_i) avec Ⓣ
1       pour chaque brin b_s ∈ <α1,α2,α3>(s_i) marqué par Ⓣ faire
            Démarquer(<α1,α3>(b_s), Ⓣ)
2           si b_s ∉ ω ∪ Ω alors
                b_a ← b_s
                v ← Vecteur(b_a)
                sect1 ← TrouverSecteur(s_(3−i), P_(3−i), Ⓜ, v)
3               si sect1 ≠ nil alors
                    répéter
                        seg1 ← b_a
                        b_a ← α1(α0(b_a))
a                       si b_a ∈ γ alors
                            sect2 ← sommet correspondant à b_a dans γ
                            sect2 ← TrouverSecteur(sect2, P_(3−i), Ⓜ, −v)
                            seg2 ← ÉclaterFace(sect1, sect2)
b                       sinon
                            seg2 ← InsérerArête(sect1, Point(b_a))
                            sect1 ← α1(α0(sect1))
                            v ← Vecteur(b_a)
c                       si EstMarqué(seg1, Ⓕ) alors Démarquer(<α0,α2,α3>(seg1), Ⓕ)
                        si i = 1 alors ω ← ω ∪ {(seg1, seg2)}
                        sinon ω ← ω ∪ {(seg2, seg1)}
                    jusqu'à b_a ∈ γ
```

L'algorithme 7.18 décrit la manière de procéder pour générer les segments de coupe entre les points d'intersection. Pour chaque point d'intersection, son but est de parcourir les arêtes des deux faces qui lui sont incidentes jusqu'à atteindre un autre point d'intersection. Pour chaque sommet correspondant au point d'intersection, l'algorithme commence par orienter ce dernier à l'aide d'une marque Ⓣ. Ceci est nécessaire afin d'éclater les faces entre deux brins orientés de la même manière. Les traitements effectués après sont alors les suivants :

1. Nous effectuons une boucle permettant de traiter chaque arête marquée par Ⓣ incidente au sommet courant. Pour ne pas traiter plusieurs fois les mêmes arêtes, nous démarquons la première moitié de l'arête de cette marque.
2. Si l'arête n'est pas déjà un segment de coupe (*i.e.* elle n'appartient ni à ω, ni à Ω) nous devons la traiter et tester si elle se trouve à l'intérieur de l'autre face. Pour cela, nous recherchons un secteur angulaire $sect_1$ sur le sommet de l'autre face qui correspond au même point d'intersection et qui contient l'arête courante.

3. Si l'arête courante se trouve à l'intérieur d'un secteur angulaire de la seconde face, nous pouvons alors générer les segments de coupe depuis cette arête et ce jusqu'à atteindre un prochain point d'intersection. Nous utilisons alors une boucle qui effectue les traitements suivants pour chaque arête parcourue :
   a. Si le second sommet de l'arête courante correspond à un point d'intersection, nous effectuons le même traitement que pour le premier sommet de l'arête ; nous récupérons le sommet de la deuxième face correspondant au même point d'intersection puis nous recherchons un secteur angulaire $sect_2$ incident à ce sommet et contenant l'arête courante. Notons ici que l'arête doit obligatoirement appartenir à un secteur angulaire de la deuxième face. Pour finir, nous relions les secteurs $sect_1$ et $sect_2$ à l'aide d'une arête qui correspond à un segment de coupe.
   b. Si le second sommet de l'arête courante ne correspond pas à un point d'intersection, nous insérons alors simplement une arête dans la seconde face. Nous mettons ensuite à jour le secteur $sect_1$ pour qu'il corresponde à l'extrémité de cette arête afin de continuer le parcours.
   c. Finalement, nous démarquons l'arête courante de la marque $\textcircled{f}$ dans le cas où il s'agirait d'une arête fictive, et nous l'insérons dans l'ensemble des segments de coupe ω.

7.4.7 Mise à jour de la topologie

Comme nous l'avons vu précédemment en section 7.3.5, la mise à jour de la topologie consiste à trier angulairement les faces incidentes aux segments de coupe afin de générer une subdivision de l'espace valide. La technique utilisée ici est exactement la même que celle utilisée pour le co-raffinement 2D (cf. section 3.2.1.3). Nous ne donnons donc pas l'algorithme correspondant car il est en tout point identique à l'algorithme 3.8 excepté que les vecteurs utilisés lors du classement sont les normales aux faces et que les éléments à coudre ne sont plus des arêtes mais des faces. Cet algorithme est ensuite exécuté sur chaque couple de brins contenus dans l'ensemble des segments de coupe Ω.

7.4.8 Post-traitements

Plusieurs post-traitements sont nécessaires à divers endroits de l'algorithme. Ces post-traitements visent à supprimer des objets les cellules inutiles et gênantes. En particulier, les cellules à supprimer sont les arêtes fictives et les faces doubles.

7.4.8.1 Suppression des arêtes fictives

Comme nous l'avons vu en section 7.3.3.1, il est nécessaire de supprimer les arêtes fictives inutiles après chaque intersection. L'algorithme 7.19 décrit les traitements à effectuer et doit être appelé à la fin de chaque intersection entre deux faces (*i.e.* fonction `IntersectionFaces` de l'algorithme 7.7). Cet algorithme fait appel à une fonction `FusionnerFaces` qui permet de supprimer une arête servant d'interface entre deux faces et de mettre à jour les liaisons α_1 des sommets auxquels elle est reliée.

ALGO. 7.19 – Procédure `SupprimerArêtesFictives`

Entrées : Un brin b_f d'une face F.
Résultat : Les arêtes fictives inutiles de la face F sont supprimées.

début
  $b \leftarrow b_f$
  répéter
    si `EstMarqué`(b, Ⓕ) **et** `SurOrbite`($\alpha_2(b)$, $<\alpha_0, \alpha_1>(b)$) = `faux` **alors**
      $b' \leftarrow \alpha_1(\alpha_2(b))$
      `FusionnerFaces`(b, $\alpha_2(b)$)
      $b \leftarrow b'$
    sinon
      $b \leftarrow \alpha_1(\alpha_0(b))$
  jusqu'à $b = b_f$
fin

7.4.8.2 Suppression des faces doubles

ALGO. 7.20 – Procédure `SupprimerFaceDouble`

Entrées : Un brin b_f d'une face F.
Résultat : La face F est supprimée et la topologie des arêtes incidentes est mise à jour.

début
  pour chaque brin $b \in <\alpha_0, \alpha_1>(b_f)$ **faire**
    $b_1 \leftarrow \alpha_2(b)$
    $b_2 \leftarrow \alpha_2(\alpha_3(b))$
    `Découdre`(b, α_2)
    `Découdre`($\alpha_3(b)$, α_2)
    `Coudre`(b_1, b_2, α_2)
    supprimer les brins b et $\alpha_3(b)$
fin

La suppression d'une face double est une opération simple à effectuer. Elle consiste uniquement à découdre toutes les arêtes de la face par α_2 puis de recoudre les brins auquels elles étaient reliées. L'algorithme 7.20 donne les traitements à effectuer et est alors exécuter à la fin de l'algorithme principal sur toutes les faces marquées par Ⓓ.

7.5 Résultats

Nous présentons ici quelques résultats obtenus avec l'algorithme décrit dans ce chapitre. Comme pour le précédent algorithme de co-raffinement 3D, nous commençons par donner une estimation de la résolution de la grille régulière utilisée pour réduire la taille des listes de faces dont nous devons calculer les intersections. Ensuite, pour une résolution de grille fixée, nous

$\#F$	t_i	t_l	t_r	t_c	t_m	t_t
60	25 ms	23 ms	138 ms	112 ms	36 ms	355 ms
220	53 ms	59 ms	107 ms	188 ms	57 ms	492 ms
480	103 ms	89 ms	103 ms	330 ms	78 ms	761 ms
840	190 ms	223 ms	82 ms	370 ms	90 ms	1 s 11 ms
1300	263 ms	220 ms	82 ms	493 ms	112 ms	1 s 250 ms
1860	375 ms	316 ms	86 ms	578 ms	135 ms	1 s 620 ms
2520	510 ms	424 ms	85 ms	660 ms	158 ms	1 s 985 ms
3280	709 ms	557 ms	88 ms	783 ms	180 ms	2 s 505 ms
4140	847 ms	695 ms	94 ms	900 ms	207 ms	2 s 973 ms
5100	1 s 45 ms	861 ms	108 ms	967 ms	230 ms	3 s 498 ms
6160	1 s 263 ms	1 s 42 ms	111 ms	1 s 51 ms	256 ms	4 s 61 ms
7320	1 s 506 ms	1 s 236 ms	116 ms	1 s 223 ms	276 ms	4 s 753 ms
8580	1 s 775 ms	1 s 444 ms	125 ms	1 s 303 ms	308 ms	5 s 413 ms
9940	2 s 43 ms	1 s 904 ms	130 ms	1 s 411 ms	329 ms	6 s 350 ms
11400	2 s 343 ms	1 s 919 ms	139 ms	1 s 501 ms	356 ms	6 s 861 ms
12960	2 s 664 ms	2 s 455 ms	180 ms	1 s 735 ms	395 ms	8 s 145 ms
14620	3 s 20 ms	2 s 480 ms	188 ms	1 s 862 ms	417 ms	8 s 762 ms
16380	3 s 383 ms	2 s 779 ms	195 ms	2 s 10 ms	452 ms	9 s 702 ms
18240	3 s 772 ms	3 s 98 ms	205 ms	2 s 66 ms	481 ms	10 s 601 ms
20200	4 s 352 ms	3 s 529 ms	223 ms	2 s 236 ms	507 ms	11 s 932 ms
20200	11 s 932 ms	11 s 932 ms	11 s 932 ms	11 s 932 ms	11 s 932 ms	11 s 932 ms

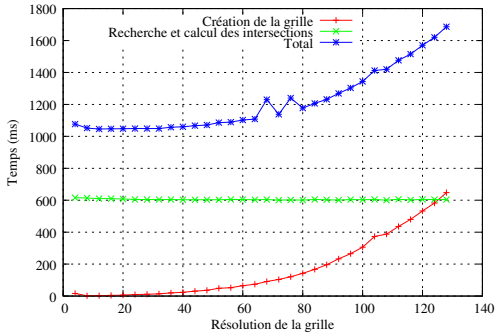
TAB. 7.1 – Temps de calcul obtenus pour le co-raffinement entre deux sphères.

donnons les résultats obtenus sur différents objets. Tous les résultats présentés ici ont été obtenus sur une machine PC dotée d'un processeur Athlon 64 3000+ et de 512 Mo de mémoire vive.

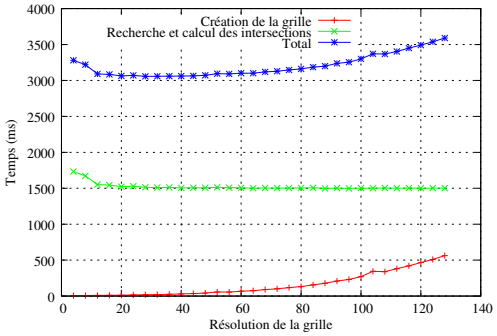
Les tests de performance concernant la détermination d'une résolution optimale pour la grille régulière ont été réalisés sur les mêmes objets que pour le précédent algorithme (*i.e.* un cylindre et une sphère). Nous pouvons observer que les résultats obtenus sur les courbes de la figure 7.24 sont similaires à ceux du précédent algorithme. Pour les tests suivants, nous avons donc utilisé la même résolution de grille, c'est-à-dire 64^3.

Nous présentons maintenant quelques résultats obtenus sur divers objets. Tout d'abord, nous effectuons les mêmes tests de performance que le précédent algorithme à l'aide de deux sphères (cf. section 6.4). Nous obtenons ainsi les résultats du tableau 7.1 et les courbes de la figure 7.25. Les colonnes du tableau correspondent aux données suivantes :

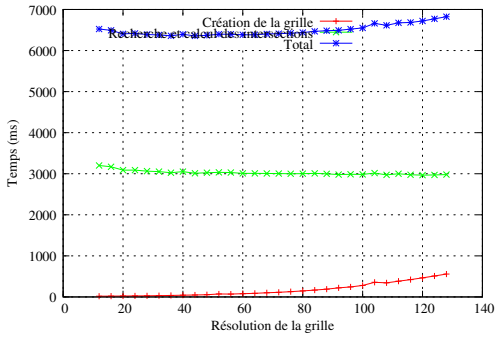
$\#F$ Nombre de faces sur chaque sphère.
t_i Durée de l'initialisation des objets.
t_l Durée de la création des listes de faces.
t_r Durée de la réduction des listes de faces.
t_l Durée de la recherche et de la création des intersections.
t_m Durée de la mise à jour de la topologie.
t_t Temps de calcul total.



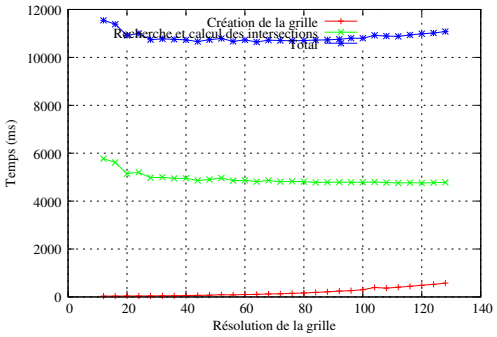
(a) sphère de 840 faces, cylindre de 240 faces



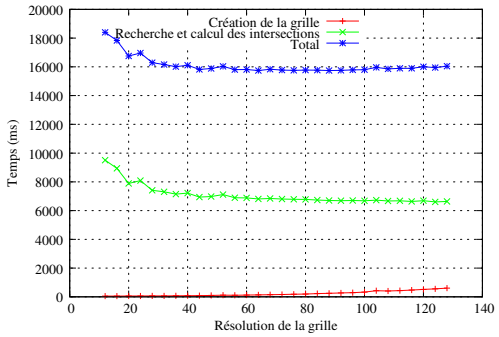
(b) sphère de 3280 faces, cylindre de 880 faces



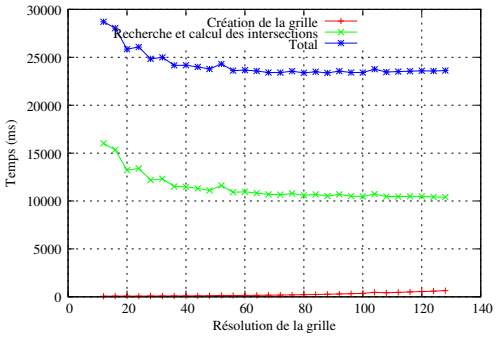
(c) sphère de 7320 faces, cylindre de 1920 faces



(d) sphère de 12960 faces, cylindre de 3360 faces



(e) sphère de 20200 faces, cylindre de 5200 faces



(f) sphère de 29040 faces, cylindre de 7440 faces

FIG. 7.24 – Détermination d'une résolution optimale pour la grille utilisée lors du co-raffinement d'un cylindre et d'une sphère.

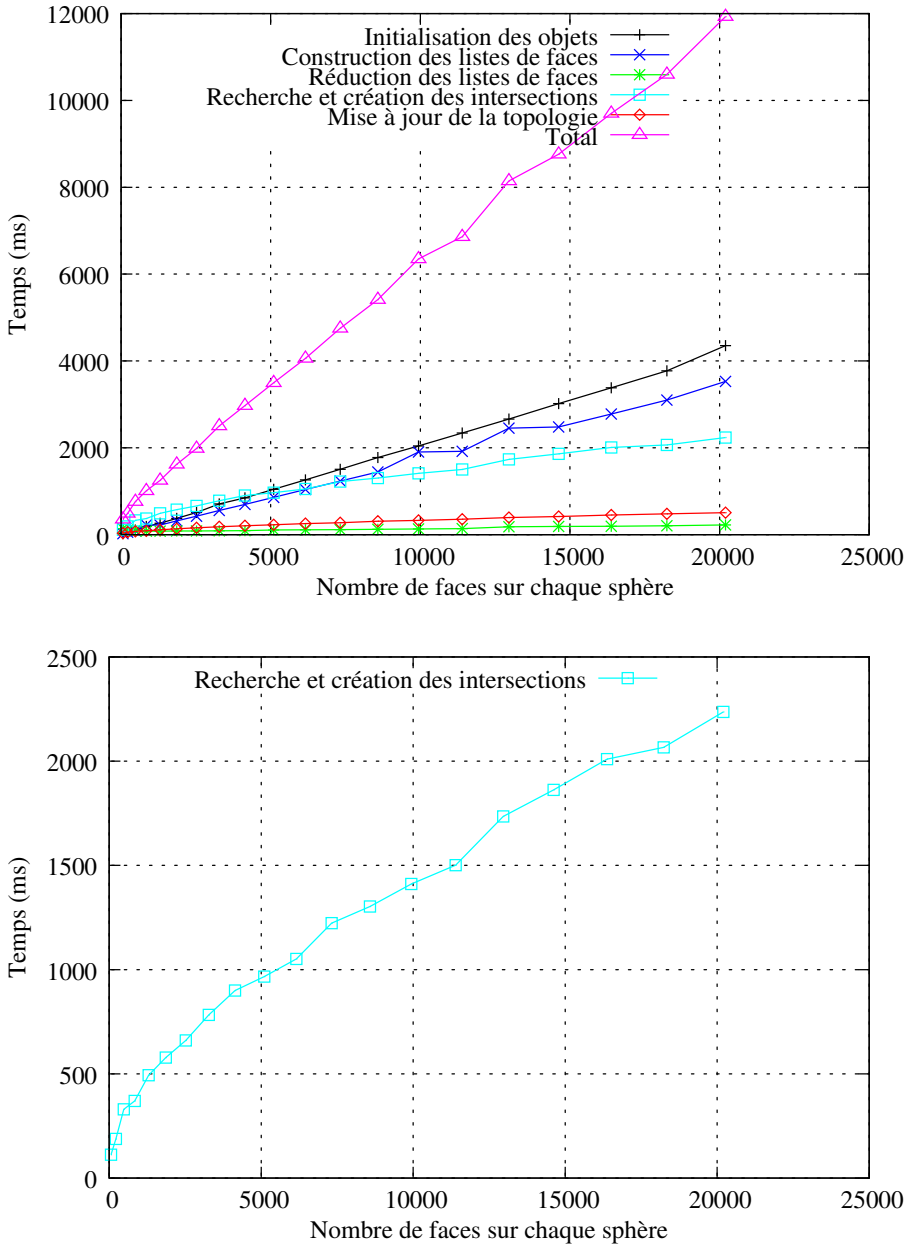


FIG. 7.25 – Évolution des temps de calcul pour le co-raffinement entre deux sphères dont le nombre de faces varie, en utilisant une grille régulière pour réduire le nombre de faces à traiter.

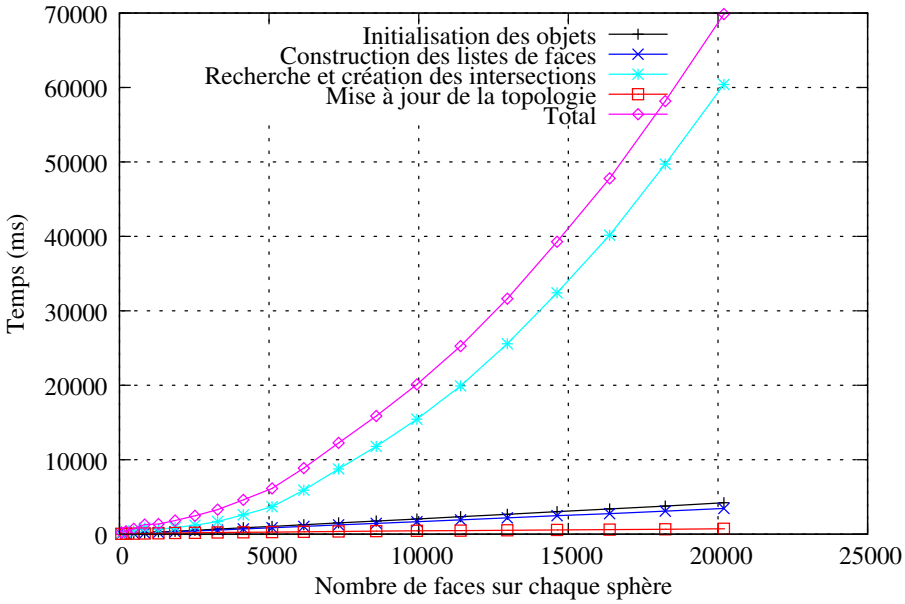
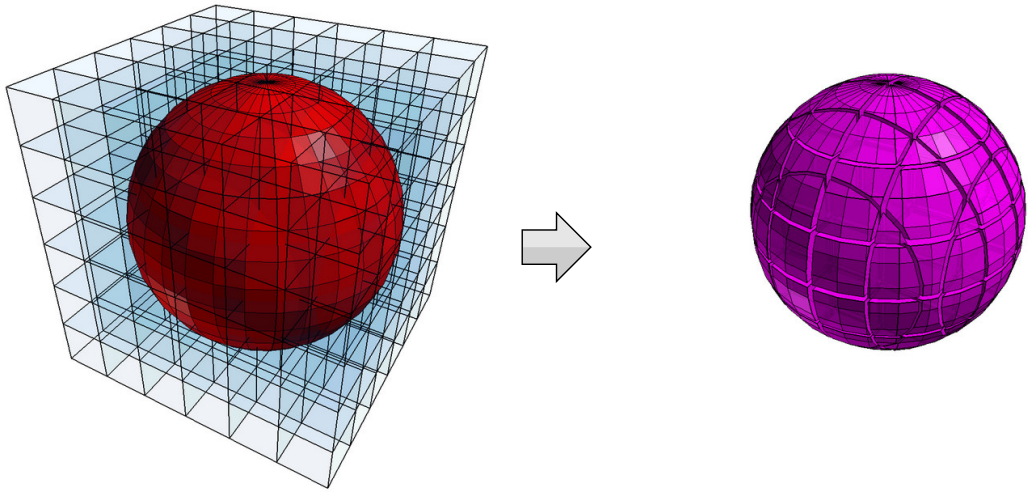


FIG. 7.26 – Évolution des temps de calcul pour le co-raffinement entre deux sphères dont le nombre de faces varie, sans utiliser de grille régulière.

Nous pouvons observer sur ces résultats que les temps de calcul sont nettement inférieurs à ceux du précédent algorithme. En effet, nous pouvons noter que le temps de calcul obtenu pour des spheres de 20 000 faces est passé d'approximativement 18 secondes à seulement 12 secondes. De plus, ce gain est surtout visible pour la recherche et la création des intersections pour lesquelles le temps a été réduit de deux à trois fois.

Afin d'estimer le gain de performance obtenu à l'aide de la grille régulière, nous avons aussi effectué le même test mais sans cette dernière. Nous obtenons alors les courbes de la figure 7.26 sur lesquelles nous pouvons observer que l'algorithme de base qui consiste à calculer l'intersection entre chaque face du premier objet et chaque face du second possède bien une complexité en $\mathcal{O}(n^2)$.

Pour continuer, nous effectuons maintenant des tests similaires à ceux de l'algorithme 2D (cf. section 3.3). Nous commençons par calculer le co-raffinement entre une sphere et un maillage dont le nombre de cellules varie (cf. FIG. 7.27). Nous obtenons alors le tableau 7.2 ainsi que les courbes de la figure 7.28. Nous pouvons observer sur ces courbes que les résultats obtenus sont relativement satisfaisants car l'évolution du temps de calcul tend à être linéaire en fonction du nombre de faces se trouvant dans le maillage.



(a) (b)

FIG. 7.27 – Une sphère et un maillage utilisés pour tester les performances de l'algorithme de co-raffinement 3D : (a) objets originaux ; (b) résultat correspondant à l'intersection des deux objets.

$\#F$	t_i	t_l	t_r	t_c	t_m	t_t
36	52 ms	45 ms	68 ms	801 ms	74 ms	1 s 58 ms
240	69 ms	60 ms	93 ms	1 s 496 ms	279 ms	2 s 31 ms
756	123 ms	108 ms	92 ms	2 s 550 ms	556 ms	3 s 492 ms
1728	216 ms	192 ms	160 ms	4 s 146 ms	847 ms	5 s 664 ms
3300	371 ms	327 ms	124 ms	6 s 644 ms	1 s 322 ms	8 s 955 ms
5616	597 ms	527 ms	162 ms	8 s 993 ms	1 s 733 ms	12 s 258 ms
8820	905 ms	810 ms	165 ms	12 s 499 ms	2 s 237 ms	16 s 968 ms
13056	1 s 319 ms	1 s 179 ms	371 ms	15 s 993 ms	2 s 808 ms	22 s 154 ms
18468	1 s 834 ms	1 s 664 ms	219 ms	21 s 894 ms	3 s 504 ms	29 s 770 ms
25200	2 s 501 ms	2 s 240 ms	275 ms	27 s 57 ms	4 s 91 ms	37 s 15 ms

TAB. 7.2 – Temps de calcul obtenus pour le co-raffinement entre une sphère et un maillage dont le nombre de cellules varie.

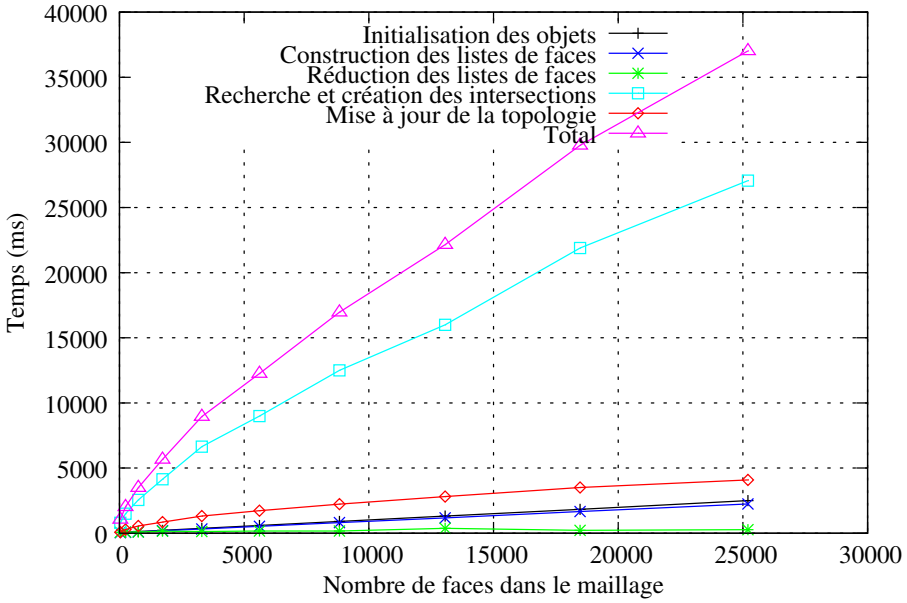
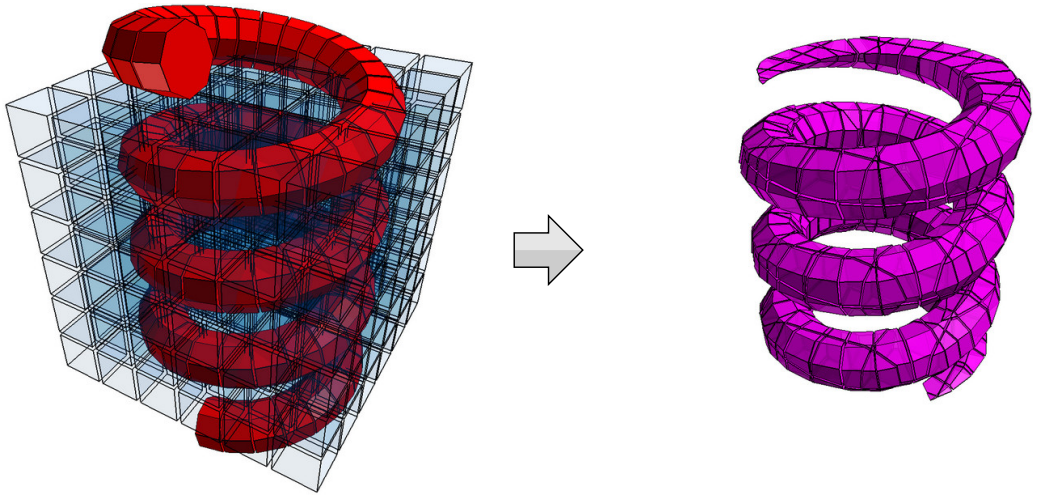


FIG. 7.28 – Évolution des temps de calcul pour le co-raffinement entre une sphère et un maillage dont le nombre de cellules varie.



(a) (b)

FIG. 7.29 – Un chemin 3D et un maillage utilisés pour tester les performances de l'algorithme de co-raffinement 3D : (a) objets originaux ; (b) résultat correspondant à l'intersection des deux objets.

Nous effectuons ensuite le co-raffinement entre un chemin maillé à section circulaire et un maillage dont le nombre de cellules varie (cf. FIG. 7.31). Nous obtenons alors les résultats se trouvant dans le tableau 7.3 et modélisés sur les courbes de la figure 7.30. Nous pouvons observer que les résultats obtenus sont similaires aux précédents et sont donc tout à fait satisfaisants.

$\#F$	t_i	t_l	t_r	t_c	t_m	t_t
36	106 ms	86 ms	49 ms	852 ms	237 ms	1 s 369 ms
240	125 ms	100 ms	56 ms	4 s 372 ms	932 ms	5 s 662 ms
756	176 ms	147 ms	63 ms	6 s 454 ms	1 s 391 ms	8 s 348 ms
1728	269 ms	234 ms	72 ms	12 s 497 ms	2 s 324 ms	15 s 583 ms
3300	426 ms	371 ms	83 ms	19 s 760 ms	3 s 291 ms	24 s 203 ms
5616	651 ms	573 ms	94 ms	26 s 943 ms	4 s 322 ms	32 s 962 ms
8820	963 ms	858 ms	111 ms	39 s 108 ms	5 s 518 ms	47 s 76 ms
13056	1 s 379 ms	1 s 230 ms	139 ms	48 s 581 ms	6 s 659 ms	58 s 668 ms
18468	1 s 906 ms	1 s 721 ms	148 ms	55 s 165 ms	7 s 885 ms	1 m 7 s
25200	2 s 580 ms	2 s 303 ms	169 ms	1 m 22 s	10 s 101 ms	1 m 38 s

TAB. 7.3 – Temps de calcul obtenus pour le co-raffinement entre un chemin 3D et un maillage dont le nombre de cellules varie.

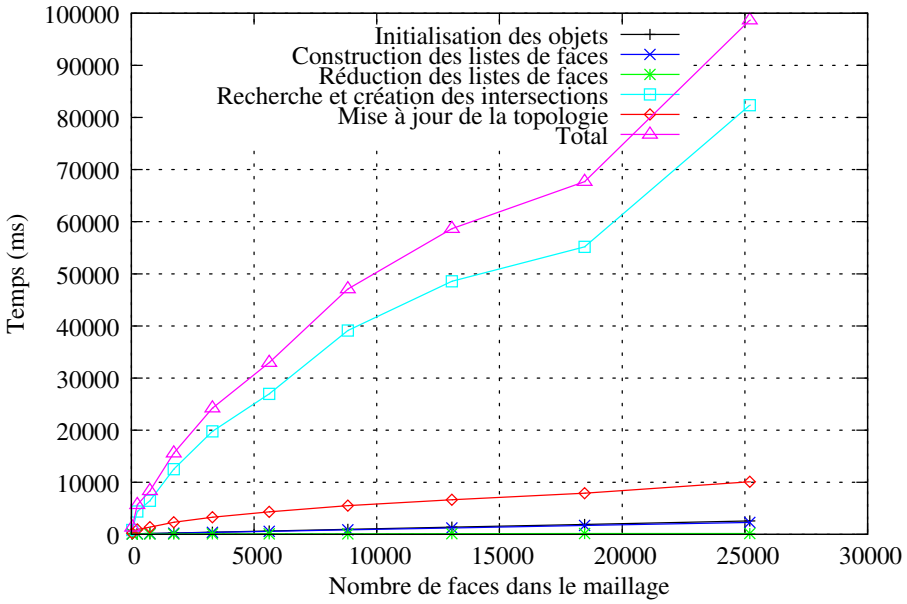
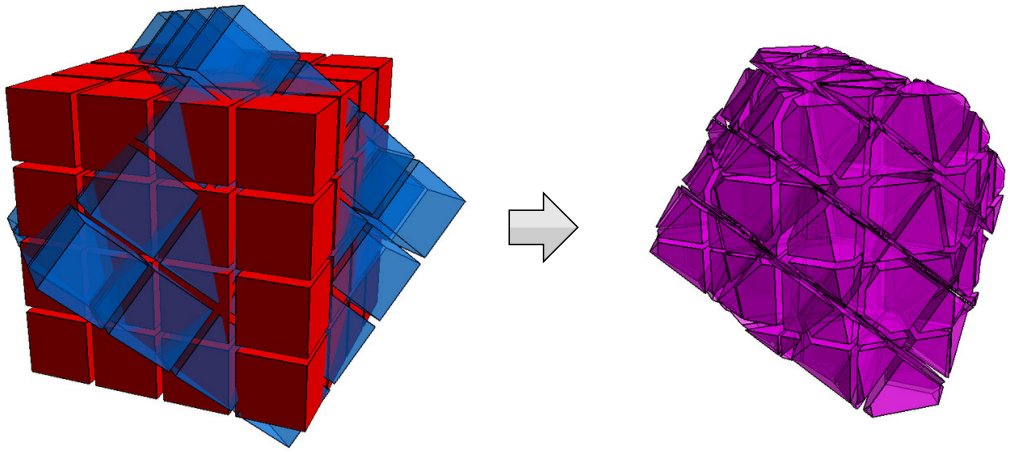


FIG. 7.30 – Évolution des temps de calcul pour le co-raffinement entre un chemin 3D et un maillage dont le nombre de cellules varie.



(a) (b)

FIG. 7.31 – Deux maillages utilisés pour tester les performances de l'algorithme de co-raffinement 3D : (a) objets originaux ; (b) résultat correspondant à l'intersection des deux objets.

$\#F$	t_i	t_l	t_r	t_c	t_m	t_t
36	6 ms	5 ms	$<$ 1 ms	1 s 159 ms	129 ms	1 s 306 ms
240	42 ms	36 ms	164 ms	8 s 813 ms	998 ms	10 s 105 ms
756	143 ms	127 ms	185 ms	29 s 852 ms	3 s 326 ms	33 s 809 ms
1728	333 ms	296 ms	198 ms	1 m 11 s	7 s 817 ms	1 m 20 s
3300	639 ms	574 ms	227 ms	2 m 25 s	15 s 194 ms	2 m 43 s
5616	1 s 88 ms	982 ms	263 ms	4 m 27 s	26 s 135 ms	4 m 57 s
8820	1 s 724 ms	1 s 548 ms	301 ms	7 m 38 s	41 s 392 ms	8 m 26 s
13056	2 s 541 ms	2 s 281 ms	349 ms	12 m 29 s	1 m 1 s	13 m 39 s
18468	3 s 607 ms	3 s 248 ms	401 ms	19 m 50 s	1 m 27 s	21 m 29 s
25200	4 s 891 ms	4 s 411 ms	460 ms	30 m 48 s	1 m 59 s	33 m 3 s

TAB. 7.4 – Temps de calcul obtenus pour le co-raffinement entre deux maillages dont le nombre de cellules varie.

Pour finir, nous réalisons le co-raffinement entre deux maillages identiques tournés l'un par rapport à l'autre et dont nous faisons varier le nombre de cellules. Nous obtenons alors les résultats se trouvant dans le tableau 7.4 et modélisés sur les courbes de la figure 7.32. Nous pouvons observer sur ces derniers résultats que les courbes ont changé de tendance et prennent maintenant une forme parabolique. De plus, nous pouvons remarquer que les temps de calcul obtenus sont très important. Ces deux phénomènes sont dûs au fait que la croissance des maillages est identique et par conséquent la quasi-totalité des faces des maillages possède une intersection avec d'autres faces. Ainsi, la taille des listes de faces n'est presque pas réduite et nous tendons alors vers un algorithme dont la complexité est $\mathcal{O}(n^2)$.

Nous présentons de nombreux autres exemples d'utilisation du co-raffinement et des opérations booléennes en annexe A.

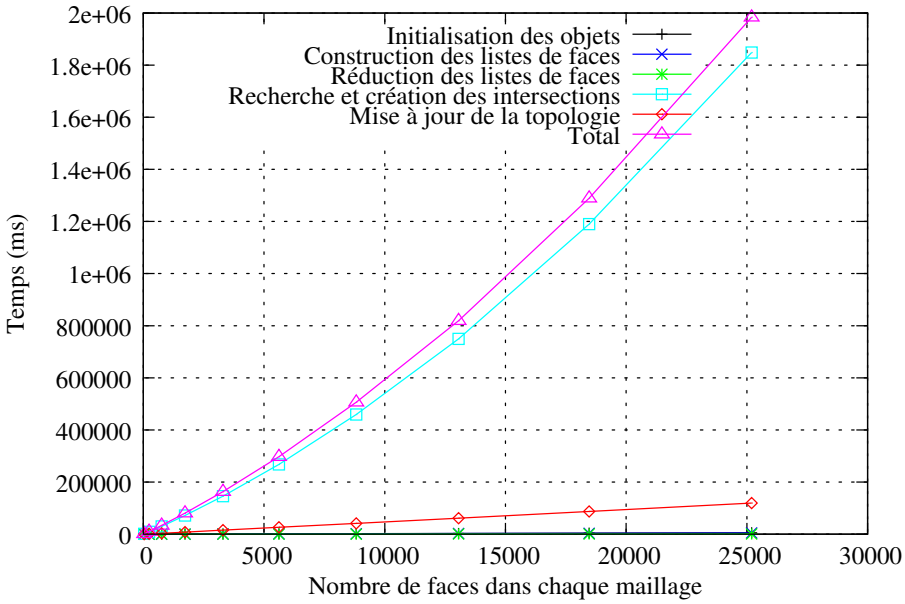


FIG. 7.32 – Évolution des temps de calcul pour le co-raffinement entre deux maillages dont le nombre de cellules varie.

7.6 Conclusion

Nous avons présenté ici un algorithme de co-raffinement 3D dont le but est de calculer tous les segments de coupe entre deux faces en un seul traitement. Cette méthode nous a permis de résoudre tous les problèmes posés par le précédent algorithme ainsi que de diminuer les temps de calculs.

Cependant, l'algorithme peut encore être optimisé en accélérant la recherche des intersections. Nous avons vu en section 7.3.1 que cette dernière était effectuée en éliminant un très grand nombre de faces à l'aide d'une grille régulière. Cependant, nous avons observer que cette réduction n'était pas suffisante dans le cas de maillages denses (cf. section 7.5). Une optimisation possible consisterait à ne tester les intersections qu'entre les faces chevauchant une même case de la grille régulière. Ceci impliquerait alors d'exécuter la boucle principale non plus sur deux listes globales de faces mais sur autant de listes locales que la grille comporte de cases.

Troisième partie

Utilisation du co-raffinement pour la la modélisation géologique

Chapitre 8

Introduction à la modélisation géologique 3D

La modélisation géologique vise à représenter de manière informatique la structure du sous-sol. La représentation informatique d'une scène géologique est alors appelé **modèle géologique** et est le résultat d'un processus de reconstruction complexe.

La modélisation de scènes géologiques 3D date des années 80 et a été rendue possible grâce aux progrès effectués dans le domaine de la Conception Assistée par Ordinateur ainsi que de l'informatique graphique en général. Depuis, des progrès considérables ont été faits dans ce domaine et les modèles géologiques 3D sont devenus des outils incontournables pour l'exploration pétrolière.

8.1 Les éléments d'une scène géologique

Une scène géologique est le résultat d'une succession d'événements géologiques. Dans le cas des réservoirs pétroliers, la scène se compose généralement de différentes couches s'étant déposées les unes sur les autres au fil du temps par un phénomène de sédimentation. Ces couches sont appelées **formations géologiques** et peuvent être différenciées les unes des autres à l'aide de leurs caractéristiques pétrologiques. Ces formations sont alors bornées par des surfaces (aussi appelées **horizons**) qui constituent autant d'interfaces entre chacune d'entre elles.

Au cours du temps, les formations géologiques peuvent subir des déformations qui sont susceptibles de modifier leur forme ou d'interrompre leur continuité suite à l'apparition de **failles**. Les formations sont alors découpées en plusieurs fragments appelés **blocs géologiques** qui peuvent glisser les uns sur les autres le long des surfaces de failles et provoquer ainsi des décalages dans les formations.

La figure 8.1 montre un exemple de scène géologique composée de plusieurs formations. Les formations sont représentées à l'aide de différents niveaux de gris correspondant à leur âge respectif. De plus, chacune d'entre elles est identifiée à l'aide de la surface définissant sa limite supérieure. Nous pouvons aussi voir dans cet exemple deux failles F_1 et F_2 venant découper les formations en plusieurs blocs.

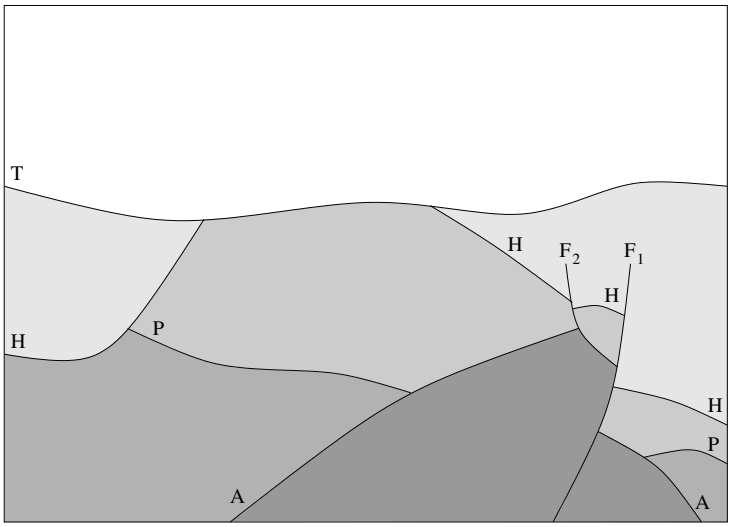


FIG. 8.1 – Exemple de scène géologique 2D. Les blocs ayant la même teinte appartiennent à une même formation et plus le niveau de gris de celle-ci est foncé, plus elle est ancienne.

8.2 Modélisation de scènes géologiques [RPB05]

Il existe de nombreuses méthodes de réprésentation de scènes géologiques qui dépendent du résultat que l'on veut obtenir ainsi que de la manière de l'exploiter. Dans le domaine de l'exploration pétrolière et gazière, les modèles de bassin peuvent représenter des scènes comprises entre 10 et 100 km de long. Ces modèles peuvent être statiques (dans le but de visualiser la structure géologique) ou bien dynamiques (dans le but de pouvoir identifier et quantifier des générations et des migrations d'hydrocarbures).

Il existe aussi des modèles de réservoir qui se limitent à une centaine de mètres de long. Ces modèles peuvent représenter les relations entre les différentes surfaces géologiques (modèles structuraux), des arrangements stratigraphiques et les compositions pétrologiques (modèles stratigraphiques), des propriétés pétrophysiques (modèle de réservoir complet permettant la simulation d'écoulements).

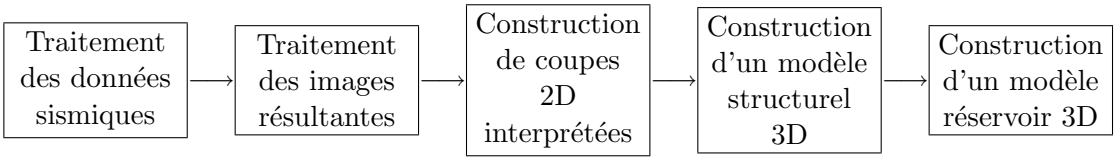


FIG. 8.2 – Chaîne de traitements pour la construction d'un modèle de réservoirs

Quels que soient leur taille et leur but, ces représentations sont la plupart du temps très complexes. Par exemple, les horizons stratigraphiques qui sont représentés dans un modèle structural correspondent à des surfaces qui sont le résultat de nombreux traitements effectués sur des relevés sismiques (cf. FIG. 8.3). De la même manière, ces relevés sismiques sont issus d'un processus complexe de traitement de signaux et de propagation d'ondes. Ainsi, plusieurs

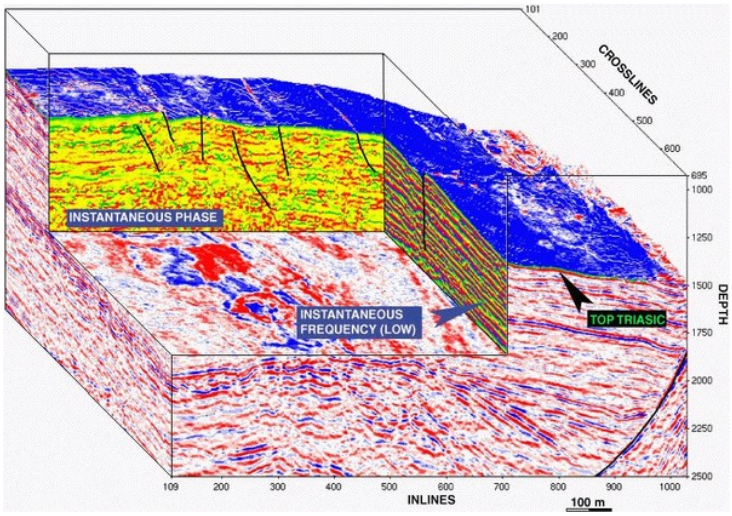


FIG. 8.3 – Exemple de données provenant de relevés sismiques.

représentations successives sont affectées à un même objet durant les différentes phases de traitement (cf. FIG. 8.2).

Durant chaque étape de construction du modèle, la topologie et la géométrie des objets ne dépendent pas seulement de la nature physique de leur représentation mais de l'interprétation du géologue et des nombreux choix qu'il fait. En effet, c'est l'utilisateur qui analyse et fait la distinction entre les surfaces et des artefacts au sein de relevés sismiques. C'est aussi lui qui fournit une interprétation de la structure des failles associées au modèle et qui de ce fait définit les discontinuités qui modifient la topologie des surfaces. Ainsi, il est plus qu'important que l'interprétation de l'utilisateur puissent être conservée et qu'elle reste valide tout au long du processus de construction dans le but de conserver les caractéristiques du modèle.

Chapitre 9

Outils de modélisation existants

9.1 Approches classiques et modeleurs commerciaux

Plusieurs logiciels sont actuellement disponibles sur le marché et proposent des solutions différentes concernant la construction de modèles géologiques. Le point commun entre tous ces logiciels est la participation de l'utilisateur dans le processus de construction. Aucun d'entre eux ne possède de processus automatisé et l'utilisateur doit effectuer toutes les opérations de construction à la main. De plus, la topologie des modèles qu'ils permettent de générer est très souvent directement dépendante des solveurs qui sont utilisés pour effectuer des simulations d'écoulement.

Actuellement, les logiciels GOCAD, ROXAR et PETREL se partagent 90% du marché à eux trois. Les 10% restants sont partagés par d'autres modeleurs, dont la RML et Earth Vision à hauteur de 3% chacun. Nous présentons ici les principales caractéristiques de ces cinq modeleurs.

9.1.1 GOCAD

GOCAD est un modeleur géologique développé depuis 1989 au sein de l'École Nationale Supérieur de Géologie de Nancy [Mal92] et actuellement distribué par la société ED[1]. Celui-ci est basé sur une représentation des modèles à l'aide de surfaces triangulées. L'intérêt principal de ce mode de réprésentation est dû au fait que les surfaces peuvent avoir des formes quelconques et qu'elles peuvent accomoder des discontinuités géométriques complexes permettant de simuler aisément des réseaux de failles.

Couramment, la reconstruction des surfaces est faite à l'aide de l'interpolateur discret DSI (Discrete Smooth Interpolator) [Mal89]. Celui-ci permet d'effectuer de nombreuses opérations sur les surfaces comme des déformations ou bien des extensions. Son intérêt réside dans le fait qu'il peut tenir compte des discontinuités sur les surfaces et est particulièrement bien adapté dans le cas de surfaces découpées par un très grand nombre de failles. Un inconvénient de DSI tient toutefois au fait qu'il s'agit d'un interpolateur global. Ainsi, il n'est pas possible de corriger une imperfection ponctuelle sans intéragir sur la forme d'ensemble de la surface.

[1] `http://www.earthdecision.com/`

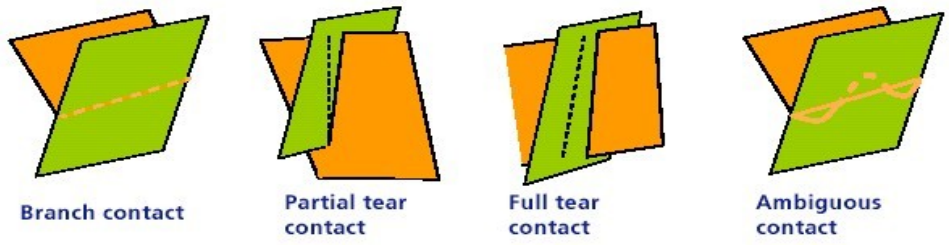


FIG. 9.1 – Les principaux types de contact entre failles proposés par GOCAD.

GOCAD permet de gérer la topologie des modèles et permet ainsi de décrire les relations entre les différents éléments les composant. Cependant, il ne permet pas de stocker l'interprétation géologique du modèle et ne permet donc pas d'indiquer pour quelles raisons les objets géologiques possèdent une intersection. Les seuls éléments pouvant être stockés sont des scripts permettant de décrire l'enchaînement des opérations de construction à effectuer.

L'atout majeur de ce logiciel réside dans son interface utilisateur qui est très conviviale et qui permet de guider l'utilisateur pas à pas dans la construction des modèles. C'est aussi un logiciel complet permettant de générer des maillages 3D à partir des modèles structuraux, dans le but d'effectuer des simulations d'écoulement.

La construction d'un modèle structural passe par plusieurs étapes qui sont entièrement manuelles. Nous nous intéressons ici plus particulièrement à celles permettant de modéliser les intersections entre les différents objets, c'est-à-dire l'intersection entre plusieurs failles et l'intersection entre un horizon et des failles.

L'intersection entre plusieurs failles se fait en plusieurs étapes. Tout d'abord, l'utilisateur doit spécifier manuellement les failles étant isolées et les failles étant en intersection ainsi que la forme des contacts entre celles-ci. Il a alors le choix entre quatre principaux types de contacts qui sont représentés sur la figure 9.1. Notons ici que le choix de l'utilisateur se fait arbitrairement de la géométrie des surfaces et qu'aucune interprétation géologique n'est associé aux contacts entre failles.

Ensuite l'utilisateur peut modifier la géométrie des contacts à la main puis déformer les surfaces des failles pour qu'elles repassent par les nouvelles courbes d'intersection.

De la même manière que pour les failles, l'intersection entre un horizon et des failles se réalise en plusieurs étapes. L'utilisateur doit tout d'abord indiquer quelles sont les failles qui doivent découper l'horizon puis le logiciel calcule les différentes courbes d'intersection. Généralement, comme la géométrie des horizons est assez chaotique au voisinage des failles, les courbes d'intersection peuvent être assez complexes. Il se peut même que les courbes soient déconnectées les unes des autres et que par conséquent la topologie de l'intersection soit différente de celle des réseaux de failles. L'utilisateur a alors la possibilité d'éditer les courbes à la main afin de modifier leur géométrie et leur topologie en les raccordant ou en supprimant certaines d'entre elles. Il peut aussi décaler les courbes afin de faire apparaître un rejet au niveau de la faille. Pour finir, il doit ensuite déformer l'horizon afin que sa surface passe par les nouvelles courbes d'intersection.

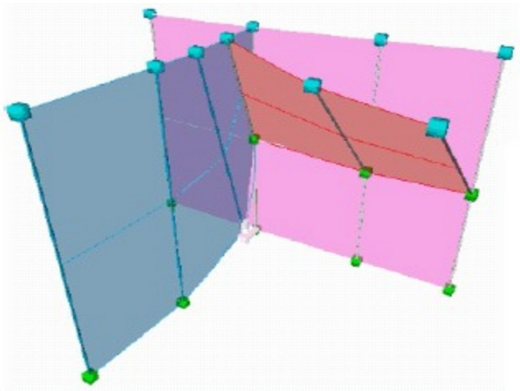


FIG. 9.2 – Exemple de failles modélisées dans PETREL.

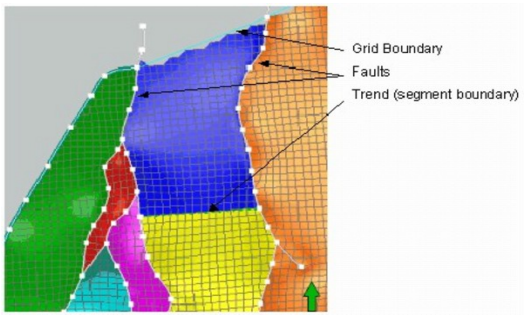


FIG. 9.3 – Exemple de grille générée à partir des failles.

9.1.2 PETREL

PETREL est un modeleur volumique développé par la société Technoguide entre 1998 et 2003 et racheté par Schlumberger[2], qui le commercialise actuellement. Contrairement aux autres logiciels, il ne permet pas de construire des modèles structuraux surfaciques. Son objectif est de construire directement des maillages 3D pour les simulations.

La construction d'un maillage est entièrement manuelle et se fait en plusieurs étapes qui sont les suivantes :

- **modélisation des failles :** définition des failles qui vont définir la structure générale de la grille. Les failles sont généralement des surfaces pseudo-verticales définissant les piliers principaux de la grille. Elles sont raccordées topologiquement les unes aux autres et découpent ainsi la grille en plusieurs régions (cf. FIG. 9.2)

- **création des piliers de la grille :** génération de tous les piliers de la grille en fonction des piliers principaux définis par les réseaux de failles (cf. FIG. 9.3). La grille peut être déduite automatiquement de la géométrie ou bien peut être définie manuellement par l'utilisateur. Celui-ci peut rajouter des segments de contrôle permettant de forcer le passage de la grille à certains endroits.

- **création des horizons :** intersection des surfaces des horizons avec les piliers de la grille obtenue à l'étape précédente. L'intersection entre chaque pilier et les horizons définissent les noeuds des grilles correspondant aux interfaces entre les différentes formations. Au voisinage des failles, une zone est définie par l'utilisateur afin de supprimer toute la géométrie des horizons compris dans celle-ci. Ensuite, les horizons sont étendus jusqu'à atteindre la surface de la faille.

[2] `http://www.slb.com/`

9.1.3 La « Reservoir Modeling Line »

La RML est une suite de logiciels développée par Beicip-Franlab[3] et GeoMath, au sein de l'Institut Français du Pétrole. Chaque logiciel de cette suite est dédiée à une étape précise de l'étude d'une scène géologique 3D : par exemple le logiciel GeoSurf permet de construire des modèles structuraux surfaciques tandis que les logiciels GeoSim et SimGrid permettent de générer des grilles 3D (ou maillages 3D) afin d'effectuer des simulations d'écoulement.

Les modèles structuraux de la RML sont construits à l'aide de surfaces paramétriques en utilisant les bibliothèques Open CAS.CADE[4] et SisCat de Sintef[5]. Ces bibliothèques permettent de gérer toute la géométrie de la scène et fournissent un très grand nombre d'opérations permettant de modifier aisément la forme des surfaces.

Cependant, l'utilisation de surfaces paramétriques dans la modélisation de scène géologique rend délicate la représentation de certains objets géologiques comme les failles pendantes. Ainsi lorsqu'une faille s'arrête en plein milieu d'un bloc, il est nécessaire de la prolonger afin d'atteindre le bord du modèle et de découper ainsi la totalité du bloc.

La RML permet de construire une macro-topologie en partitions de blocs géologiques. Cette macro-topologie est utilisée pour construire des maillages héxaédriques pour la représentation de modèles stratigraphiques ou de modèles de réservoirs. Néanmoins la construction intéractive de cette macro-topologie peut s'avérer délicate et longue.

Un module venant se greffer sur le logiciel GeoSurf a été développé par Sébastien Schneider pendant sa thèse [Sch02] et permet entre autre de constituer ce type de topologie à partir d'une représentation de l'interprétation géologique (cf. section 9.2.2). Nous revenons sur cette extension en section 9.3.

Dans le logiciel GeoSurf, la construction de modèles structuraux est entièrement manuelle. L'utilisateur dispose alors de plusieurs opérations géométriques lui permettant de traiter les surfaces et est libre d'en faire ce qu'il veut du fait de l'absence totale de topologie. Pour générer l'intersection entre un horizon et une faille, l'utilisateur doit interactivement découper les surfaces à l'aide des outils fournis par le logiciel puis il doit les déformer afin de faire apparaître des rejets.

9.1.4 Earth Vision/Dynamic Graphics

Dynamic Graphics[6] est un éditeur de logiciel pionnier dans le domaine de la géomodélisation. Au départ Dynamic Graphics a été longtemps un des leaders dans le domaine de la cartographie. Ce logiciel exploite le fait que les surfaces géologiques sont généralement monovaluées et peuvent être représentées par des grilles régulières.

Ceci conduit à définir totalement les surfaces des horizons et failles dans un espace rectangulaire et à organiser les intersections entre elles en fonction de règles d'assemblage définies à partir d'une description « topologique » de la géologie. Lorsque l'utilisateur doit modéliser des

[3] `http://www.beicip.com/`
[4] `http://www.opencascade.org/`
[5] `http://www.sintef.no/`
[6] `http://www.dgi.com/`

surfaces multivaluées, Earth Vision propose tout simplement de dupliquer l'information. L'utilisateur obtient des blocs topologiquement attachés entre eux. La qualité de ces modèles est souvent excellente.

Ce logiciel permet ainsi de préparer interactivement la construction des modèles et de la réaliser en « batch ». Le grand intérêt de cette méthode est de pouvoir rejouer assez facilement des constructions de modèles en changeant les règles ou les surfaces utilisées en entrée. C'est d'ailleurs grâce à cette facilité que Dynamic Graphics commence à être utilisé pour remettre à jour des modèles structuraux en cours de forage.

Son grand handicap est de ne pas proposer de méthode de remplissage en propriété pétrophysiques ni de logiciels de simulation d'écoulement. Ceci oblige une connexion avec d'autres logiciels et réduit leurs champs d'action.

9.1.5 IRAP/RMS/Roxar

Le produit IRAP/RMS[7] est issu d'un travail réalisé par l'association SINTEF/GEOMATIC (Norvège) au cours du début des années 90. Le produit a alors été acheté par SMEDVIG, une société de forage concurrente de Schlumberger et Halliburton. Actuellement il est la propriété d'un Groupe Anglo-Norvégien qui a également fait l'acquisition d'autres outils (en simulation de réservoir par exemple) pour couvrir toute la chaîne de traitement de la géologie structurale au réservoir.

Ce fut le premier outil proposant une couverture complète des fonctionnalités de remplissage en propriétés pétrophysiques. C'est donc aujourd'hui celui qui a la base installée la plus importante. Au départ, comme Earth Vision, ce produit est basé sur l'utilisation de grilles régulières. Comme Earth Vision, IRAP/RMS a la possibilité d'utiliser plusieurs surfaces si les horizons sont multivalués.

Afin de rendre le système plus souple, lorsque les failles croisent les horizons, les mailles régulières décrivant les horizons sont découpées en triangles. Avec ces deux fonctionnalités et un traitement de qualité des maillages permettant de résoudre des problèmes difficiles au niveau des failles, ce produit permet de modéliser tous les environnements.

Néanmoins, c'est de tous les modeleurs celui qui est aujourd'hui le plus grossier en ce qui concerne la modélisation structurale (sans être forcément le moins efficace en production) car il bénéficie de l'expérience accumulée par des études répétées.

Actuellement Roxar essaye de pallier ces difficultés pour conserver son marché et prépare une nouvelle génération.

9.2 Nouvelle approche pilotée par les connaissances géologiques

Comme nous l'avons vu au chapitre 8, les données d'entrée permettant de construire un modèle structural 3D sont directement dépendantes de l'interprétation des géologues. De plus,

[7] `http://www.roxar.com`

ces interprétations sont effectuées à tous les niveaux du processus de traitement et permettent au final d'obtenir autant de résultats possibles qu'il y a eu d'interprétations différentes.

Les modeleurs classiques fournissent uniquement une palette d'outils plus ou moins conviviaux qui permettent à l'utilisateur de construire un modèle à la main. Cependant, ces outils ne permettent pas d'associer une interprétation géologique aux relations existantes entre les différents objets composant un modèle. Ainsi, le géologue doit déduire sa propre interprétation à partir de la topologie générée par le logiciel. Par conséquent, il doit faire en sorte de gérer au mieux la cohérence topologique du modèle tout au long de la chaîne de traitement de manière à ce qu'elle corresponde à sa propre interprétation. S'il décide à un moment de modifier son interprétation, il doit alors défaire toutes les opérations de construction dépendante de cette interprétation afin de les effectuer à nouveau.

L'approche pilotée par la connaissance consiste à préserver l'interprétation du géologue à chaque étape de construction d'un modèle. Ceci implique que tous les éléments du modèle doivent être identifiés (horizons, failles...) et que leur géométrie ainsi que leur position dans l'espace ne doivent pas être dépendante de la topologie du modèle, c'est-à-dire de l'interprétation du géologue.

Depuis quelques années, l'École des Mines de Paris, l'Institut Français du Pétrole et l'Université de Poitiers travaillent en collaboration sur une méthode permettant d'automatiser la constrution des modèles géologiques en déduisant leur topologie à partir de l'interprétation du géologue [SPG$^+$04, BSP$^+$04, BSP$^+$05, RPB05]. Cette méthode est basée sur le fait qu'une scène géologique est le résultat d'une suite d'événements successifs dans le temps. Ces événements définissent ainsi un ensemble d'objets géologiques possédant des propriétés différentes, mais étant en relation les uns avec les autres dans le temps et l'espace. Ces relations permettent alors de définir l'ordre des opérations de construction à effectuer et permettent ainsi d'automatiser le processus de traitement.

Dans cette optique, Michel Perrin a défini des règles de syntaxe géologique [Per97, Per98] permettant de décrire les éléments géologiques d'un modèle ainsi que les relations qu'ils entretiennent. Ces règles se décomposent en deux catégories qui sont :

- une description formelle des objets géologiques,
- un formalisme décrivant les relations temporelles ou spatiales entre les objets géologiques.

Dans la suite, nous détaillons tout d'abord les règles de syntaxe permettant de décrire la structure d'un modèle, puis nous expliquons comment les utiliser pour définir une chaîne de traitement automatisée.

9.2.1 Définition des règles de syntaxe géologique

Les règles définies par M. Perrin attribuent et classifient les objets rencontrés en géologie à l'aide d'un formalisme particulier. Ce formalisme a été élaboré en tenant compte des spécificités métiers propres aux objets géologiques, spécificités dont on peut se faire une idée au travers des manuels classiques de géologie structurale [FR88, HMW76]. Nous nous limitons ici au rappel des règles de syntaxe de base énoncées par M. Perrin et nécessaires à la compréhension de la suite de ce manuscrit.

Type géologique	FTR	FTA
Parallèle	CONC	CONC
Érosion (discordance)	CONC	DISC
Onlap	DISC	CONC
Discordance + Onlap	DISC	DISC

TAB. 9.1 – Les différents types d'interfaces géologiques

Les règles de syntaxe caractérisent des types de propriétés dans un modèle géologique. La première se situe au niveau des objets géologiques et décrit le type de sa surface (polarisée ou non) et la manière dont il s'intègre dans une « scène géologique ». La seconde concerne la relation entre deux événements géologiques successifs, c'est-à-dire décrit comment un événement actuel influe temporellement ou structuralement sur son ou ses prédécesseurs.

9.2.1.1 Élements de syntaxe pour la description des événements géologiques

Une surface géologique peut être soit polarisée, soit non polarisée. Dans le premier cas, elle correspond à une **interface géologique** entre deux formations et dans le second cas il s'agit d'une **faille** ou d'un **contact anormal**. Une propriété importante d'une surface géologique, polarisée ou non, est son **âge ponctuel unique**. Cette propriété nous permet de définir l'enchaînement successif des événéments de manière géologiquement cohérente.

Comme deux formations adjacentes possèdent des âges différents, les deux côtés de l'interface géologique les séparant possèdent des propriétés différentes. Cette dernière possède alors une face tournée vers la formation la plus récente (FTR) et l'autre face tournée vers la formation la plus ancienne (FTA). De plus, nous affectons une propriété binaire à chacune des faces : un côté d'interface polarisée est soit concordant, soit discordant (concordant signifiant que cette face ne pourra pas venir recouper une autre surface, discordant signifiant au contraire que cette face pourra recouper des surfaces d'un point de vue géométrique). De cette propriété sur chacune des faces, nous pouvons déduire quatre types différents d'interfaces polarisées : les surfaces parallèles, les surfaces d'érosion (ou discordantes), les surfaces onlap (ou de dépot), et les surfaces à la fois discordantes et onlap.

Le tableau 9.1 décrit ces types de surfaces en donnant, pour chaque couple de propriétés attribuées aux deux faces d'une surface, le type géologique correspondant de la surface. La manière dont le type de la surface influe sur sa relation avec les autres surfaces du modèle est détaillée en section 9.2.1.2.

Contrairement aux surfaces polarisées, les deux faces des surfaces non polarisées sont tournées vers des formations plus anciennes. En effet, les surfaces non polarisées sont des accidents dans des structures géologiques, représentés par des failles ou des contacts anormaux. Par exemple, une faille F peut venir « casser » à un moment t un ensemble de formations géologiques toutes plus anciennes que F, c'est-à-dire qu'elle possède des intersections avec une série d'interfaces ayant chacune un âge ponctuel unique inférieur à t. Un contact anormal est une interface pouvant être générée par une déformation de type chevauchement ; dans ce cas, le contact anormal sépare deux domaines géologiques distincts.

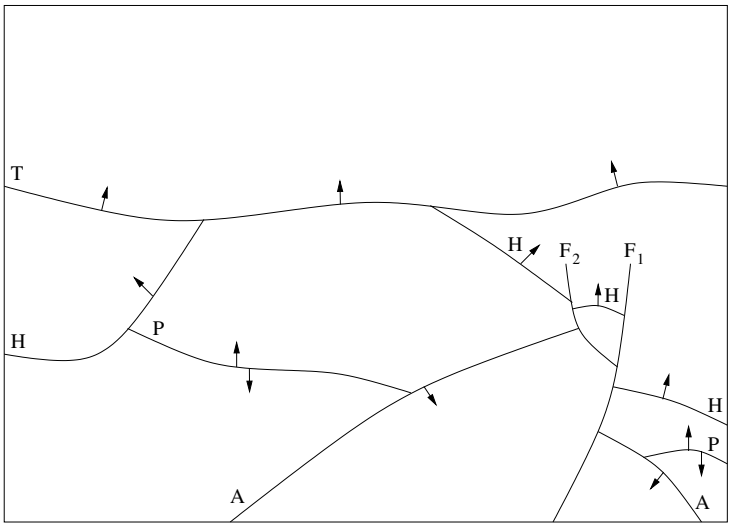


FIG. 9.4 – Exemple d'une scène géologique regroupant les principaux types dévénements

L'exemple de la figure 9.4 montre une scène géologique comportant quatre surfaces polarisées (T et H sont des surfaces d'érosion, P est une surface parallèle et A est une surface onlap) et deux failles F_1 et F_2. Les flèches représentent les côtés concordants des interfaces polarisées.

9.2.1.2 Éléments de syntaxe pour la description des relations entre objets géologiques

Une relation lie deux événements géologiques et l'ensemble des événements et des relations définies à l'aide des règles de syntaxe constitue la structure d'un modèle géologique. Une relation entre deux surfaces géologiques est à la fois une relation chronologique et une relation topologique :

- une relation chronologique car chacune des surfaces a un âge ponctuel unique (cf. section 9.2.1.1), la relation est de type « antérieur à » ou « postérieur à ». Il peut toutefois arriver, dans le cas de surfaces de failles, que l'on ne sache pas établir une différence d'âge entre les deux événements. Dans ce cas, les deux objets sont dits **contemporains** l'un par rapport à l'autre et, s'ils possèdent une intersection, la relation entre eux est de type « s'arrête sur ». Ce type de relation est dit structural. Ces deux types de relations sont représentés par l'intermédiaire d'une liaison entre deux événements géologiques (cf. FIG. 9.5).
- une relation topologique, qui est déduite du type géologique des surfaces (cf. TAB. 9.1) et de leur différence d'âge. Cette relation est appelée « recoupant/recoupé » et permet de savoir, lors de l'intersection entre deux surfaces, laquelle découpe l'autre (cf. TAB. 9.2). Quel que soit le type géologique des surfaces, leur intersection est toujours du type « recoupant/recoupé » car deux interfaces géologiques ne peuvent pas se traverser.

Le tableau 9.2 indique les relations topologiques entre deux événements polarisés. Ce tableau peut être condensé en un tableau 9.3 à deux lignes et deux colonnes qui présente les trois types

FIG. 9.5 – Convention de représentation des deux types de relations : (a) la relation chronologique ; (b) la relation structurale ou « s'arrête sur ».

Ancien(2) \ Récent(1)	Parallèle C/C	Érosion C/D	Onlap D/C	Discordant D/D
Parallèle C/C	IMPOSSIBLE	(1) recoupe (2)	IMPOSSIBLE	(1) recoupe (2)
Érosion C/D	IMPOSSIBLE	(1) recoupe (2)	IMPOSSIBLE	(1) recoupe (2)
Onlap D/C	(2) recoupe (1)	(1) recoupe (2)	(2) recoupe (1)	(1) recoupe (2)
Parallèle D/D	(2) recoupe (1)	(1) recoupe (2)	(2) recoupe (1)	(1) recoupe (2)

TAB. 9.2 – Les relations topologiques de type recoupant/recoupé

de relations topologiques. Seule la face significative de la surface est conservée, et suffit, avec son âge, à déterminer le type de la relation.

La combinaison de la relation chronologique et du type (polarisé ou non) des deux surfaces suffit à déterminer le type de la relation topologique :

- **Relation topologique entre deux événements non contemporains :** dans ce cas, nous avons juste à lire le tableau 9.3 pour trouver le type de topologie entre les deux surfaces. Dans le cas d'une surface non polarisée, il suffit de l'identifier à une surface ayant ses deux faces discordantes dans le tableau.
- **Relation topologique entre deux événements contemporains :** cette relation est particulière, et le seul cas possible intervient entre deux surfaces non polarisées. La relation topologique est alors de type « s'arrête sûr » et elle indique par elle même laquelle des deux surfaces s'arrête sur l'autre.

La figure 9.6 montre des exemples de l'influence de la relation recoupant/recoupé entre deux événements géologiques en fonction de leur type et de leur relation.

		Surface récente (1)	
		Face Concordante	Face Discordante
Surface Ancienne (2)	Face Concordante	IMPOSSIBLE	(1) recoupe (2) (discordance)
	Face Discordante	(2) recoupe (1) (onlap)	

TAB. 9.3 – Les relations « recoupant/recoupées » résumées.

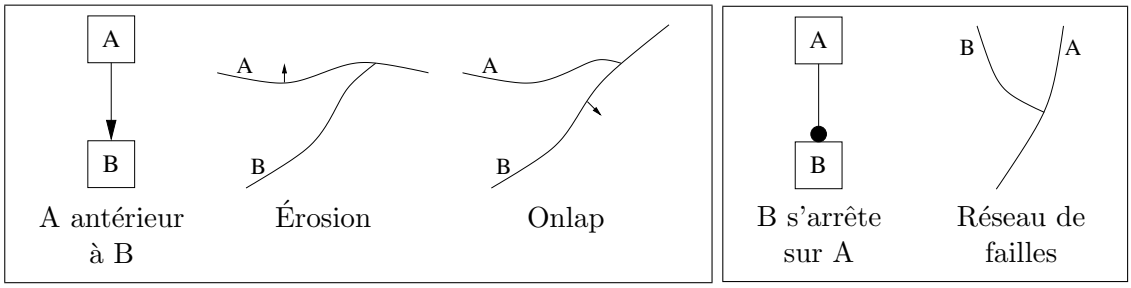


FIG. 9.6 – Exemples de relations recoupant/recoupé : à gauche, une relation chronologique entre deux événements induit le découpage indiqué en fonction du type des surfaces A et B (un manque d'information sur l'orientation d'un côté indique que la propriété géologique peut être soit concordante soit discordante) ; à droite, c'est une relation du type « s'arrête sûr » qui indique une relation structurale entre deux failles contemporaines.

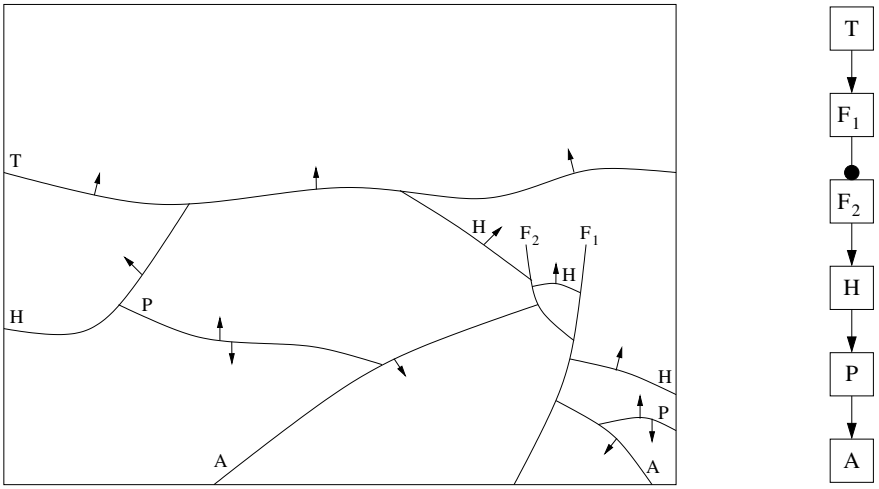


FIG. 9.7 – Une scène géologique et son Schéma d'Évolution Géologique associé.

9.2.2 Le Schéma d'Évolution Géologique (SEG)

À partir des règles de syntaxe décrit précédemment, il est possible de représenter la structure d'une scène géologique à l'aide d'un Schéma d'Évolution Géologique. Le SEG est un graphe orienté dont chaque noeud définit un objet géologique d'une scène (en général une surface représentant un horizon ou une faille) et chaque arc correspond à une relation chronologique ou structurale (cf. section 9.2.1.2). Chaque noeud peut alors avoir plusieurs successeurs et prédécesseurs mais le graphe ne peut posséder aucun circuit fermé. De plus, un SEG peut être hiérarchique et définir plusieurs niveaux de détails. Ainsi, un noeud peut correspondre à un SEG représentant la structure géologique d'un objet de manière plus précise.

La figure 9.7 montre le Schéma d'Évolution Géologique correspondant à la scène géologique donnée en exemple sur la figure 9.4. Nous pouvons observer que la disposition des noeuds se fait en correspondance avec la structure de la scène géologique, à savoir les noeuds les plus récents en haut et les noeuds les plus ancien en bas. De plus, les relations entre les noeuds

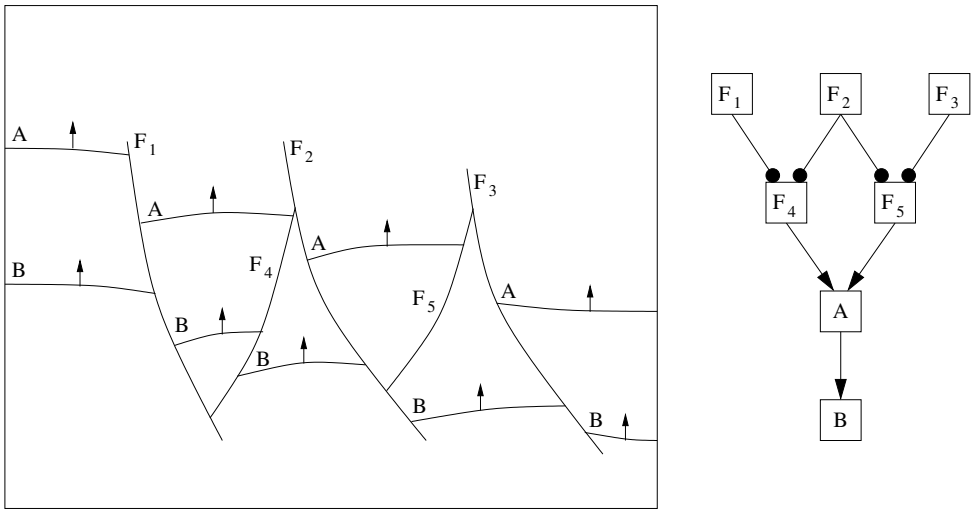


FIG. 9.8 – Une scène géologique comportant un réseau de failles complexe et son Schéma d'Évolution Géologique associé.

sont orientées de manière à parcourir le SEG de haut en bas. La figure 9.8 montre une scène géologique comportant un réseau de failles complexe.

Ce graphe permet donc de décrire sans ambiguité la structure d'une scène géologique et peut ainsi être utilisé afin de déterminer la manière dont doivent être assemblés les différents éléments d'un modèle structural.

9.2.3 Automatisation de la construction d'un modèle

Comme nous l'avons vu précédemment, le SEG permet de décrire précisément la structure d'un scène géologique ainsi que la chronologie des événements dont elle provient. De plus, il existe deux règles importantes en géologie structurale qui nous permettent de définir certaines propriétés sur la topologie finale du modèle reconstruit. Ces règles sont les suivantes :

- un événement plus ancien ne peut en aucun cas modifier un événement plus récent ;
- deux surfaces ne peuvent pas se croiser.

Ainsi, il est possible de construire un modèle structural de manière incrémentale en commençant par les éléments les plus récents et en finissant par les éléments les plus anciens. De cette manière, la topologie du modèle n'est pas remise en cause à chaque ajout d'une nouvelle surface.

Le SEG doit alors être parcouru du haut vers le bas et l'élément correspondant à chaque noeud doit être ajouté au modèle. Comme cet élément est plus ancien que les éléments déjà présent dans le modèle, ces derniers n'ont pas besoin d'être modifiés. Cependant, le nouvel élément ajouté doit être la plupart du temps découpé afin de ne pas traverser des surfaces déjà existantes dans le modèle. Pour ce faire, il est nécessaire de calculer l'intersection entre chaque nouvel élément et tous ses prédécesseurs en remontant dans le SEG depuis le noeud courant.

(a)

(b)

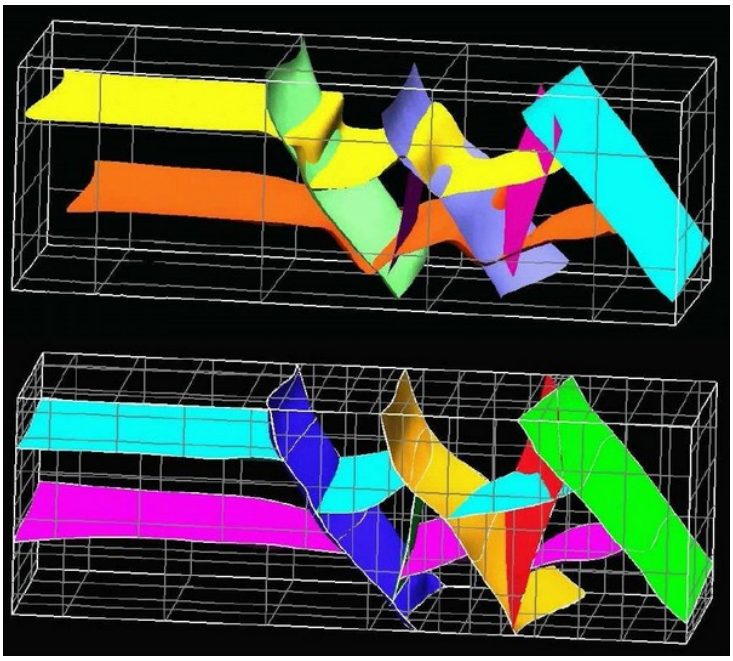


FIG. 9.9 – Exemple de modèle construit à l'aide du Pilote 3D : (a) les surfaces originales ; (b) le modèle structural construit.

Ce processus peut alors être automatisé puisque chaque ajout relève d'opérations élémentaires ne nécessitant pas l'intervention de l'utilisateur.

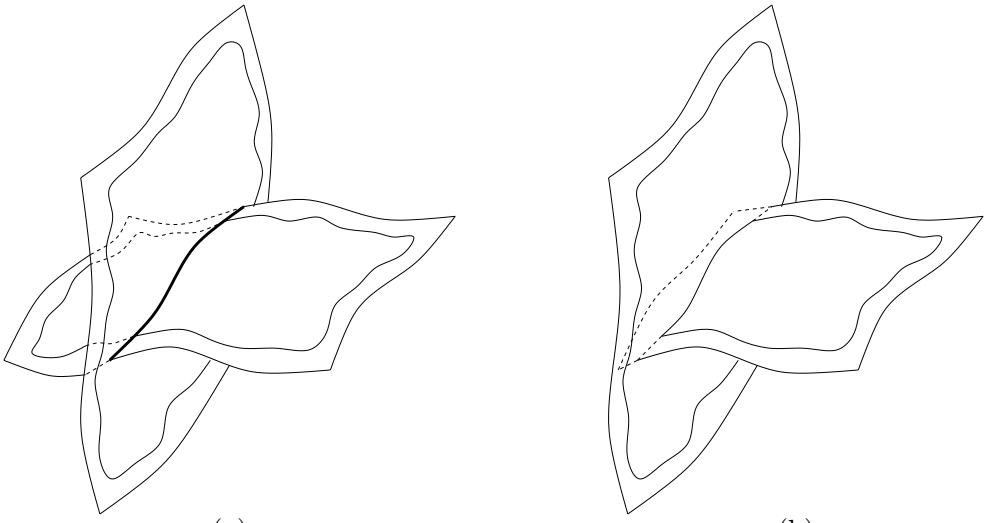
9.3 Le Pilote Géologique

Le pilote géologique 3D a été conçu par Sébastien Schneider durant sa thèse [Sch02] et consiste à automatiser la construction de scènes géologiques 3D en limitant au maximum les interventions de l'utilisateur. Il a été conçu comme un module du logiciel GeoSurf et utilise donc les fonctionnalités de la bibliothèque CAS.CADE pour réaliser les différentes opérations géométriques.

Les modèles qu'il permet de générer sont des modèles volumiques topologiquement consistants et décrivant toutes les relations géologiques et géométriques existantes entre les différents éléments (cf. FIG. 9.9). Pour cela, la topologie des modèles est décrite à l'aide de 3-G-Cartes auquelles ont été ajoutées plusieurs informations. Il est alors possible de connaître facilement les différents blocs appartenant à une même formation ainsi que la manière dont ils ont été générés.

9.3.1 Principe de fonctionnement

Son principe de fonctionnement est basé sur la méthode expliquée en section 9.2.3. Il prend en entrée un SEG et le parcours en ajoutant au fur et à mesure les différents éléments dans le modèle structural courant. Au début du traitement, ce dernier est représenté par une simple



(a) (b)

FIG. 9.10 – Exemple de découpe d'une surface paramétrique par une autre : (a) calcul de la courbe d'intersection entre deux surfaces ; (b) réduction du domaine de définition de la surface découpée.

boîte définissant les limites de l'étude. Cette boîte est ensuite découpée en blocs à l'aide des différentes surfaces ajoutées.

Afin de respecter les règles citées en section précédente, les surfaces insérées doivent être découpées par les surfaces déjà présentes dans le modèle. Le processus d'intersection de surfaces se passe alors en deux étapes. Tout d'abord, une courbe d'intersection est calculée entre les deux surfaces. Ensuite, le domaine de définition de la surface devant être découpée est réduit en calculant l'intersection avec cette courbe (cf. FIG. 9.10).

Lorsque l'intersection concerne un horizon et une faille, le processus est un peu plus compliqué. Dans ce genre de configuration, l'horizon doit être découpé par la faille mais aucune des parties résultantes ne doit être supprimée. Cependant, la partie d'horizon se trouvant à proximité de la faille est souvent une région possédant de grandes incertitudes géométriques. Elle doit alors être modifiée de manière à être dans le prolongement du reste de l'horizon. Cette déformation fait généralement apparaître un rejet le long de la faille.

Cette dernière opération d'intersection s'opère alors en plusieurs étapes (cf. FIG. 9.11). Tout d'abord, la faille est épaissie en dupliquant sa surface et en l'éloignant de chacun de ses côtés jusqu'à sortir de la région d'incertitude. Des courbes d'intersection sont ensuite calculées pour chacune des surfaces d'épaississement puis l'horizon est découpé le long de ces courbes. La région se trouvant entre les deux courbes est supprimée et les deux côtés de l'horizon sont extrapolés de manière à calculer des nouvelles courbes d'intersection distinctes de chaque côté de la faille, faisant ainsi apparaître un rejet.

Lors de la découpe des horizons par des failles, une information est rajoutée dans la G-Carte afin de relier les différentes parties d'horizons et de se souvenir de leur provenance. Cette information est modélisée à l'aide d'une involution β_2 ayant les mêmes propriétés que l'involution α_2.

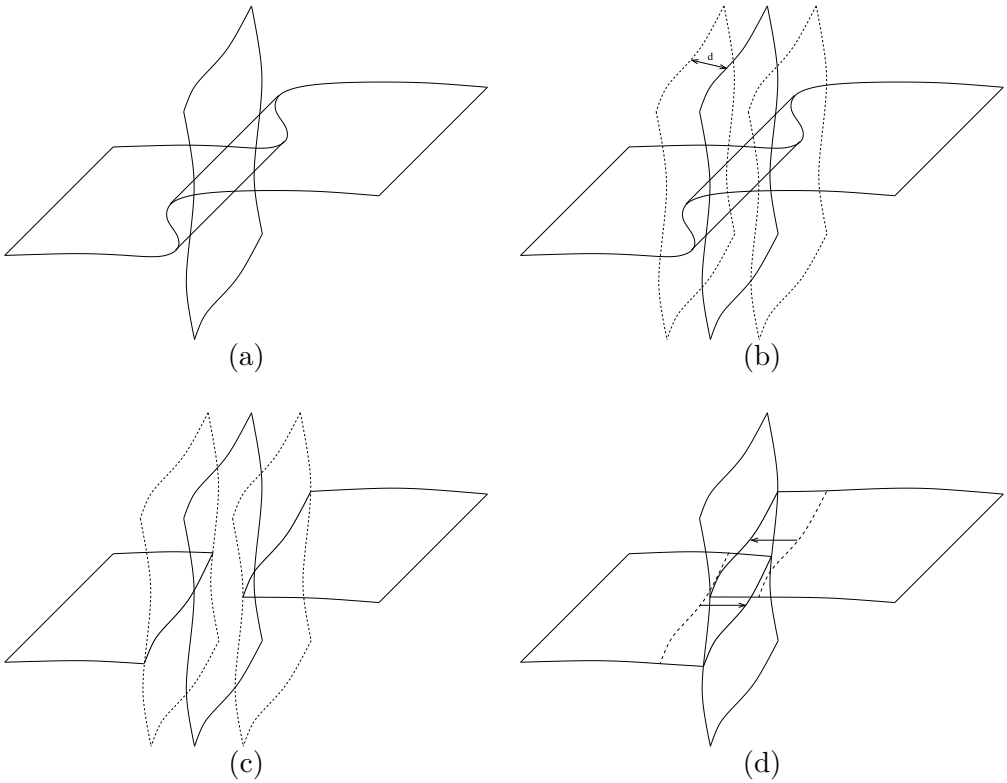


FIG. 9.11 – Exemple de découpe d'un horizon par une faille : (a) épaississement de la faille ; (b) calcul des courbes d'intersection entre l'horizon et les surfaces d'épaississement ; (c) découpage de l'horizon le long des courbes d'intersection et suppression de la zone d'incertitude ; (d) extrapolation des bords de l'horizon pour recoller à la surface de faille.

La seule différence entre ces deux involutions réside dans le fait que deux faces étant cousues par α_2 partagent une arête ayant une géométrie unique alors que ce n'est pas le cas avec la relation β_2.

9.3.2 Limites et problèmes rencontrés

L'utilisation de surfaces paramétriques a posé de nombreux problèmes au niveau de la modélisation et de la déformation des surfaces. Pour commencer, le pilote 3D devait respecter les limites imposées par le logiciel GeoSurf. Ainsi, il ne pouvait pas modéliser de failles pendantes et devait impérativement construire des blocs géologiques dont la géométrie est eulérienne.

De plus, lorsque les surfaces n'étaient pas assez grandes, il fallait systématiquement les étendre de manière à ce qu'elles découpent intégralement les blocs. L'extension se faisant par extrapolation du bord de la surface, des problèmes se posaient lorsque celles-ci possédaient de fortes courbures. Dans certains cas, il pouvait même se produire des phénomènes d'oscillations.

Chapitre 10

Nouvelle méthodologie

Suite aux restrictions apportés par la précédente version du Pilote 3D, une nouvelle méthodologie a été adoptée afin de pouvoir répondre aux différentes configurations géométriques existantes. Son but est alors de construire des topologies qui sont dépendantes des objets géologiques et non pas des solveurs qui sont utilisés pour effectuer des simulations d'écoulement. Pour ce faire, cette méthodologie utilise des surfaces triangulées au lieu des surfaces paramétriques, permettant ainsi de représenter des objets plus complexes.

Nous présentons tout d'abord dans la section 10.1 les modèles de représentation que nous utilisons pour construire les scènes géologiques. Nous expliquons ensuite dans la section 10.2 le processus de construction adopté.

10.1 Modèles de représentation

Pour modéliser les surfaces des scènes géologiques, nous utilisons le modèle des cartes généralisées de dimension 3. Ce modèle nous permet de représenter n'importe quel type de surface et nous permet d'effectuer des opérations complexes sur celles-ci (comme par exemple des intersections à l'aide d'un algorithme de co-raffinement). Ce modèle nous permet de plus de représenter toutes les relations topologiques existantes entre les surfaces, ce qui facilite la définition des propriétés géologiques.

Cependant, comme une G-Carte n'est composée que de brins reliés les uns aux autres par des involutions, nous ajoutons à l'aide de nouvelles structures toutes les informations permettant de définir les propriétés géologiques d'un modèle.

De plus, la manipulation d'un modèle déjà construit ne requiert pas les mêmes besoins en terme d'accès aux données qu'un modèle en cours de construction. En général, lorsque nous voulons manipuler un modèle fini, nous souhaitons surtout accéder rapidement aux informations géologiques de celui-ci sans se préocuper de sa géométrie. Au contraire, lors de la construction d'un modèle, nous avons besoin de réaliser de nombreuses opérations topologiques et géométriques de bas niveau nécéssitant un accès direct aux sommets des surfaces.

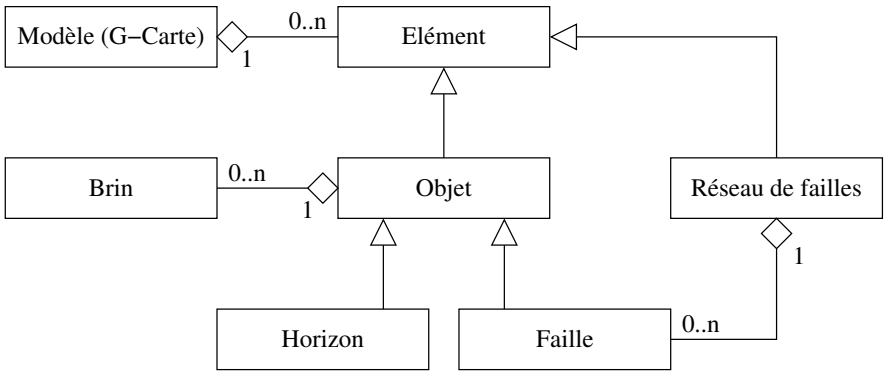


FIG. 10.1 – Diagramme UML de la structure.

Nous avons alors décidé de réaliser une première structure bas niveau nous servant à construire le modèle puis une structure haut niveau permettant d'accéder plus facilement aux informations géologiques. Ces deux structures peuvent être vues comme deux niveaux de détails du même modèle et la deuxième est simplement construite à partir de la première.

Ces deux structures sont basées sur une troisième qui permet de découper une G-Carte en groupes de brins correspondant par exemple à des objets géologiques. Nous détaillons dans la suite ces trois structures de données.

10.1.1 G-Cartes orientées objets géologiques

Un modèle géologique étant en général composé de plusieurs éléments (failles et horizons), nous avons besoin d'identifier au sein de la G-Carte, quels sont les brins appartenant à chaque objet. Pour cela, nous définissons une structure permettant de représenter un objet. Cette structure contient un ensemble de brins correspondant à la topologie de l'objet et chacun de ses brins connait l'objet auquel il appartient.

Notons que les objets ne sont pas des G-Cartes indépendantes les unes des autres car des brins appartenant à deux objets différents peuvent être reliés entre eux. Par exemple, un réseau de failles est composé de plusieurs failles reliées entre elles.

Par ailleurs, un réseau de failles est représenté sous une forme abstraite car il ne contient pas directement un ensemble de brins mais il contient un ensemble de failles qui sont des objets concrets. Ainsi, le modèle peut contenir plusieurs entités abstraites.

Au final, nous définissons un modèle géologique comme une G-Carte composée d'objets géologiques qui peuvent être soit concrets, soit abstraits. Un objet concret (comme un horizon ou une faille) est alors un sous type d'un objet abstrait auquel nous ajoutons une représentation topologique. La figure 10.1 montre le diagramme UML de la structure que nous utilisons.

10.1.2 Micro-modèle

Pour construire un modèle, nous utilisons une structure bas niveau que nous appelons « micro-modèle ». Celle-ci est basée sur la structure expliquée précédemment et définit plu-

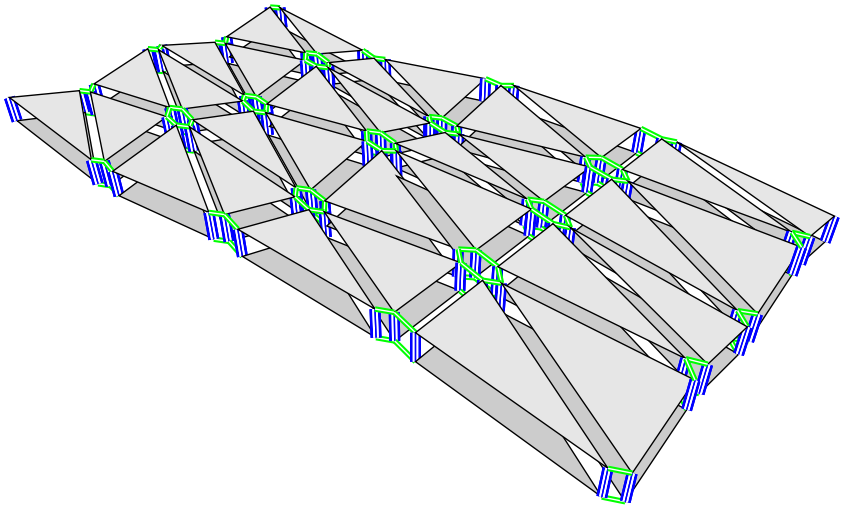


FIG. 10.2 – Modélisation d'une surface dans un micro-modèle. La surface est formée de deux volumes ouverts reliés entre eux par des involutions α_3.

sieurs propriétés complémentaires.

Dans cette structure, les surfaces géologiques correspondent à des ensembles connexes de brins. Ces surfaces sont plongés linéairement (*i.e.* un point par sommet topologique) et sont la plupart du temps formées de facettes triangulaires. Comme une surface correspond à une interface entre deux blocs géologiques, celle-ci possède deux faces avec des propriétés différentes. Nous avons alors décidé de modéliser ces deux faces en représentant une surface à l'aide de deux volumes ouverts collés entre eux (cf. FIG. 10.2).

Nous pouvons ensuite attribuer des propriétés aux surfaces à différents niveaux du modèle. Concernant le type de l'objet, son nom et les autres propriétés abstraites, ces informations sont stockées au niveau de l'objet contenant les brins. Cependant, pour toutes les propriétés physiques comme par exemple la polarité de la surface, ses liaisons avec d'autres surfaces, ces informations sont stockées directement au niveau des brins.

Pour affecter des propriétés au niveau des brins, nous utilisons deux méthodes qui sont le marquage et l'ajout de nouvelles involutions. Pour définir la polarité d'une surface, nous marquons à l'aide d'une marque © les côtés de surfaces (*i.e.* les volumes) devant être concordants. Cependant, comme le modèle des G-Cartes permet de représenter des objets non orientables, les côtés des surfaces ne possèdent pas d'orientation (*i.e.* nous ne savons pas lequel des deux volumes correspond à la surface du dessus). Pour cela, nous les orientons en les marquant un brin sur deux à l'aide d'une marque ⓞ. Cette marque permet ainsi d'identifier les brins de la surface à partir desquels les parcours des facettes doivent être effectués et par conséquent elle détermine l'orientation des normales à ces facettes (cf. section 2.1.3).

Lors de la construction du modèle, nous devons être capable de connaître les brins provenant de découpes et nous devons disposer de plusieurs informations à leur sujet. Nous définissons alors deux nouvelles involutions que nous notons β_2 et γ_2.

L'involution β_2 possède les mêmes propriétés que celle utilisée dans la précédente version du

Pilote 3D. Elle permet tout simplement de relier deux parties d'un même horizon qui ont été séparées suite à l'intersection avec une faille. L'involution γ_2 permet quant à elle de connaître la provenance d'une intersection. Elle est utilisée pour relier un bord d'horizon provenant d'une découpe à la surface ayant généré l'intersection. La manière de définir ces involutions est expliquée dans la section 10.2.4.

10.1.3 Macro-modèle

Pour pouvoir facilement manipuler le modèle après construction, nous utilisons une structure haut niveau que nous appelons « macro-modèle ». Celle-ci permet de représenter uniquement la topologie de la scène géologique au niveau des relations entre les différents objets la composant.

Un macro-modèle possède exactement les mêmes propriétés qu'un micro-modèle. Par conséquent toutes les informations stockées dans le micro-modèle sont retrouvées dans le macro-modèle : polarité, orientation, relations β_2 et γ_2. La seule différence existante entre les deux types de modèles se trouve au niveau de la réprésentation des objets.

Nous avons vu précédemment qu'une surface d'un micro-modèle est réprésentée par un très grand nombre de facettes et par conséquent par un très grand nombre de brins. De plus, lorsque le modèle est reconstruit, les informations les plus importantes (relations β_2 et γ_2) sont stockées dans les brins du bord des objets. Ainsi, tous les brins se trouvant à l'intérieur des surfaces ne sont pas utiles et allourdissent les traitements lors de la récupérations de ces informations.

Le macro-modèle vise alors à supprimer tous les brins inutiles de la G-Carte en représentant les surfaces à l'aide de simples faces topologiques. Ces faces ne comportent alors que quelques brins et leur plongement est défini à l'aide des surfaces du micro-modèle (cf. Fig. 10.3). Pour conserver une relation entre les brins du macro-modèle et ceux du micro-modèle, nous définissons une nouvelle involution que nous notons δ. Ainsi, les brins d'une macro-face sont reliés aux brins correspondant du micro-modèle. Cette involution permet de pouvoir retrouver les informations de plongement pour les sommets et arêtes d'une face en plus du plongement surface.

10.2 Construction d'un micro-modèle

La construction d'un micro-modèle est similaire à celle effectuée dans la précédente version du Pilote 3D (cf. section 9.3). Au départ, les seules informations dont nous disposons sont des surfaces isolées représentant des failles et des horizons ainsi qu'un Schéma d'Évolution Géologique décrivant les relations entre ces surfaces. Le but du processus de construction est alors de calculer les intersections entre les différentes surfaces afin de reconstruire une topologie correspondant aux informations se trouvant dans le SEG.

Comme les structures de données diffèrent de l'ancienne version et que la nouvelle méthodologie permet de traiter des cas géologiques plus complexes, la totalité des étapes de construction du modèle ont alors dues être modifiées. Ces étapes sont maintenant principalement basées sur l'utilisation de l'algorithme de co-raffinement détaillé dans le chapitre 7. Nous utilisons par ailleurs de nombreux autres petits algorithmes permettant d'assembler les différents morceaux entre eux pour obtenir un modèle final cohérent.

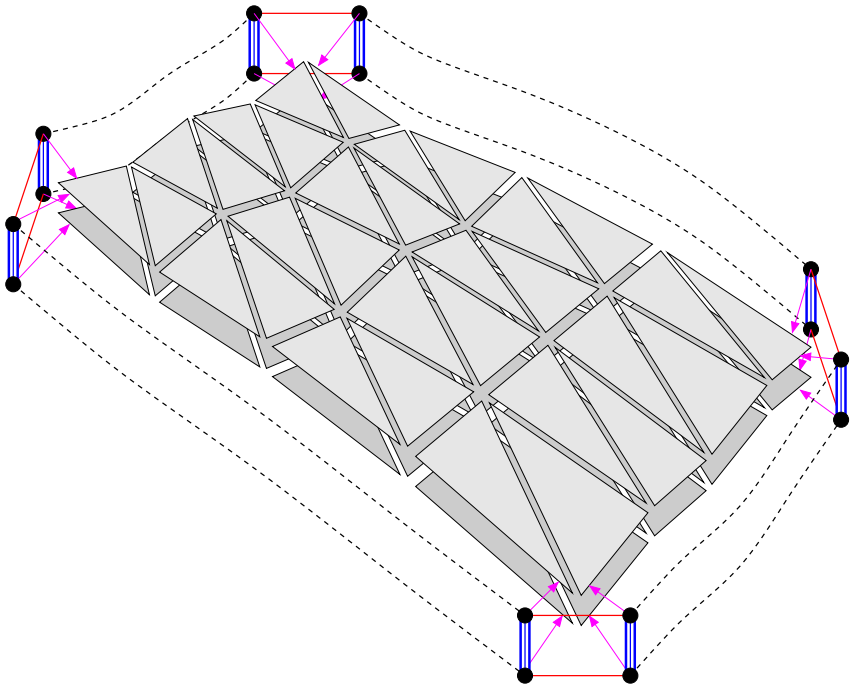


FIG. 10.3 – Exemple de face d'un macro-modèle plongée à l'aide d'une surface du micro-modèle.

Les différentes étapes de construction ainsi que les opérations effectuées pour chacunes d'entre elles sont expliquées dans les sections suivantes. Nous ne donnons pas ici d'algorithmes détaillés mais simplement la méthode utilisée.

10.2.1 Présentation de la méthodologie

Comme nous l'avons vu lors de la présentation du Pilote 3D (cf. section 9.3), l'opération la plus importante du traitement consiste à calculer l'intersection entre un horizon et une faille. Lors de cette phase, l'horizon est découpé par la zone d'incertitude de la faille afin de supprimer la partie inutile de l'horizon. Ensuite, les bords de l'horizon provenant de l'intersection sont étendus au delà du plan de faille pour être redécoupés.

Ce traitement est possible car les contraintes imposées aux surfaces permettent de le faire. Par exemple, les surfaces devant découper l'intégralité du modèle, les horizons sont ainsi découpés en deux parties et l'extension des surfaces est alors aisée.

Dans notre cas, nous voulons pouvoir gérer des failles ne traversant pas tout le modèle et qui puissent s'arrêter en plein milieu d'un horizon. Si nous gardons la même méthodologie, nous nous confrontons à des problèmes lors de l'intersection entre la faille et l'extension de l'horizon. En effet, dans le cas de failles pendantes, l'extension de l'horizon peut générer des auto-intersections (cf. FIG. 10.4) et le résultat ne peut alors pas être traité par l'algorithme de co-raffinement sous peine d'obtenir une topologie finale incorrecte.

Pour pallier ce problème, nous avons décidé de ne pas supprimer la région de l'horizon

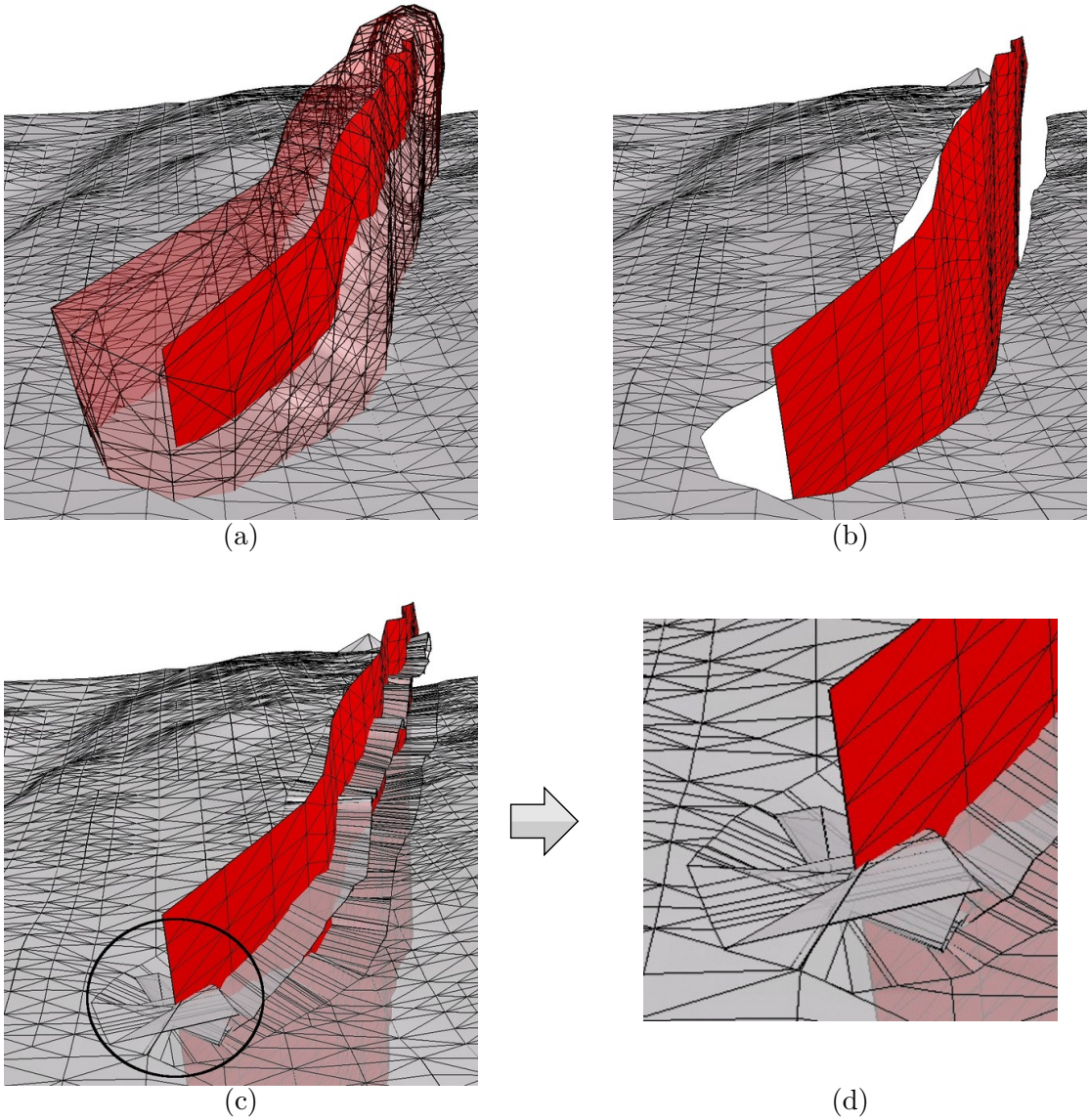
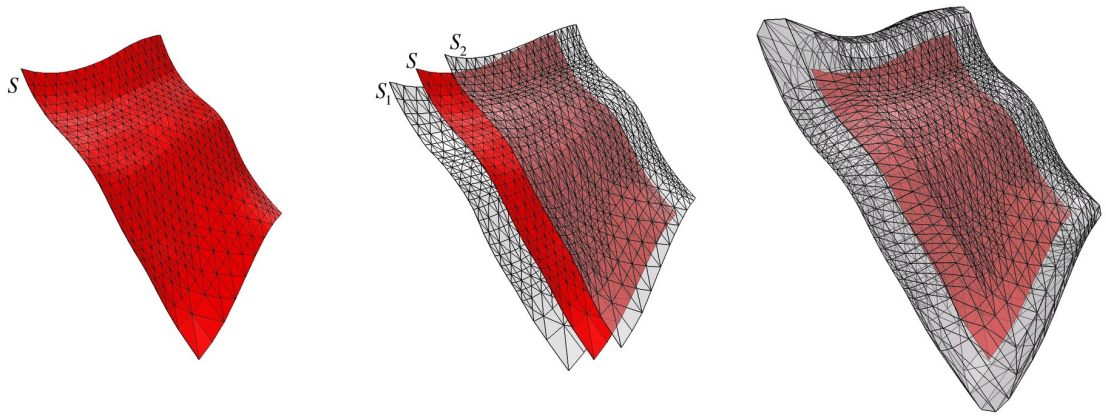


Fig. 10.4 – Exemple d'extension d'un horizon après intersection avec la zone d'incertitude d'une faille pendante : (a) la zone d'incertitude entourant la faille ; (b) l'horizon découpé par la zone d'incertitude ; (c) auto-intersections obtenues après l'extension de l'horizon.



(a) (b) (c)

FIG. 10.5 – Exemple de zone d'incertitude construite autour de la surface d'une faille : (a) surface S de la faille à épaissir ; (b) duplication de S et écartement des copies S_1 et S_2 de chaque côté de la faille ; (c) création du volume final en reliant les bords des surfaces S_1 et S_2 à l'aide d'une surface arrondie.

découpée par la zone d'incertitude. Ainsi, comme nous conservons ces données, nous n'avons plus besoin d'effectuer d'extension pour recoller l'horizon sur la surface des failles. La zone d'incertitude sert alors uniquement à délimiter les régions d'incertitude sur les horizons qui sont ensuite découpés par les failles. Pour faire apparaître le décrochement généré par les failles, il ne reste plus qu'à faire glisser les bords des horizons découpés le long des surfaces de failles (cf. section 10.2.4).

Une autre nouveauté par rapport à la version précédente du Pilote 3D, est que nous ne considérons plus les failles séparément dans le cas où elles forment des réseaux de failles. En effet, comme les failles appartenant à un même réseau de failles correspondent à des événements d'âges identiques, nous construisons tout d'abord l'objet correspondant au réseau de failles. Ensuite, lorsque ce dernier est construit, nous pouvons calculer l'intersection entre le réseau de failles complet et les horizons qu'il coupe.

10.2.2 Construction de zones d'incertitude autour des failles

Pour construire une zone d'incertitude autour d'une faille, nous dupliquons tout d'abord la surface S de la faille en deux nouvelles surfaces S_1 et S_2. Ces nouvelles surfaces sont ensuite écartées de chaque côté de S suivant les vecteurs normaux aux sommets de S (cf. FIG. 10.5(b)).

Comme le processus doit être capable de gérer les failles pendantes, la zone d'incertitude entourant la faille doit être un volume fermé dont tous les sommets se trouvent à égale distance de la surface S. Pour ce faire, nous relions les bords des surfaces S_1 et S_2 en construisant une surface arrondie (cf. FIG. 10.5(c)). Au final, le volume que nous obtenons ressemble à une sorte de « coussin ».

Lorsque que la surface S comporte de fortes courbures, il est possible que les surfaces S_1 et S_2 puissent contenir des auto-intersections. Ces artefacts pouvant générer des problèmes lors de

futurs calculs d'intersection, nous les faisons disparaître en adoucissant les surfaces S_1 et S_2. Si les courbures sont vraiment trop importantes et qu'il s'agit de défauts sur la surface S, nous lissons au préalable cette dernière.

10.2.3 Construction des réseaux de failles

Pour construire un réseau de failles, nous devons calculer les intersections entre toutes les failles qu'il contient ainsi qu'entre leurs zones d'incertitude respectives. La plupart du temps, les failles se trouvant dans un tel réseau s'arrêtent les unes sur les autres et les morceaux de failles inutiles sont supprimés. Pour effectuer une telle opération, nous devons co-raffiner les objets dans un ordre précis puis supprimer les parties en trop après chaque intersection.

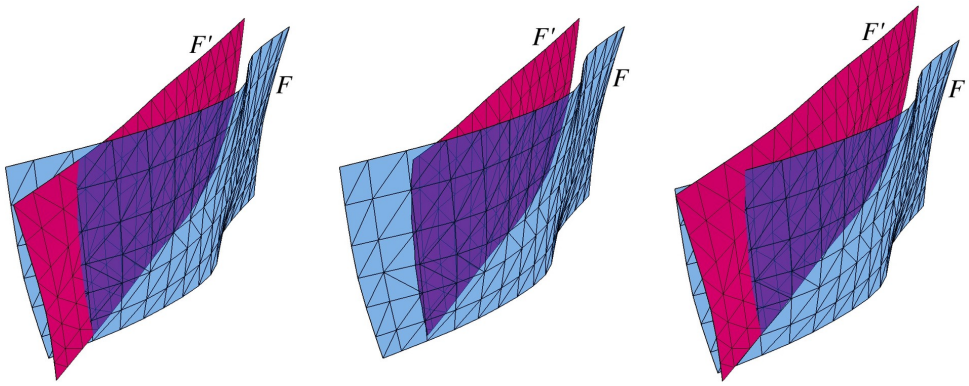
Cet ordre est défini par le SEG et en particulier par les relations structurales qui indiquent quelles sont les failles s'arrêtant les unes sur les autres. Comme nous l'avons vu dans la section 9.2.3, les surfaces doivent être traitées en commençant par la plus récente. Cependant les failles d'un même réseau possèdent toutes le même âge, nous commençons alors par la faille qui n'est pas modifiée, c'est-à-dire celle qui recoupe toutes les autres. Pour cela, il suffit simplement de parcourir le SEG de haut en bas de la même manière que pour les relations temporelles.

Nous allons maintenant voir comment calculer l'intersection entre deux failles et quel en est le résultat. Puis nous expliquerons comment effectuer la même opération avec les zones d'incertitude.

10.2.3.1 Intersection entre deux failles

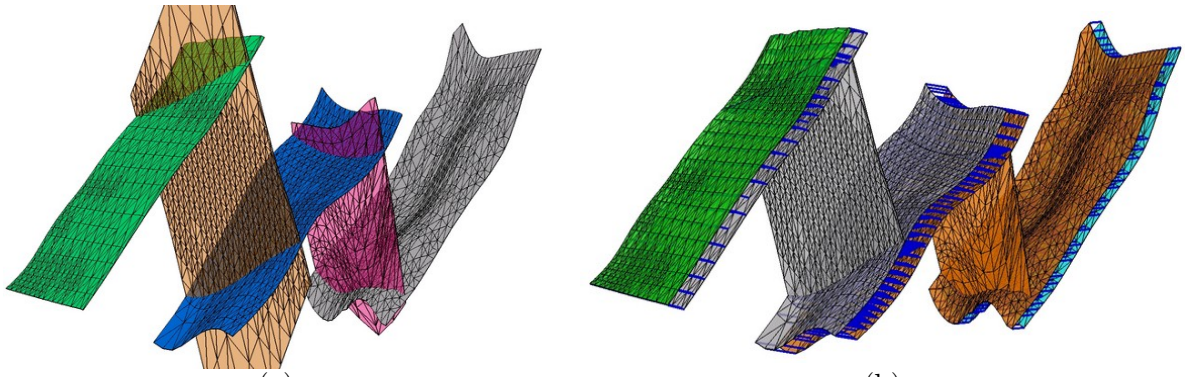
Lors de l'intersection entre deux failles, plusieurs cas de figures peuvent se présenter. Ces cas dépendent essentiellement des positions et formes relatives des failles. Si une faille F découpe intégralement une autre faille F' (cf. Fig. 10.6(a)), F' se retrouve divisée en plusieurs morceaux et il est possible d'isoler ceux ne devant pas être conservés (cf. Fig. 10.6(b)). Dans le cas contraire, F' n'étant pas coupée suffisamment, aucun morceau ne peut être supprimé (cf. Fig. 10.6(c)) ;

Les deux cas pouvant apparaître dans une scène géologique, l'utilisateur est le seul à pouvoir modéliser correctement les surfaces en fonction du résultat qu'il souhaite obtenir. De même, lorsque le SEG indique qu'une faille F s'arrête sur une autre faille F', la faille F ne doit pas être plus large que la faille F' et découper intégralement cette dernière. Aujourd'hui un traitement s'appuyant sur les règles définies par le GES ne peut être totalement automatique que si cette configuration est respectée. Traiter les autres configurations sera possible en enrichissant le GES afin de décrire d'autres configurations et de lever les ambiguïtés qui en découlent. Trois cas devront être traités : la faille F est plus large que la faille F', la faille F ne coupe que partiellement la faille F', la faille F' doit être recoupée par la faille F. D'autres heuristiques devront être appliquées en fonction de ces configurations. Ces heuristiques donneront lieu a des enchaînements d'opérations qui ont été effectuées jusqu'ici interactivement (extensions, découpages, lissages...). Certains marquages interactifs devront donc être effectués par l'utilisateur lors de la mise en place des réseaux de failles. Pour les exemples qui nous ont été confiés, ces opérations interactives existent mais n'ont pour l'instant pas été automatisées.



(a) (b) (c)

FIG. 10.6 – Exemples d'intersections entre deux failles : (a) la faille F découpe intégralement la faille F' ; (b) un morceau de la faille F' peut être isolé et supprimé ; (c) la faille F ne découpe que partiellement la faille F' : aucun morceau de F' n'est supprimé.



(a) (b)

FIG. 10.7 – Exemple de réseau de failles reconstruit : (a) failles dont l'intersection doit être calculée ; (b) volumes résultant de la construction du réseau de failles.

Les calculs d'intersection sont ensuite exécutés en suivant les règles ainsi définies. Ces calculs d'intersection étant réalisés à l'aide de l'opération de co-raffinement, le résultat obtenu correspond à une nouvelle subdivision de l'espace en plusieurs volumes. Les surfaces des failles sont alors reliées topologiquement les unes aux autres et définissent les interfaces entre ces volumes (cf. FIG. 10.7).

Suite au calcul d'intersection, les morceaux gênants de la surface découpée sont supprimés. En général, nous conservons le morceau le plus grand et nous supprimons tous les autres. Cependant, ce choix doit pouvoir être facilité par des informations fournies par le géologue. Nous obtenons alors l'objet final que nous pouvons utiliser de nouveau pour calculer les intersections avec d'autres failles plus anciennes.

10.2.3.2 Intersection entre deux zones d'incertitude

L'intersection entre deux zones d'incertitude est régie par les mêmes règles que l'intersection entre deux failles. Ainsi, la structure du résultat de l'intersection entre deux zones d'incertitude est identique à celle de l'intersection des failles correspondantes. Cependant, les zones d'incertitude étant régies par un paramètre supplémentaire qui est leur épaisseur, il est important de régler ce dernier en correspondance avec la structure devant être obtenue. En effet, le résultat obtenu après intersection est directement dépendant de l'épaisseur des zones. Si une zone est plus grosse qu'une autre, il est probable que la plus petite soit découpée en deux parties (cf. Fig. 10.8).

De plus, d'un point de vue physique, ces zones d'incertitude correspondent à des espaces broyés qui perturbent les mesures et produisent des artéfacts sur les horizons au voisinage des failles. Ceci implique d'être circonspect quant à la validité de la modélisation près de ces zones. Par conséquent la qualité des intersections est très dépendante de l'épaisseur de ces zones de failles. Si l'intersection est calculée près de la faille, l'horizon y étant « perturbé », cela peut provoquer des artéfacts. Si l'intersection est calculée trop loin de la faille, il y a des risques de collision avec d'autres zones de failles. On voit donc à quel point le compromis est difficile à établir.

Sur des exemples réels cette valeur d'incertitude pourra être directement extraite des mesures et l'utilisateur pourra ainsi se baser sur des éléments physiques concrets. Comme nous travaillons aujourd'hui sans avoir une réelle connaissance de cette valeur, il s'agit d'un point d'ambiguïté pour lequel nous manquons de données. Nous proposons donc des solutions « pragmatiques » : l'épaisseur doit correspondre à la largeur de la zone ou l'horizon peut sembler mal modélisé (ou à l'épaisseur d'une zone de l'horizon où la courbure varie rapidement). Ceci entraîne de temps à autre un réglage de l'épaisseur en fonction du résultat envisageable.

Des opérations « d'essai erreur » préliminaires devront être proposées comme base à l'utilisateur, qui pourra en déduire des valeurs sur lesquelles il peut appliquer les algorithmes. Dans la suite du traitement, ces paramètres seront utilisés pour les reconstructions automatiques.

Le résultat brut du co-raffinement de deux zones d'incertitude consiste en un assemblage de volumes appartenant soit à la première zone, soit à la deuxième, soit aux deux (cf. Fig. 10.9). Lorsque deux zones d'incertitude sont intersectées à l'aide de l'algorithme de co-raffinement, nous supprimons alors les volumes gênants puis nous fusionnons les volumes restants afin de n'en obtenir plus qu'un seul. Le volume résultant correspond donc à la zone d'incertitude du réseau de failles et il peut ensuite être mis à jour lors de l'insertion de nouvelles failles dans le réseau.

10.2.4 Découpage des horizons par les failles

Lorsqu'un horizon est découpé par une faille, le résultat devant être obtenu est un horizon ayant subi un décalage au niveau de la faille et dont le bord provenant de la découpe colle au mieux à la surface de la faille. Cette opération se réalise en plusieurs étapes que nous détaillons dans les sections suivantes.

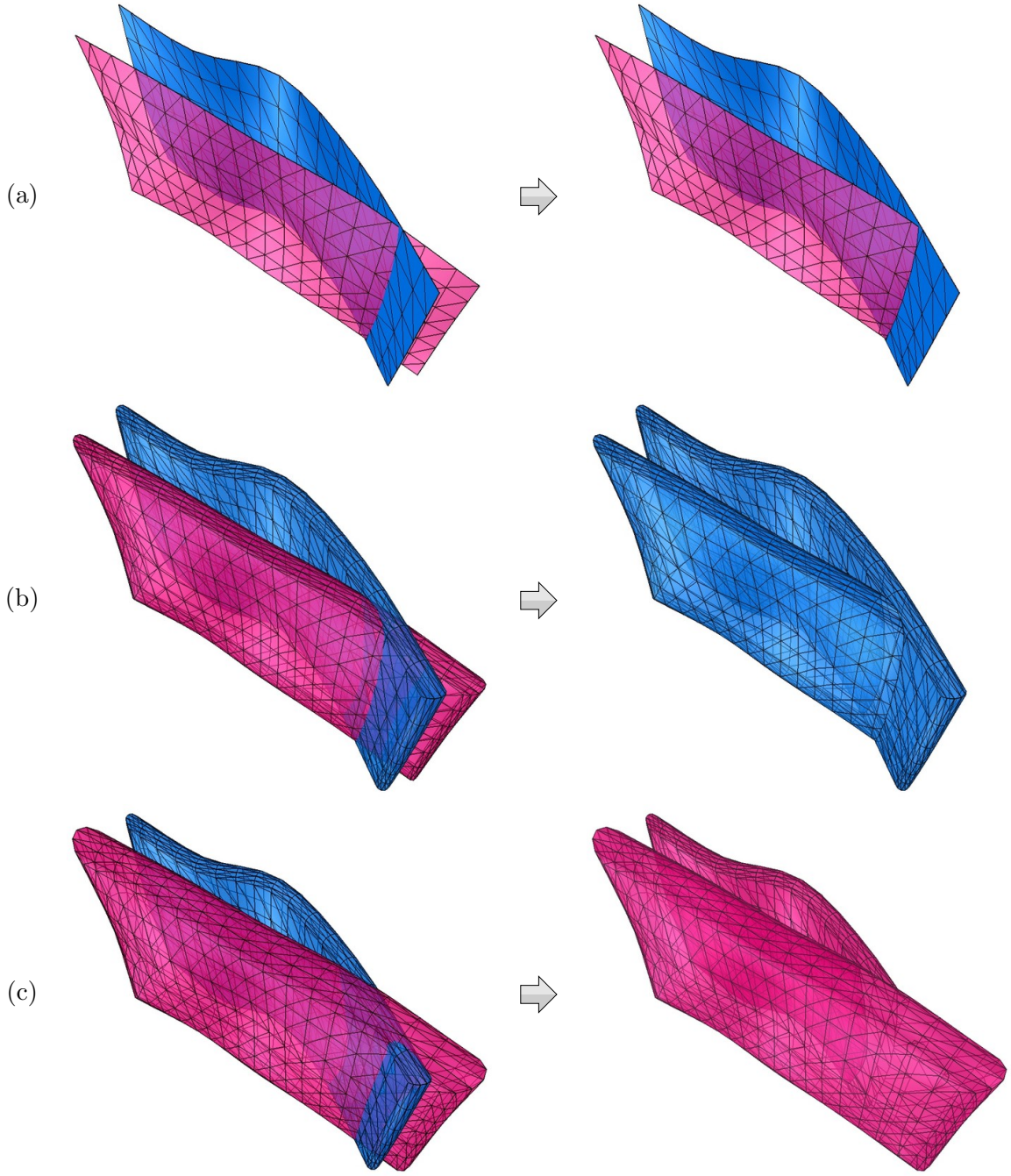
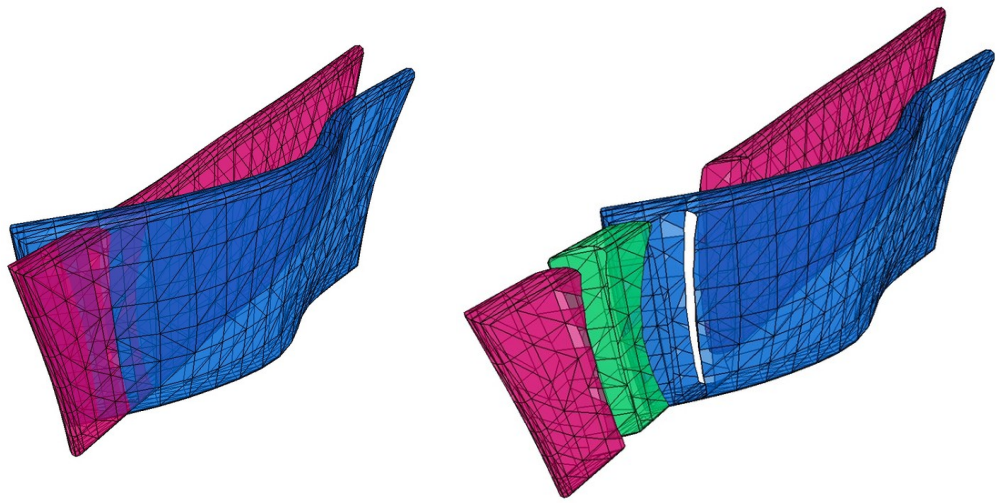


FIG. 10.8 – Résultats d'intersections obtenues entre deux zones d'incertitudes en fonction de leur épaisseur : (a) les failles en intersection et le résultat obtenu ; (b) l'épaisseur de la faille coupée est égale à celle de l'autre faille ; (c) l'épaisseur de la faille coupée est supérieure à celle de l'autre faille.



(a) (b)

FIG. 10.9 – Exemple de co-raffinement entre deux zones d'incertitude : (a) les zones en intersection ; (b) vue éclatée des volumes obtenus à la suite du co-raffinement.

10.2.4.1 Traitement des régions d'incertitude

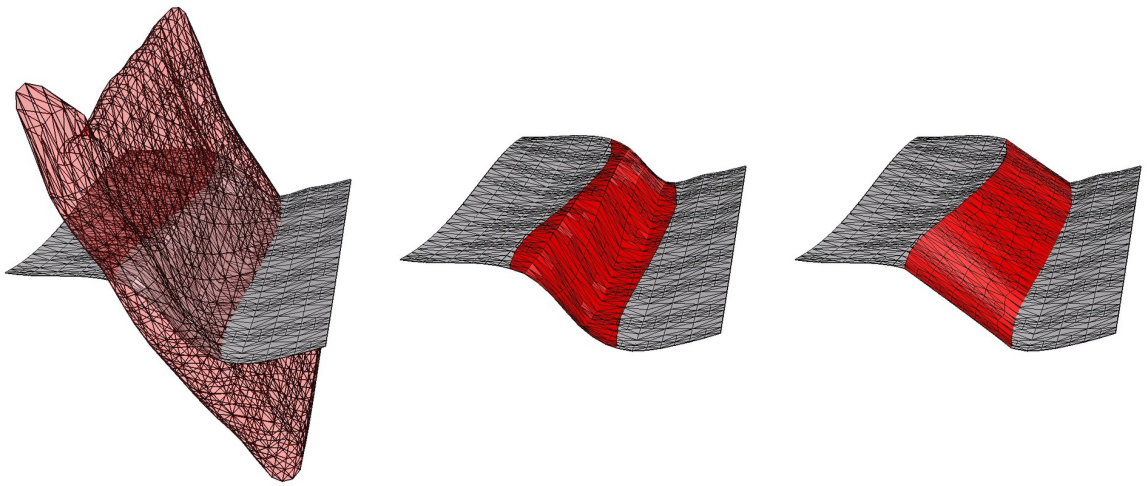
La première étape consiste à repérer sur l'horizon la région se trouvant à l'intérieur de la zone d'incertitude de la faille. Pour cela, nous utilisons la première étape de l'algorithme de co-raffinement pour calculer les arêtes d'intersection entre l'horizon et la zone d'incertitude. Ainsi, les deux objets ne sont pas reliés entre eux mais nous connaissons les arêtes d'intersection communes. Nous pouvons alors marquer les facettes de l'horizon se trouvant entre ces arêtes d'intersection (cf. FIG. 10.10(b)).

Lorsque cette région est récupérée, nous l'adoucissons afin d'obtenir une région la plus plate possible (cf. FIG. 10.10(c)). Cette opération est importante car elle conditionne le découpage de l'horizon et par conséquent la topologie résultante.

10.2.4.2 Découpage de l'horizon

Comme l'étape précédente a permis de simplifier la région de l'horizon se trouvant dans la zone d'incertitude, nous pouvons maintenant calculer l'intersection entre ce dernier et la faille en utilisant l'algorithme de co-raffinement. L'étape précédente nous assure que l'horizon ne possède plus de pli dans la région d'incertitude et que par conséquent, ce dernier ne peut pas définir plusieurs lignes de coupes avec la faille. Cette propriété est très importante car la topologie de l'intersection effectuée définit la topologie finale du modèle et les relations entre les différents objets géologiques.

Après l'intersection, l'algorithme de co-raffinement a relié les deux objets entre eux pour n'en former plus qu'un. Nous devons alors les découdre en nous souvenant de la position des involutions α_2 générées par l'algorithme de co-raffinement. En effet, ces involutions sont importantes car elles correspondent aux futures involutions β_2 et γ_2.



(a) (b) (c)

FIG. 10.10 – Traitements associés aux régions d'incertitudes sur les horizons : (a) l'horizon et la zone d'incertitude à traiter ; (b) marquage de la région de l'horizon se trouvant dans la zone d'incertitude ; (c) lissage de la région d'incertitude.

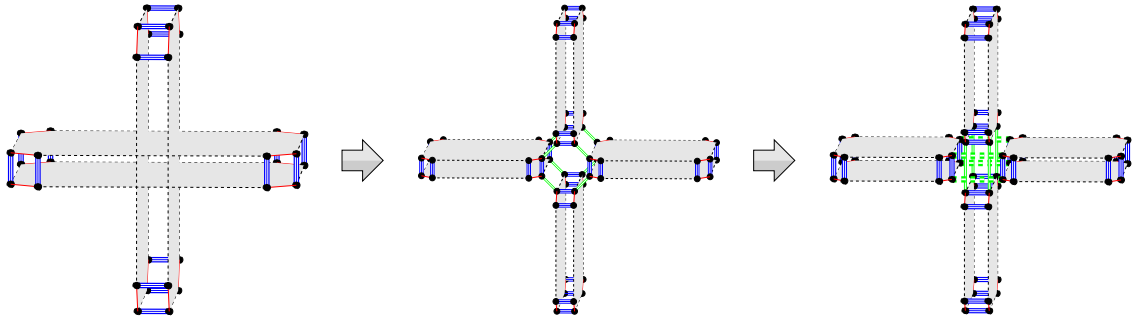
Pour découdre les deux surfaces, nous parcourons les segments de coupe qui ont été fournis par l'algorithme de co-raffinement et nous effectuons les opérations suivantes pour chacun d'entre eux (cf. FIG. 10.11) :

1. Nous recopions l'état des involutions α_2 reliant l'horizon et la faille dans les involutions γ_2. Ainsi, lorsque l'horizon sera découpé et détaché de la faille, nous connaîtrons toujours la provenance des bords de la coupe générés par l'intersection.
2. Nous relions les brins de l'horizon se trouvant de chaque côté de la faille par des involutions β_2. Les brins correspondant sont retrouvés en parcourant l'orbite de l'arête topologique. Les involutions β_2 nous permettront ensuite de garder une liaison entre deux portions d'horizons découpés.
3. Nous décousons par α_2 les brins n'appartenant pas au même objet et nous recousons par α_2 les brins de la faille se trouvant de chaque côté de l'horizon. Ces opérations permettent de séparer les deux objets en laissant une découpe sur l'horizon et une « cicatrice » sur la faille.

Lorsque ces opérations sont terminées, nous obtenons la topologie finale du « micro-modèle » et les seules opérations restantes sont purement géométriques.

10.2.4.3 Déplacement des bords de l'horizon

Suite au découpage de l'horizon, la région d'incertitude est restée aussi plate que nous l'avons laissée après l'opération de lissage (cf. section 10.2.4.1). Elle doit alors être déformée afin de modéliser (si possible) un décalage entre les portions d'horizon se trouvant de chaque côté de la faille. Ce décalage apparaît en déformant la région d'incertitude de manière à ce qu'elle suive



(a) (b) (c)

FIG. 10.11 – Les différentes étapes de la mise à jour de la topologie durant le processus d'intersection entre une faille et un horizon : (a) les surfaces avant l'intersection ; (b) la configuration topologique générée par l'algorithme de co-raffinement : les involutions α_2 obtenues correspondent aux futurs involutions γ_2 ; (c) séparation topologique de la faille et de l'horizon : les doubles pointillés en gras correspondent aux involutions β_2.

la courbure que l'horizon possède à l'extérieur de cette région. De plus, lors de la déformation, les bords provenant de la découpe doivent se déplacer le long de la surface de faille tout en y restant collés.

Ce processus de déformation s'opère en plusieurs étapes qui sont les suivantes :

1. Pour chaque sommet S du bord de coupe, nous recherchons à l'extérieur de la région le point P le plus proche de S et se trouvant du même côté de la faille. Ce traitement s'effectue par propagation depuis S dans la région et ce jusqu'à sortir de celle-ci (cf. FIG. 10.12). Le plan tangent à P est ensuite calculé et associé à S de manière informelle. Ceci permet par la suite de définir une limite pour le déplacement du point.
2. Des « macro-sommets » sont détectés sur le bord de coupe à l'aide des liaisons γ_2 qu'ils entretiennent avec les failles. Ces sommets correspondent à ceux se trouvant sur le bord d'origine de l'horizon et ceux étant en contact avec une « macro-arête » du réseau de failles (*i.e.* les bords des failles et les jonctions entre failles). La figure 10.13 montre un exemple d'horizon découpé par un réseau de failles et les macro-sommets correspondants.
3. Les macro-sommets détectés précédemment sont déplacés en hauteur de manière à coller le plan qui leur a été associé lors de la première étape. Pour les sommets se trouvant sur le bord extérieur de l'horizon, le déplacement s'effectue suivant l'axe $\overrightarrow{z}$ projeté sur la surface de la faille. Pour ceux étant reliés à une macro-arête du réseau de failles, ils sont déplacés le long de cette arête. Pour ces derniers, si le plan ne peut pas être atteint en suivant l'arête, ils sont déplacés de la même manière que les autres. La figure 10.15 montre des exemples de vecteurs de déplacements associés aux sommets du bord d'un horizon découpé.
4. Les sommets du bord de coupe se trouvant entre deux macro-sommets sont ensuite déplacés en hauteur de la même manière mais en suivant une direction pondérée par les directions de déplacement des macro-sommets extrémités. Ces deux dernières étapes permettent ainsi d'obtenir une position approximative des sommets du bord (cf. FIG. 10.14(b)).

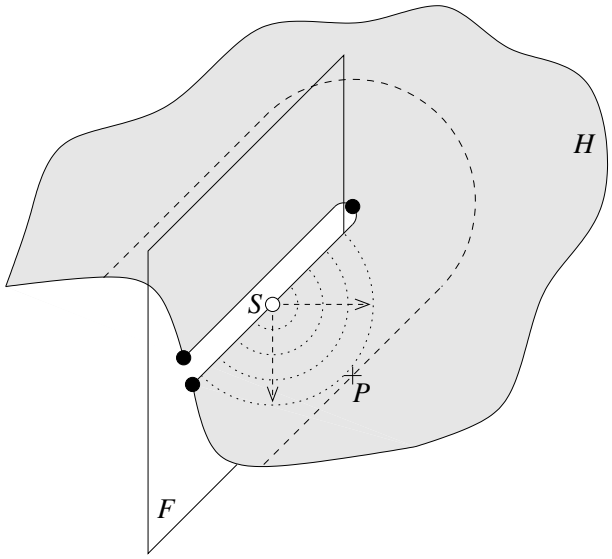


FIG. 10.12 – Exemple de recherche d'un sommet à l'extérieur d'une région d'incertitude. Le point P trouvé se situe du même côté de la faille F que du sommet de référence S. Il est déterminé en se propageant sur l'horizon H depuis S.

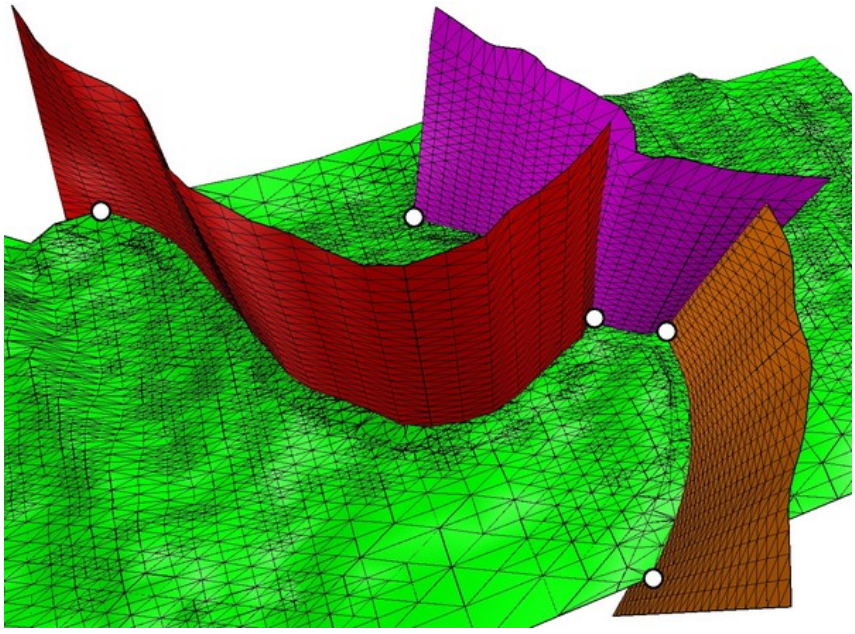


FIG. 10.13 – Exemple d'horizon découpé par un réseau de failles. Les points blancs correspondent aux macros-sommets détectés sur l'horizon.

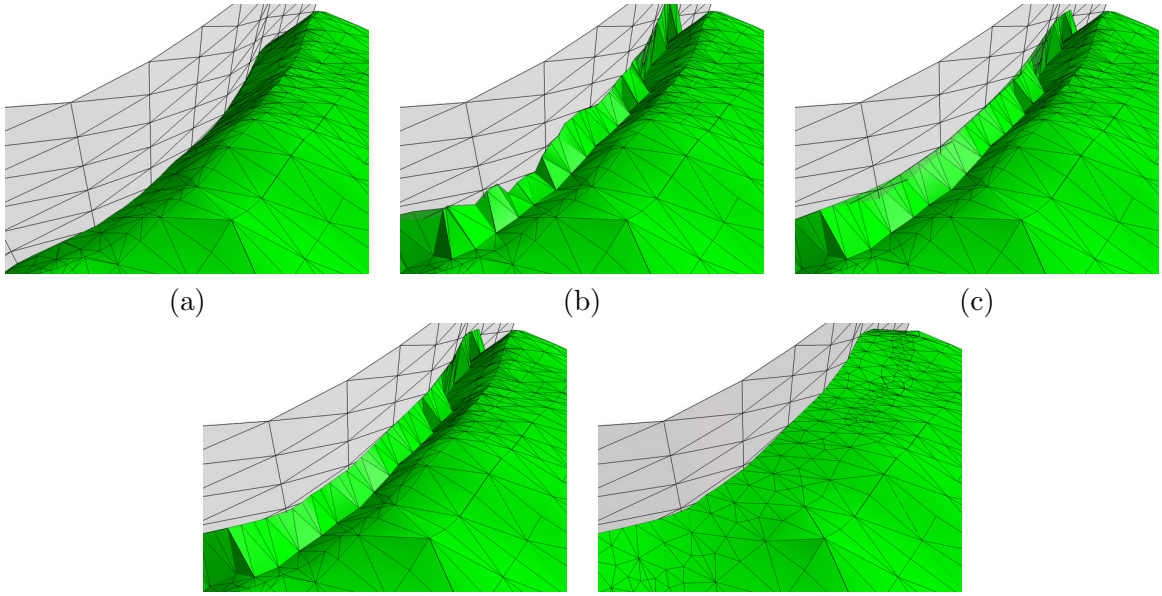


FIG. 10.14 – Exemple de portion d'horizon découpé par une faille : (a) la portion juste après la découpe ; (b) déplacement en hauteur de chaque sommet du bord de coupe ; (c) lissage du bord de coupe ; (d) projection des sommets du bord de coupe sur la surface de la faille ; (e) lissage de la l'intérieur de la région d'incertitude et obtention de la géométrie finale.

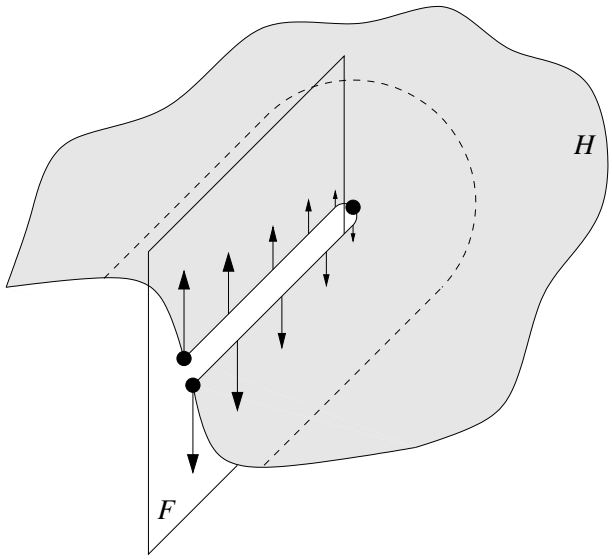


FIG. 10.15 – Exemple de vecteurs de déplacement associés aux sommets du bord d'un horizon découpé.

5. Le bord de coupe est ensuite lissé de manière à obtenir une courbe avec le moins de défauts possible (cf. Fig. 10.14(c)).
6. Les sommets du bord de coupe sont projetés sur les surfaces du réseau de failles afin de les recoller (cf. Fig. 10.14(d)). Pour être efficace, cette dernière opération est réalisée en utilisant le concept de suivi de ligne de coupe élaboré pour le premier algorithme de coraffinement (cf. chapitre 6). La méthode consiste alors à partir de la position originale du sommet sur la surface de la faille et à se propager dans les facettes de celle-ci en suivant la direction du déplacement approximatif obtenu après lissage (*i.e.* lors de l'étape précédente). Le parcours s'arrête lorsque le déplacement effectué sur la surface de faille est équivalent au déplacement approximatif. Cette technique permet ainsi de limiter le nombre de facettes testées pour déterminer avec précision le point correspondant à la projection du sommet à déplacer.
7. Pour finir, l'intérieur de la région d'incertitude (bord de coupe non compris) est lissé de manière à obtenir une surface la plus plate possible. La géométrie finale de l'objet est alors obtenue (cf. Fig. 10.14(e)).

10.2.5 Découpage de failles et d'horizons par une surface

Dans un modèle géologique, il n'est pas rare de rencontrer des horizons coupant d'autre horizons. C'est par exemple le cas des surfaces d'érosion qui découpent tous les objets correspondant à des événements antérieurs à celles-ci.

Pour modéliser un tel phénomène, nous utilisons la même méthode que pour le découpage d'un horizon par une faille excepté qu'ici la surface de découpe est un horizon. Cependant, cette découpe étant franche et ne devant générer aucun décalage, il n'y a aucune modification géométrique à opérer. Ainsi, dans un premier temps, seules les opérations décrites en section 10.2.4.2 sont effectuées.

Il est à noter que contrairement aux failles, deux horizons ne peuvent pas se croiser. L'horizon découpé doit par conséquent être moins large que l'autre afin de pouvoir être découpé intégralement (cf. section 10.2.3.1).

Lorsque l'intersection est réalisée, une partie de l'horizon découpé doit être supprimée. Cette partie est déterminée simplement par la polarité de l'horizon de coupe. Plus précisément, la partie devant être supprimée se situe du côté concordant de l'horizon de coupe. Ce côté étant marqué par la marque © (cf. section 10.1.2) lors de la création de l'horizon, il est facile de détecter quel est la portion d'horizon étant relié par γ_2 à des brins marqués par cette marque.

Comme il est possible que les surfaces des horizons devant être découpés possèdent des irrégularités et qu'elles soient plus ou moins parallèles entre elles, la topologie générée par le calcul peut être complexe, voire incorrecte. Il est alors possible de définir, comme pour les failles, une zone d'incertitude autour des horizons. Ainsi, avant de procéder au calcul d'intersection, les régions des horizons se trouvant dans l'intersection des zones d'incertitude peuvent être lissées afin d'obtenir un meilleur résultat.

10.2.6 Résultats

Nous présentons ici quelques exemples de micro-modèles reconstruits à partir de données surfaciques. Ces exemples ont été générés sur une machine PC équipée d'un processeur AMD Athlon 64 3000+ et de 512 Mo de mémoire vive. Les temps de calculs pour chaque étape de l'algorithme sont détaillés dans le tableau 10.1. Les modèles correspondants sont montrés en annexe B.

Modèles	IF	IZ	IHZ	IHF	OL	DB	OD	Total
Normal Faults	9 s	13 s	52 s	44 s	8 s	8 s	1 m 31 s	3 m 45 s
ResModOne	6 s	11 s	2 m 4 s	1 m 46 s	13 s	7 s	3 m 4 s	7 m 31 s
Extension1	7 s	16 s	8 s	8 s	1 s	1 s	19 s	1 m
Extension2	13 s	20 s	28 s	23 s	4 s	11 s	57 s	1 m 36 s
N'Kossa	1 m 14 s	1 m 46 s	3 m 47 s	3 m 23 s	23 s	14 s	6 m 21 s	17 m 8 s

IF : intersections entre failles
IZ : intersections entre zones d'incertitude
IHZ : intersections entre un horizon et une zone d'incertitude
IHF : intersections entre un horizon et une faille
OL : opérations de lissage
DB : déplacement des bords d'horizons provenant de découpes
OD : opérations topologiques diverses (marquages, cousures/décousures...)

Tab. 10.1 – Temps de calcul obtenus sur différents modèles.

Nous pouvons constater au vue de ces résultats que la plus grande partie du temps de calcul est consacrée aux opérations topologiques diverses. Ces opérations concernent les marquages, les orientations, les cousures/décousures, le chargement d'objets, etc. Elles sont donc toutes dépendantes de la structure de donnée utilisée. En l'occurrence, nous utilisons ici un noyau de cartes généralisées génériques qui n'est pas optimisé pour la géologie.

Nous pouvons aussi observer que les intersections engageant des horizons sont beaucoup plus longues que les intersections entre failles. Ceci est surtout dû au fait que les surfaces des horizons comportent en général plus de facettes que les surfaces des failles et que par conséquent, l'algorithme de co-raffinement doit effectuer plus de traitements.

10.3 Conversion d'un micro-modèle en macro-modèle

Dans la suite, nous expliquons comment il est possible de passer du modèle précédemment obtenu à un modèle en macro-topologie. Cette étape n'existait pas dans l'ancienne version du pilote car les objets traités faisaient déjà partie d'une macro-topologie et les calculs d'intersection étaient directement réalisés sur cette dernière.

Le passage de la micro-topologie à la macro-topologie se fait principalement à l'aide d'opérations topologiques et de quelques opérations géométriques qui, nous le verrons par la suite, peuvent être facultatives. Toutes ces opérations sont découpées en plusieurs étapes que nous expliquons les unes après les autres.

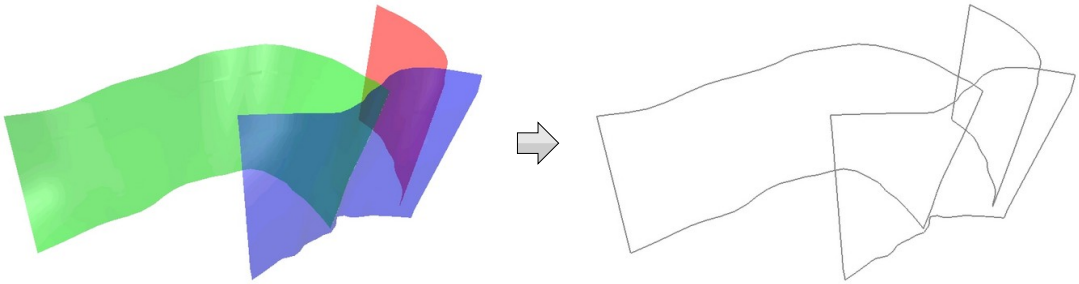


FIG. 10.16 – Exemple d'extraction des contours d'un réseau de failles : (a) les surfaces modélisant le réseau de failles ; (b) les contours extraits.

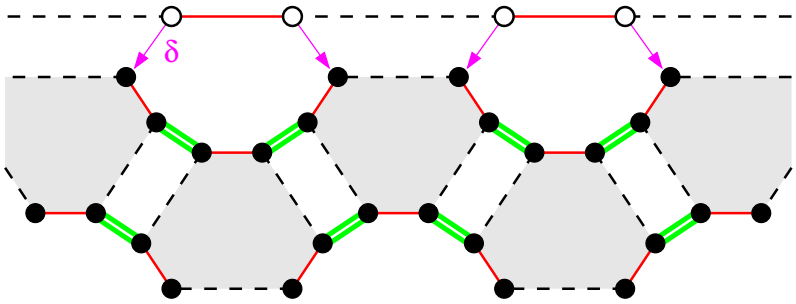


FIG. 10.17 – Exemple de macro-topologie générée à partir d'une micro-topologie. Les brins en blanc correspondent à ceux de la macro-topologie et les flèches reliant les brins de la macro-topologie à ceux de la micro-topologie correspondent aux relations δ.

10.3.1 Création de la carte des contours

La première étape de construction consiste à générer un premier macro-modèle grossier en retirant tous les brins du micro-modèle se trouvant à l'intérieur des surfaces. Ceci correspond donc à extraire les contours des surfaces du micro-modèle et à générer les macro-arêtes associées. De plus, comme le micro-modèle peut contenir des surfaces reliées les unes aux autres (par exemple dans les réseaux de failles), les arêtes modélisant ces jonctions sont aussi extraites (cf. FIG. 10.16).

Lors de la création des macro-arêtes, celles-ci sont reliées à leurs correspondantes dans le micro-modèle (arêtes du bord des surfaces) à l'aide de l'involution δ. De plus toutes les informations stockées dans le micro-modèle sont reportées dans les arêtes du macro-modèle. Ainsi les nouveaux brins possèdent les mêmes marquages par © et ⓪ que leurs correspondants et l'état des différentes involutions est conservé. Par exemple, les macro-brins sont reliés par les involutions α_0, α_2, α_3 et β_2 exactement de la même manière que leurs correspondants dans le micro-modèle. Cependant, les images des micro-brins par α_1 et γ_2 n'ayant la plupart du temps pas de correspondant dans le macro-modèle, ces involutions sont recalculées.

Concernant les nouvelles involutions α_1, elles permettent de relier les macro-arêtes entre elles afin de former des faces fermées. Les brins du macro-modèle à relier sont donc ceux correspondant

à des brins du micro-modèle qui appartiennent à un même sommet topologique (cf. FIG. 10.17). Pour les involutions γ_2, elles sont recalculées utlérieurement lorsque la topologie du macro-modèle est définitive (cf. section 10.3.4).

10.3.2 Plongement des macro-faces

Les faces créées lors de l'étape précédente doivent être plongées en utilisant les surfaces du micro-modèle. Ce plongement s'effectue simplement à l'aide des relations δ qui permettent d'accéder à la micro-topologie d'une surface depuis n'importe quel brin d'une macro-face.

Cependant, comme les surfaces du micro-modèle peuvent être reliées les unes aux autres (par exemple des failles formant un réseau), il est préférable de les désolidariser afin d'obtenir une composante connexe de brins pour chaque surface (*i.e.* pour chaque macro-face).

Ce découpage se fait tout simplement en décousant par α_2 tous les brins du micro-modèle ayant une image par δ dans le macro-modèle. Comme nous l'avons vu précédemment, ces brins correspondent aux macro-arêtes générées et plus particulièrement à celles modélisant des jonctions entre failles.

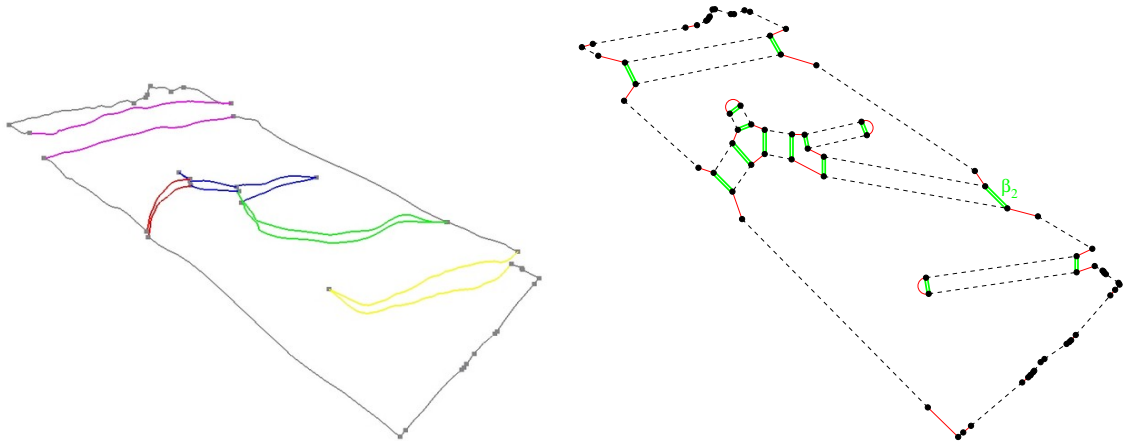
10.3.3 Simplification de la macro-topologie

La macro-topologie créée lors de la première étape peut encore être simplifée. Lors de la construction d'un micro-modèle nous avons vu qu'il était possible de repérer des sommets significatifs sur les bords des horizons et nous les avons nommés « macro-sommets » (cf. section 10.2.4.3). Ces sommets correspondent aux macro-arêtes des réseaux de failles et définissent par conséquence des frontières entre deux bords provenant d'intersections avec des objets différents.

Ainsi, toutes les arêtes se trouvant entre ces sommets proviennent de l'intersection avec un seul et même objet et sont reliées par β_2 à des arêtes ayant les mêmes propriétés. Nous pouvons alors supprimer toutes ces arêtes du macro-modèle pour n'en garder qu'une seule reliant les deux sommets. Nous obtenons alors une topologie beaucoup plus simple dans laquelle chaque intersection d'un horizon par une faille est représentée par deux macro-arêtes reliées par β_2 (cf. FIG. 10.18).

Concernant les bords des objets ne provenant pas d'intersections, nous voulons y faire apparaître des macro-sommets représentatifs de la géométrie des surfaces. Par exemple, pour une surface rectangulaire, nous voulons modéliser les coins de celle-ci. Contrairement aux autres macro-sommets, ceux-ci ne peuvent pas être détectés topologiquement car ils ne proviennent d'aucune intersection. Nous les détectons alors géométriquement en analysant la courbure moyenne en chaque micro-sommet du bord. Si cette courbure est supérieure à une valeur spécifiée par l'utilisateur, un macro-sommet est inséré.

Les sommets ainsi détectés dépendant essentiellement de la valeur de test fournie, il est possible que le résultat obtenu ne corresponde pas aux attentes de l'utilisateur (par exemple la figure 10.18(a) montre un très grand nombre de macro-sommets détectés sur le contour de l'horizon). Cependant, ces sommets étant facultatifs, ils n'influencent aucunement la cohérence



(a) (b)

FIG. 10.18 – Exemple de macro-topologie générée pour un horizon : (a) les macro-sommets détectés sur les contours de l'horizon ; (b) la macro-topologie résultante après suppression des arêtes inutiles (les doubles liaisons représentent les relations β_2).

topologique du modèle. L'utilisateur peut donc les supprimer sans risque au cas où ils seraient trop nombreux, ou bien il peut en rajouter si certains points n'ont pas été détectés.

10.3.4 Mise à jour des informations géologiques

Lorsque la topologie finale est obtenue à l'aide de l'étape précédente, nous devons mettre à jour les involutions γ_2 que nous avions précédemment laissées de côté. Ces involutions doivent permettre de relier les bords des horizons découpés à la topologie des failles. Cependant, comme les images par γ_2 des brins du bord d'un horizon désignent des brins se trouvant à l'intérieur des surfaces des failles, nous devons parcourir ces dernières afin de trouver des brins ayant une image dans le macro-modèle. De plus, les brins du bord des horizons ayant une image par γ_2 sont incidents à des macro-sommets. Nous devons alors relier ces brins aux cellules des failles correspondant aux macro-sommets.

Nous avons vu lors de la découpe d'un horizon par une faille (cf. section 10.2.4.3) que ces macro-sommets peuvent être de deux types. Soit ils correspondent à une macro-arête d'un réseau de faille, soit ils désignent un point d'intersection entre un bord original de l'horizon et une surface de faille. Ceci définit donc le type de la cellule devant être associée à chaque macro-sommet : une arête dans le premier cas et une face dans le second. Nous associons alors une information supplémentaire aux brins étant reliés par γ_2 en indiquant la dimension de la cellule (1 pour une arête et 2 pour une faille).

Pour retrouver un brin b_a de la macro-arête correspondant à un brin b_s d'un macro-sommet, nous procédons en trois étapes. Tout d'abord, nous recupérons le brin b_f de la faille F image par γ_2 du brin image par δ de b_s (*i.e.* $b_f = \gamma_2(\delta(b_s))$). Ensuite, comme ce brin appartient obligatoirement à un sommet incident au bord de la surface de F, nous parcourons ce sommet jusqu'à trouver un brin du bord b_b (*i.e.* un brin libre par α_2). Pour finir, nous parcourons le

bord de F depuis b_b jusqu'à atteindre un brin ayant une image par δ. L'image de ce brin est alors le brin b_a recherché.

Pour retrouver un brin de la macro-face correspondant à un brin b_s d'un macro-sommet, la méthode est plus simple. Il suffit simplement de parcourir les brins de la faille F depuis le micro-brin correspondant à $\gamma_2(\delta(b_s))$ jusqu'à trouver un brin de F ayant une image par δ.

Une fois ces traitemens terminés, le macro-modèle est complet et toutes les informations géolgiques stockées (les relations β_2 et γ_2 ainsi que les marques © et ◎) dans la micro-topologie deviennent inutiles et sont supprimées.

10.3.5 Résultats

Nous présentons ici les temps de calcul obtenus lors de la conversion des micro-modèles en macro-modèles. La machine utilisée ainsi que les modèles traités sont identiques à ceux présentés en section 10.2.6.

Modèles	ECPF	SC	MAJIG	Total
Normal Faults	5 s 321 ms	3 s 94 ms	0,3 ms	≈ 8 s
ResModOne	5 s 606 ms	4 s 6 ms	0,4 ms	≈ 10 s
Extension1	1 s 533 ms	1 s 793 ms	0,4 ms	≈ 3 s
Extension2	4 s 296 ms	2 s 434 ms	0,3 ms	≈ 7 s
N'Kossa	10 s 886 ms	9 s 176 ms	0,8 ms	≈ 20 s

ECPF : extraction des contours et plongement des faces
SC : simplification des contours
MAJIG : mise à jour des informations géologiques (relations γ_2)

Tab. 10.2 – Temps de calcul obtenus pour la conversion de micro-modèles en macro-modèles.

Comme nous pouvons le voir, les temps de calcul obtenus pour convertir un micro-modèle en macro-modèle sont négligeables par rapport aux temps de construction du micro-modèle. Cependant, à l'avenir, nous souhaitons pouvoir passer le plus rapidement possible d'un micro-modèle à un macro-modèle (et inversement) afin de rendre l'application interactive. Nous discutons des différentes optimisations envisageables en section 10.4.

10.4 Conclusion et perspectives

Nous avons présenté dans ce chapitre différents outils permettant de construire automatiquement des modèles structuraux 3D. Les résultats obtenus sont satisfaisants et sont encourageants pour la suite. Cependant, nous avons vu que certaines parties du processus de construction sont encore coûteuses en temps de calcul mais peuvent être optimisées.

Nous avons vu en section 10.2.6 que la majeure partie du temps de calcul est utilisée dans des opérations topologiques. Certaines d'entre elles, comme par exemple l'orientation des surfaces, sont spécifiques au modèle de représentation utilisé (G-Cartes) et pourraient être évitées en passant par exemple sur un modèle de cartes combinatoires. Ce passage à un autre modèle n'aurait

pas que pour seul effet la suppression de certaines tâches coûteuses en temps, il permettrait aussi d'accélérer la totalité des calculs et réduirait l'occupation mémoire au moins de moitié.

Une autre partie importante du temps est passée dans les calculs d'intersection. Cependant, nous avons vu dans les conclusions et perspectives de cet algorithme (cf. section 7.6) qu'il était possible d'accélérer encore la recherche des intersections, ce qui pourrait vraisemblablement diminuer significativement les temps de calcul.

De plus, les calculs d'intersection entre les zones d'incertitude et les horizons peuvent être évités. En effet, cette étape sert uniquement à délimiter une région d'incertitude sur les horizons et cette dernière n'a pas besoin d'être aussi précise. Nous envisageons donc de développer un algorithme spécifique permettant de détecter grossièrement les faces des horizons étant intersectées par les zones d'incertitudes mais sans calculer précisément ces intersections. Les faces trouvées permettraient ainsi de délimiter les régions d'incertitude sur les horizons en seulement quelques secondes.

En ce qui concerne la conversion d'un micro-modèle en macro-modèle, nous avons vu en section 10.3.5 que les résultats était assez satisfaisants. Cependant, ces derniers peuvent encore être améliorés. Tout d'abord, nous avons vu que la détection des macro-sommets géométriques était une étape facultative et qu'elle ne donnait pas toujours les résultats escomptés (cf. section 10.3.3). De plus, cette étape nécessitant de nombreux calculs géométriques, elle est beaucoup plus coûteuse que la détection des macro-sommets topologiques. Ainsi 90% du temps de calcul utilisé pour la détection des macro-sommets est consacré à la détection des macro-sommets géométriques. Cette étape peut alors être supprimée afin de laisser l'utilisateur choisir les macro-sommets qui lui conviennent.

Dans un deuxième temps, pour mettre à jour un macro-modèle en y insérant de nouveaux éléments ou en réalisant de nouvelles intersections, il n'est pas nécessaire d'effectuer des conversions sur l'intégralité du modèle. Nous pouvons alors uniquement convertir les objets concernés par les modifications et réduire le temps de conversion à seulement une ou deux secondes.

Conclusion

Le but de cette thèse était de concevoir des outils permettant la construction de modèles géologiques structuraux topologiquement consistants, et ce en dimension trois. Pour cela, nous devions définir une méthode d'assemblage des surfaces qui puisse prendre en compte des configurations géologiques particulières, comme par exemple les topologies liées aux failles pendantes (surfaces incidentes à un seul volume). De plus, l'interprétation du géologue étant à la base de toute conception de modèle géologique, elle devait être prise en compte lors du processus de construction. Ainsi, les outils devaient permettre de maintenir une topologie cohérente avec l'interprétation du géologue à chaque étape de construction.

Pour atteindre cet objectif, nous avions à notre disposition un modeleur topologique basé sur un noyau de cartes généralisées. Nous avons alors étudié un premier algorithme de co-raffinement 2D sur ce modèle, afin de nous familiariser avec les différents problèmes pouvant être rencontrés. Cet algorithme est basé sur une technique de propagation dont l'intérêt réside dans l'utilisation de la topologie des objets maillés afin d'effectuer des tests d'intersection locaux. Il en résulte un algorithme performant dont la complexité moyenne est $\mathcal{O}(n)$, où n correspond au nombre de segments composant chaque maillage.

Au vu des résultats obtenus, nous avons décidé d'utiliser cette technique pour réaliser un algorithme de co-raffinement en dimension trois. L'algorithme obtenu permet de générer localement les lignes de coupes existantes entre deux objets en se propageant sur la surface des volumes. De plus, cet algorithme permet, contrairement aux nombreux algorithmes déjà existants, de pouvoir calculer des intersections entre des objets ayant une topologie en dimension trois (*i.e.* objets composés de plusieurs volumes adjacents). Cet algorithme a cependant posé de nombreux problèmes aussi bien du point de vue des erreurs numériques, que des problèmes algorithmiques et des performances.

Nous avons alors réalisé un nouvel algorithme de co-raffinement 3D basé sur une méthode consistant à calculer tous les segments d'intersection existant entre deux faces isolées puis d'effectuer cette opération pour chaque face des objets. Nous avons résolu les nombreux problèmes algorithmiques posés par le précédent algorithme et avons réussi à traiter tous les cas de figure pouvant être rencontrés. De plus, à l'aide d'une structure accélératrice permettant de limiter les tests d'intersection, nous avons réussi à obtenir un algorithme beaucoup plus performant que le précédent.

À l'aide de ce dernier algorithme, nous avons pu concevoir différents outils permettant de calculer les intersections entre des surfaces géologiques. Ces outils permettent de contrôler finement la topologie de la scène géologique tout au long du processus de construction et de

s'assurer que la topologie obtenue correspond bien à l'interprétation du géologue. Pour stocker toutes les informations géologiques et topologiques nécessaires à la représentation d'une scène géologique, nous avons alors conçu une structure de données basée sur des cartes généralisées. Cette structure se décompose en deux niveaux de topologie permettant de faciliter aussi bien les traitements liés à la construction des modèles (micro-topologie) que ceux liés à la récupération d'informations (macro-topologie).

Tous ces outils sont en cours d'intégration dans le pilote géologique 3D à l'Institut Français du Pétrole. Ils peuvent donc être utilisés dans la chaîne de traitement complète et peuvent être pilotés directement à partir d'un Schéma d'Évolution Géologique. Ainsi, plusieurs modèles ont déjà été traités avec succès et certains d'entre eux sont présentés dans les annexes se trouvant à la fin de ce mémoire.

Malgré des résultats prometteurs, nous avons pu constater que certains points pouvaient encore être améliorés. Le dernier algorithme de co-raffinement en dimension trois peut être optimisé en modifiant la méthode employée pour déterminer les couples de faces à tester. Nous envisageons de modifier la boucle principale afin de travailler directement sur la grille régulière pour ne tester les intersections qu'entre les faces partageant une même case. De plus, nous avons privilégié la partie algorithmique et topologique de l'opération en laissant de côté l'aspect numérique. À l'avenir, nous souhaitons donc approfondir ce dernier point afin de rendre l'algorithme plus robuste.

Concernant la construction de modèles géologiques, nous avons vu qu'une grande partie des temps de calcul était dévolue à la gestion de la structure topologique. Nous souhaitons donc passer à une structure plus légère qui serait spécifique à la modélisation géologique. Le noyau topologique avec lequel nous avons travaillé est basé sur des cartes généralisées et est défini de manière à être le plus générique possible. Nous souhaitons alors concevoir un noyau basé sur des cartes combinatoires qui posséderait plusieurs optimisations permettant le stockage et la récupération d'informations à moindre coût en évitant au maximum les parcours topologiques.

Pour finir, nous souhaitons de plus utiliser les résultats obtenus pour les appliquer à l'étape suivante de la chaîne de traitement, c'est-à-dire la génération de maillages stratigraphiques. Pour cela, nous voulons utiliser la topologie du modèle structural afin de déterminer automatiquement la structure du maillage. Nous pouvons ainsi utiliser la topologie des horizons afin de déterminer le long de quelles mailles doivent être découpés les maillages et utiliser les relations géologiques entre les horizons et les failles afin de définir des piliers principaux.

Annexe A

Quelques exemples de co-raffinement et d'opérations booléennes

Nous présentons ici des exemples d'objets construits à l'aide du co-raffinement et des opérations booléennes. Ces exemples vont des simples opérations d'assemblage de primitives géométriques à l'aide d'opérations booléennes (cf. Fig. A.1, A.2 et A.3), à la création d'une scène géologique synthétique[1] montrant la puissance du co-raffinement (cf. Fig. A.4).

[1]La scène géologique présentée ici est purement fictive et n'a pas la prétention de vouloir représenter une quelconque géologie réaliste.

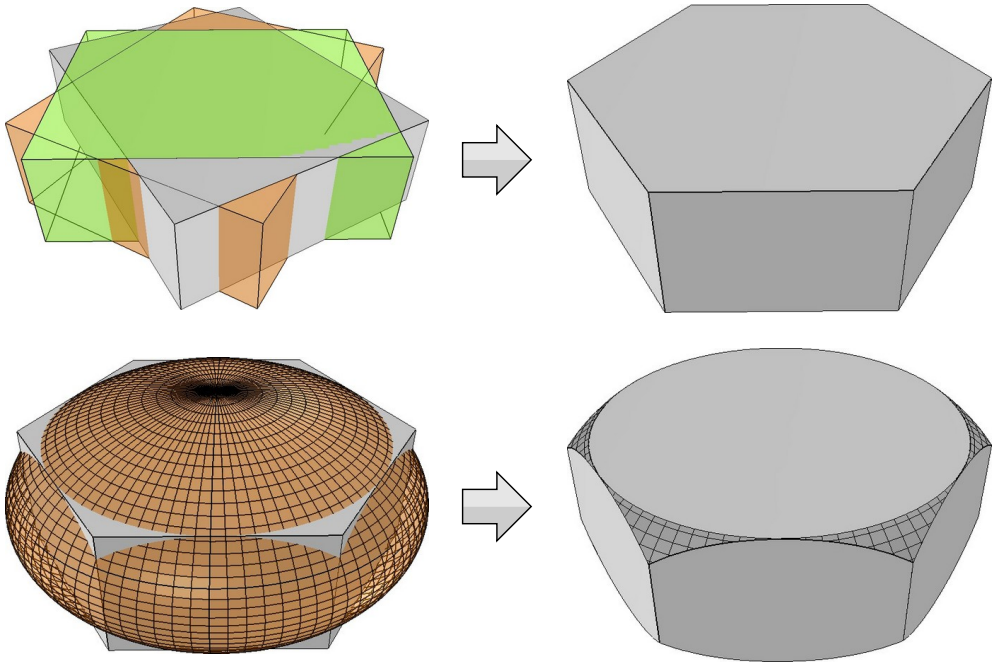


FIG. A.1 – Exemple de construction d'une tête de vis à l'aide d'opérations booléennes.

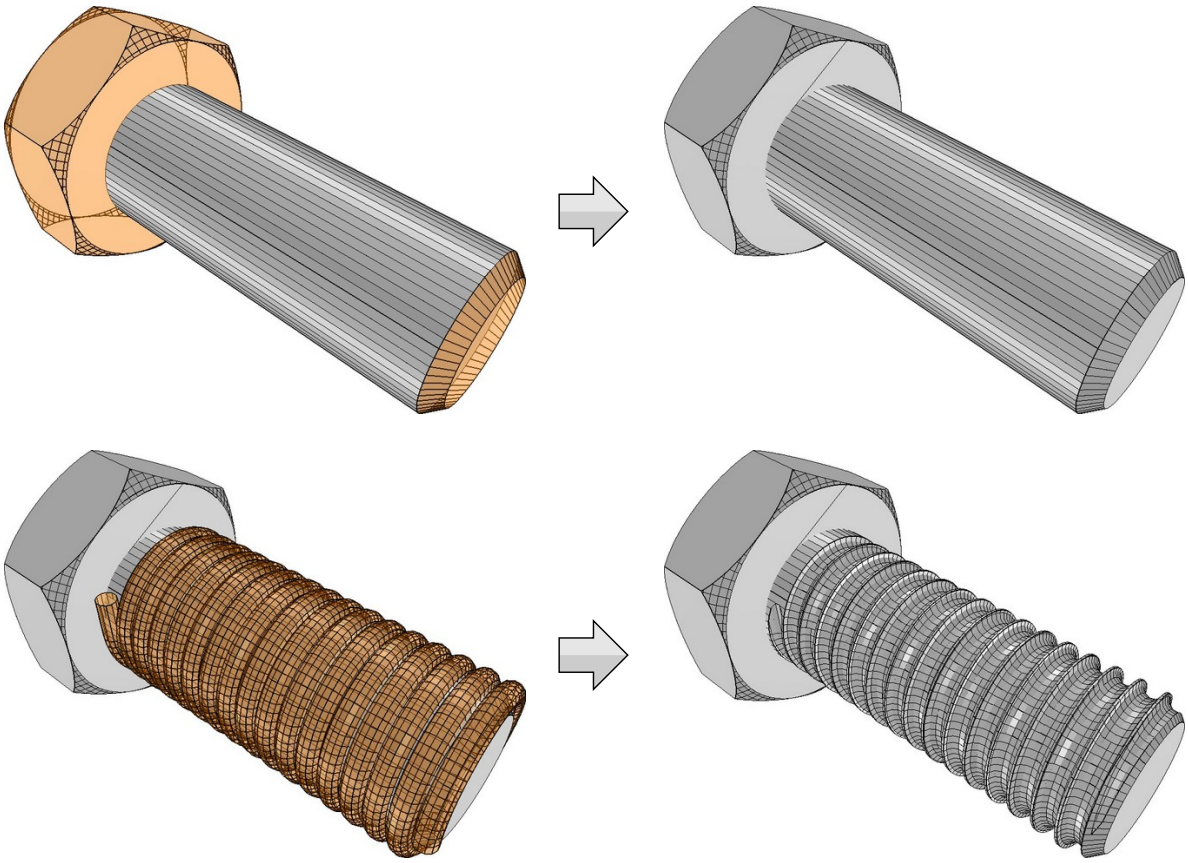


FIG. A.2 – Exemple de construction d'une vis à l'aide d'opérations booléennes.

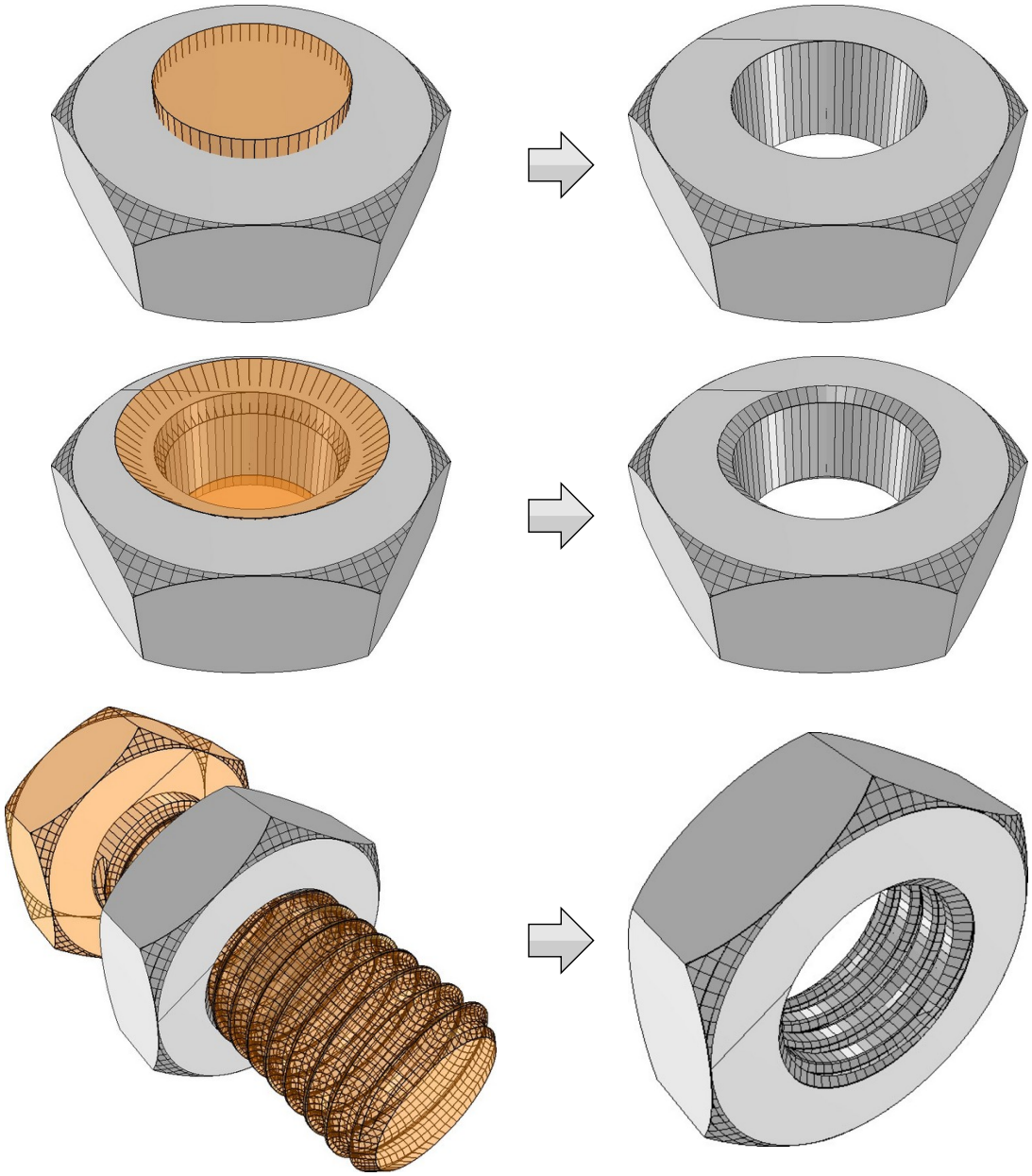


FIG. A.3 – Exemple de construction d'un écrou à l'aide d'opérations booléennes.

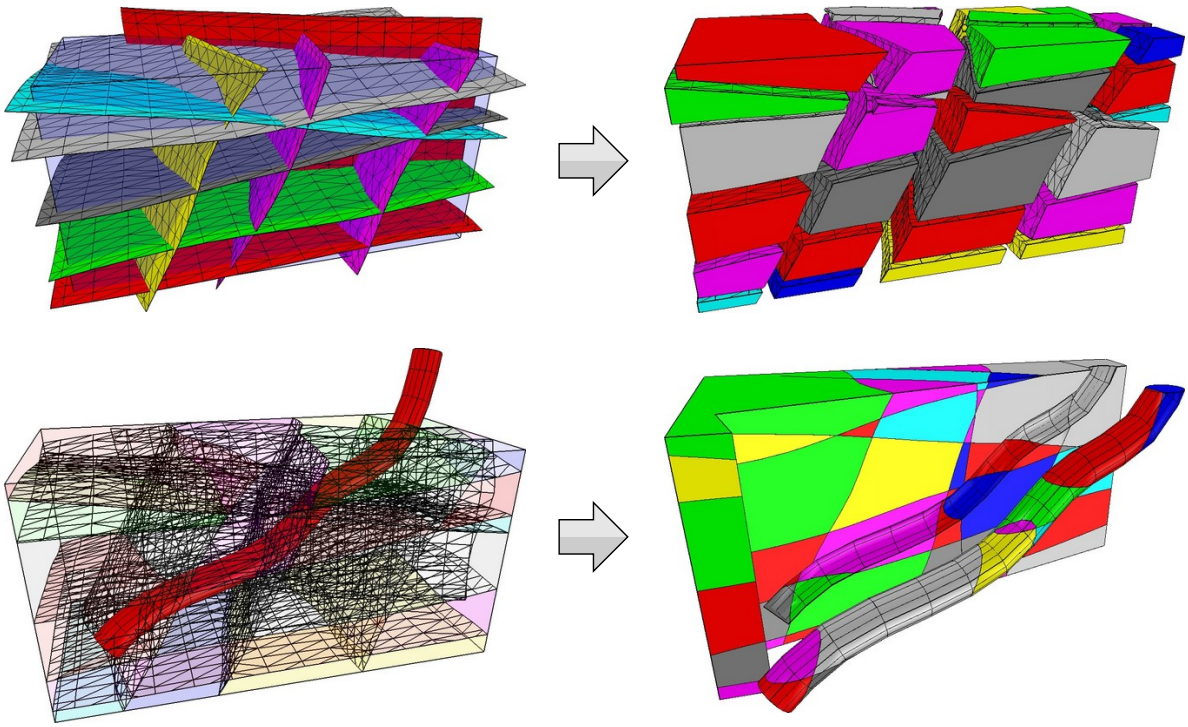


FIG. A.4 – Exemple de construction d'un modèle géologique synthétique à l'aide du co-raffinement et forage de ce modèle à l'aide d'opérations booléennes.

Annexe B

Modèles géologiques traités

B.1 Le modèle « NormalFaults »

Le modèle « NormalFaults » est un modèle réaliste confié par la société Beicip-Franlab et qui sert de tutorial pour la bibliothèque RML. Il se compose de deux horizons, deux failles isolées et un réseau de failles composé de trois failles. Les figures B.1 et B.2 présentent respectivement les différentes surfaces du modèle avant construction et le résultat obtenu après traitement.

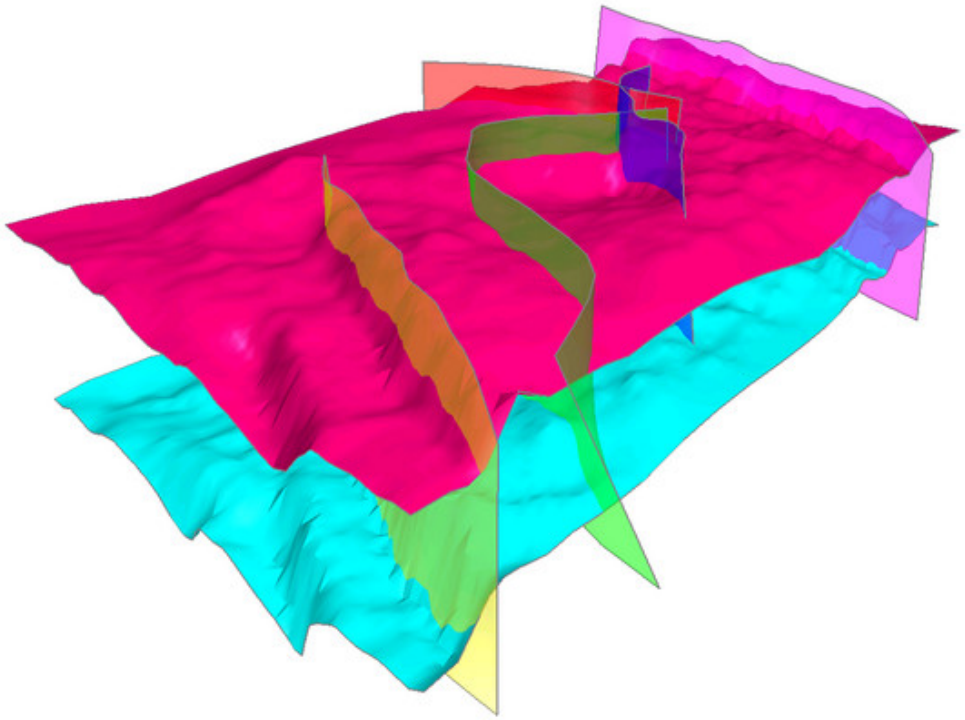


FIG. B.1 – Les surfaces originales du modèle « Normal Faults ».

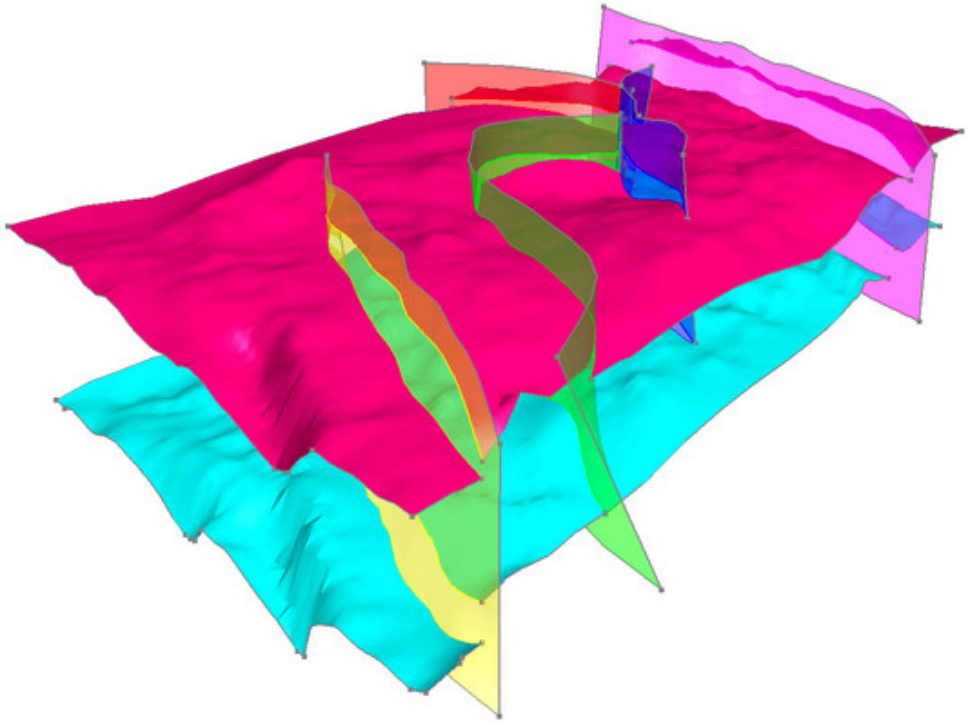


FIG. B.2 – Le modèle « Normal Faults » reconstruit.

FIG. B.3 – Un horizon du modèle « Normal Faults » reconstruit.

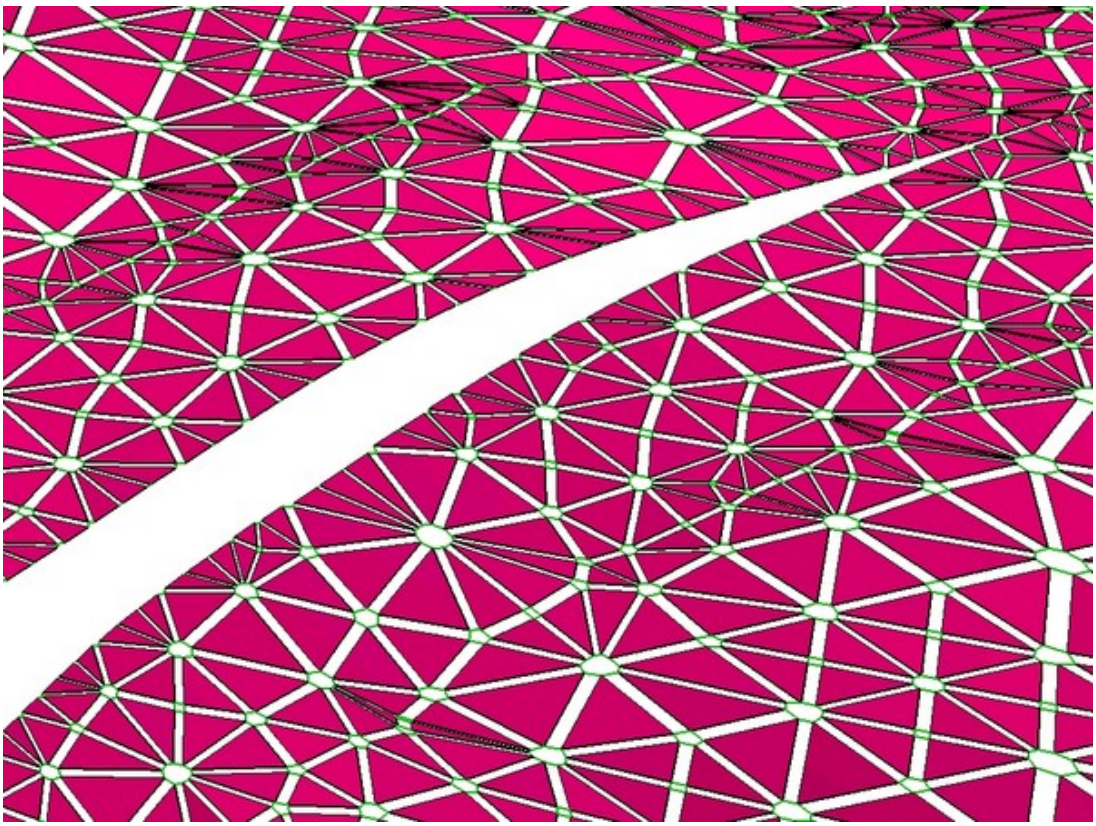


FIG. B.4 – Détail de la micro-topologie d'un horizon du modèle « Normal Faults » au voisinage d'une découpe par une faille.

B.2 Le modèle « ResModOne »

Le modèle « ResModOne » est aussi un modèle réaliste qui nous a été confié par Géocap et IRAP/RMS. Il s'agit du modèle servant de tutoriel au logiciel ROXAR. Il se compose de cinq horizons, deux failles isolées et un réseau de failles composé de deux failles. Les figures B.5 et B.6 présentent respectivement les différentes surfaces du modèle avant construction et le résultat obtenu après traitement.

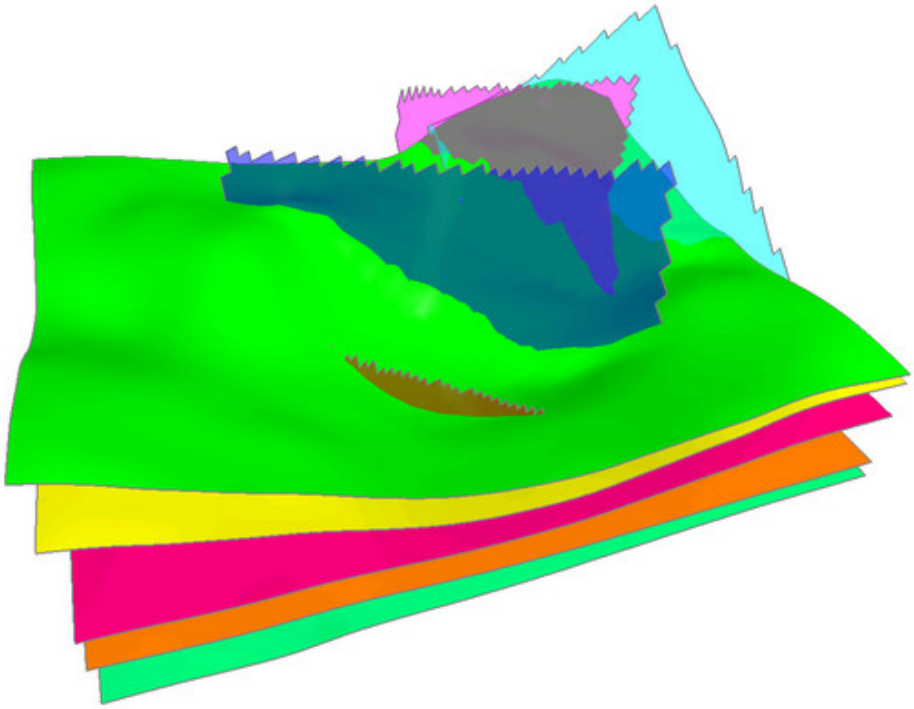


Fig. B.5 – Les surfaces originales du modèle « ResModOne ».

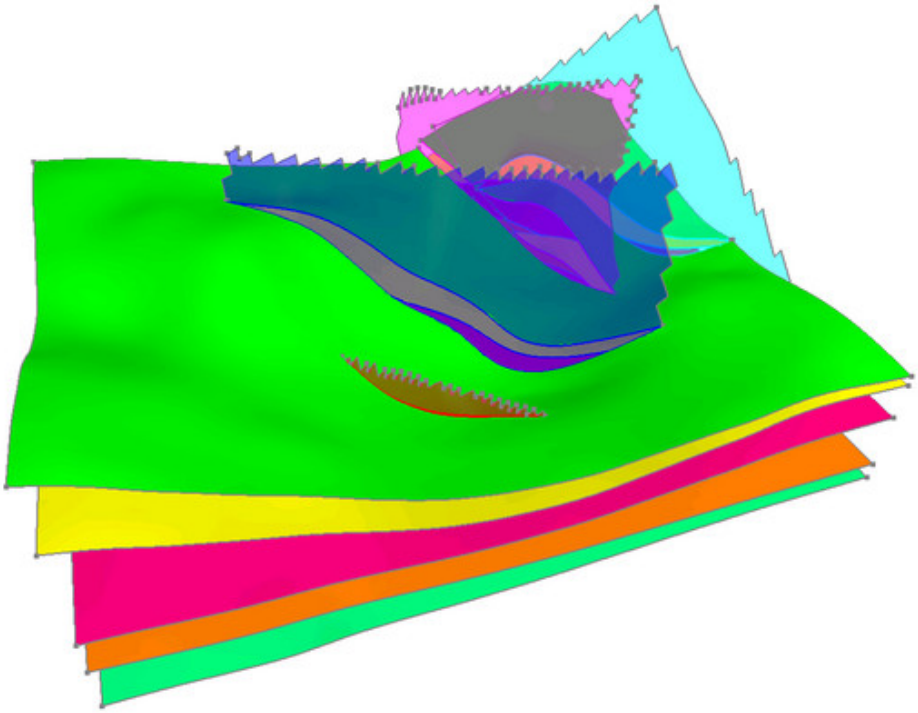


FIG. B.6 – Le modèle « ResModOne » reconstruit.

FIG. B.7 – Un horizon du modèle « ResModOne » reconstruit.

B.3 Le modèle « Extension1 »

Le modèle « Extension1 » est un modèle synthétique issu d'une simulation en bac à sable. Ce genre de simulation consiste à remplir un bac avec des sables de différentes natures de manière à modéliser des couches géologiques. Les parois du bac sont ensuite déplacées très lentement de manière à simuler des phénomènes géologiques. Dans le cas présent, les parois du bac ont été rapprochées afin de comprimer le modèle. Le modèle à ensuite été scanné par l'IFP[1] puis les différentes surfaces ont été extraites.

Ce modèle se compose de deux horizons et d'un seul réseau de failles composé de six failles. Les figures B.8 et B.9 présentent respectivement les différentes surfaces du modèle avant construction et le résultat obtenu après traitement.

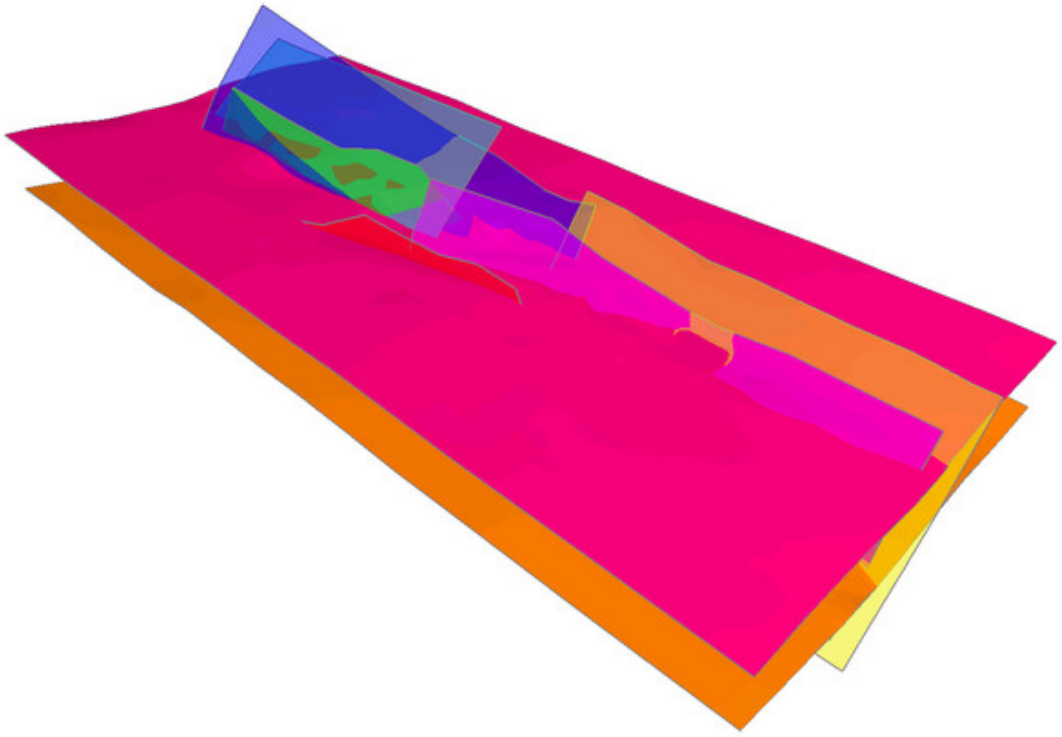


FIG. B.8 – Les surfaces originales du modèle « Extension1 ».

[1]Projet concerné : `http://www.ifp.fr/IFP/fr/rechercheindustrie/explorationproduction/Facet4D.htm`

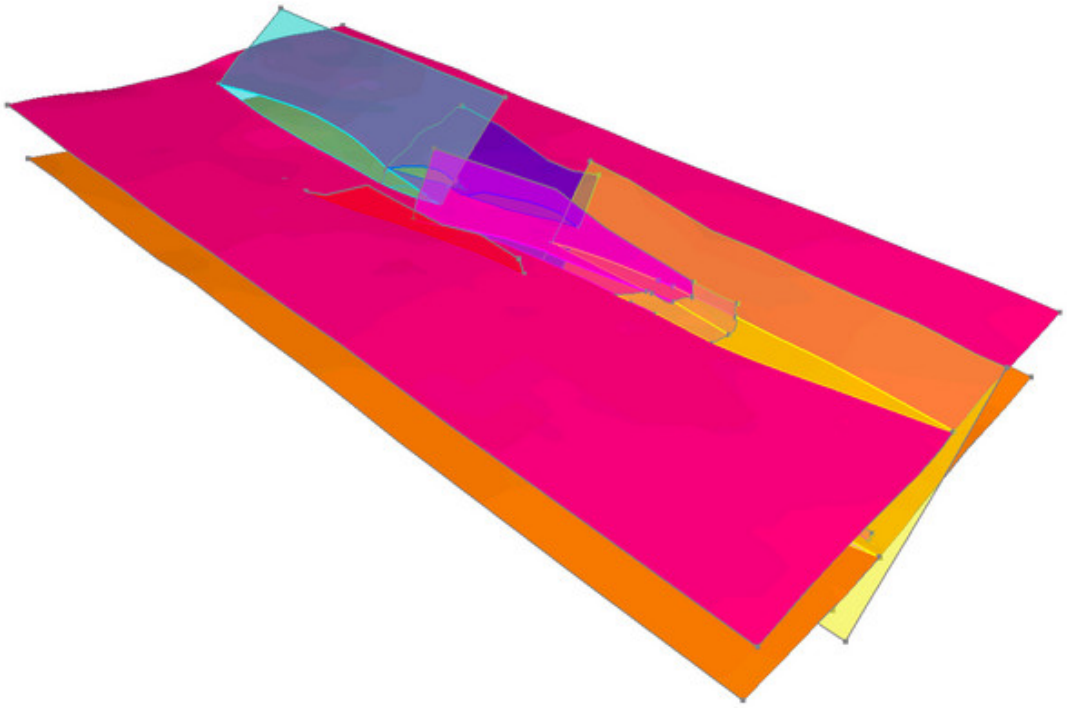


FIG. B.9 – Le modèle « Extension1 » reconstruit.

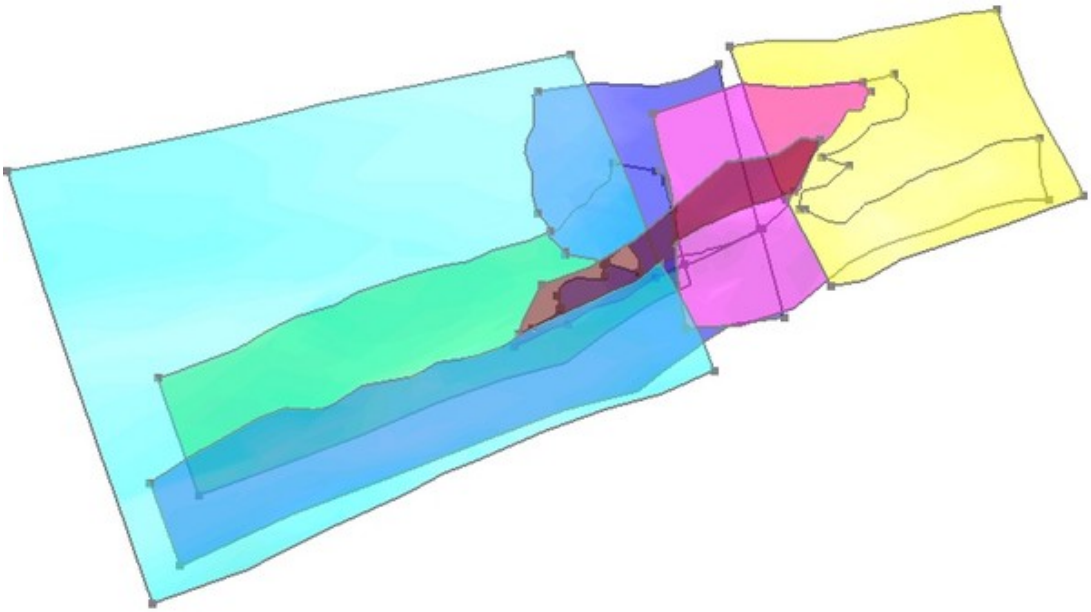


FIG. B.10 – Le réseau de failles du modèle « Extension1 ».

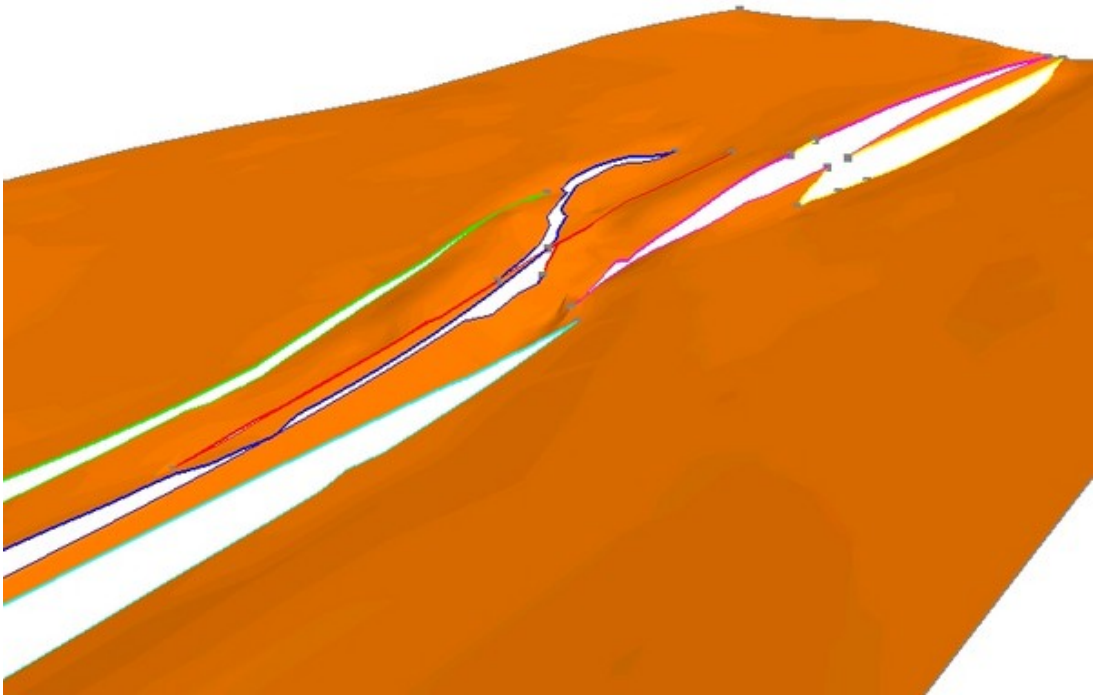


FIG. B.11 – Un horizon du modèle « Extension1 » reconstruit.

B.4 Le modèle « Extension2 »

Le modèle « Extension2 » est aussi un modèle synthétique obtenu par le même procédé que le modèle « Extension1 ». Il se compose de trois horizons et d'un seul réseau de failles composé de cinq failles. Les figures B.12 et B.13 présentent respectivement les différentes surfaces du modèle avant construction et le résultat obtenu après traitement.

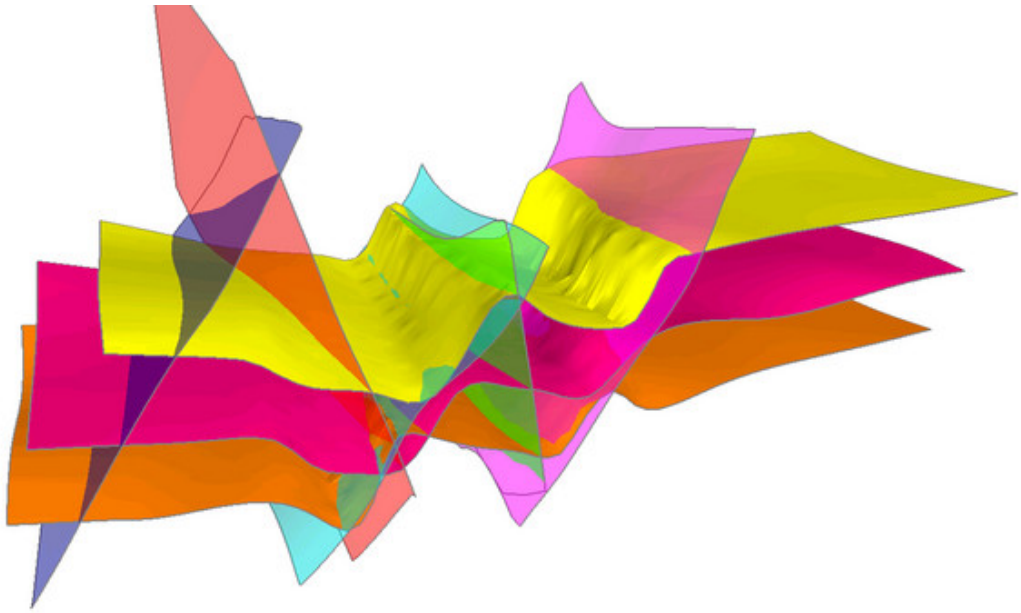


FIG. B.12 – Les surfaces originales du modèle « Extension2 ».

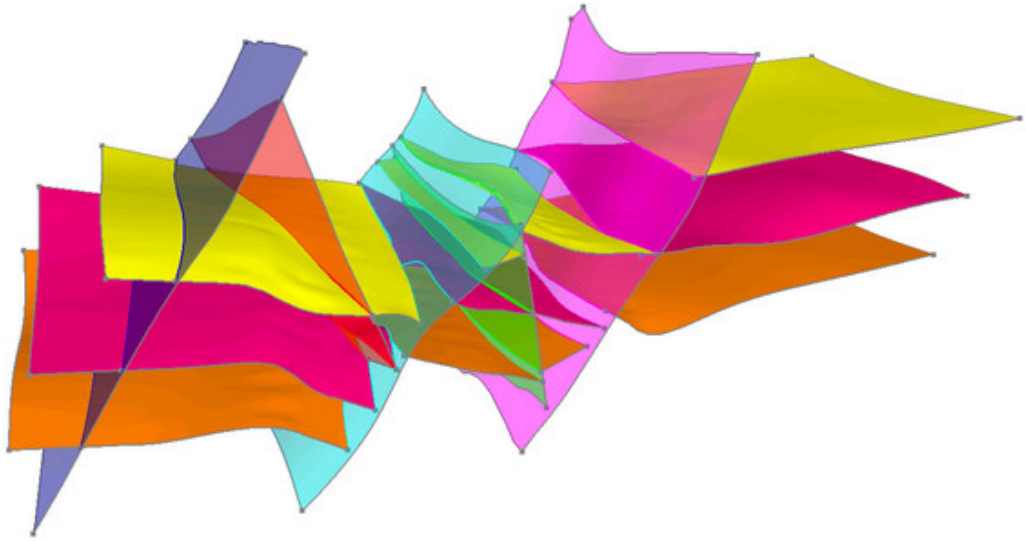


FIG. B.13 – Le modèle « Extension2 » reconstruit.

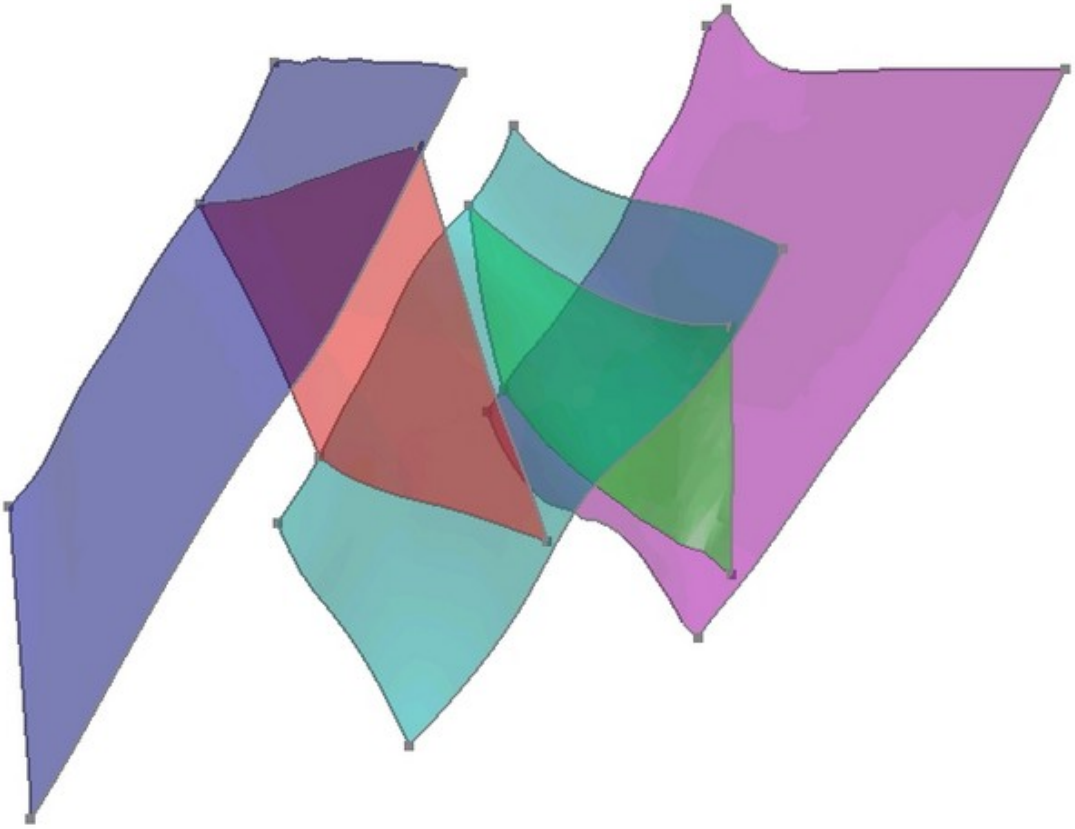


FIG. B.14 – Le réseau de failles du modèle « Extension2 ».

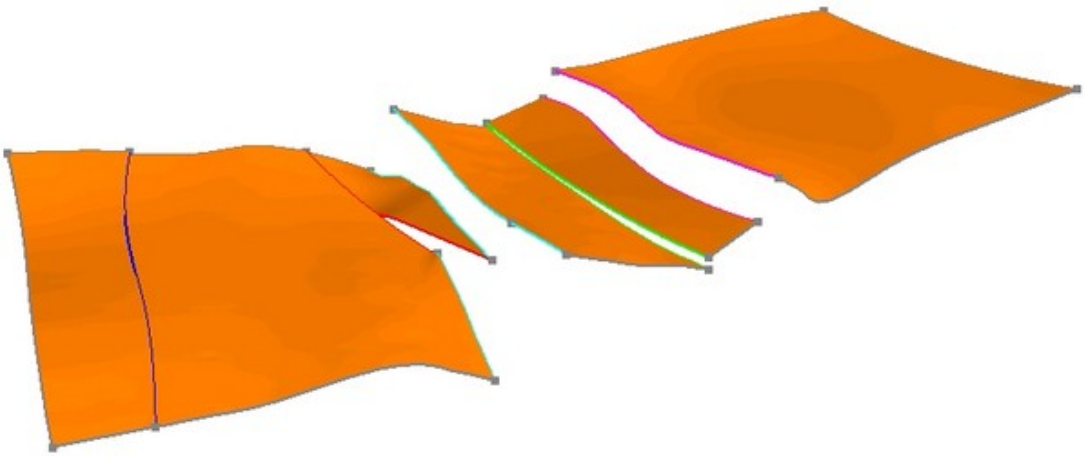


FIG. B.15 – Un horizon du modèle « Extension2 » reconstruit.

B.5 Le modèle « N'Kossa »

Le modèle « N'Kossa » a été confié à l'IFP par la société ELF entre les années 1996 et 1998, dans le cadre du projet européen THERMIE (106OG94). Ce projet à fait partie d'un projet commun entre les sociétés Beicip-Franlab et ELF Aquitaine, dont le titre est : *3D modeling chain for reservoir engineering.*

Ce modèle se compose de trois horizons, de trois failles isolées et de deux réseaux de failles composés respectivement de cinq et quatre failles. Les figures B.16 et B.17 présentent respectivement les différentes surfaces du modèle avant construction et le résultat obtenu après traitement.

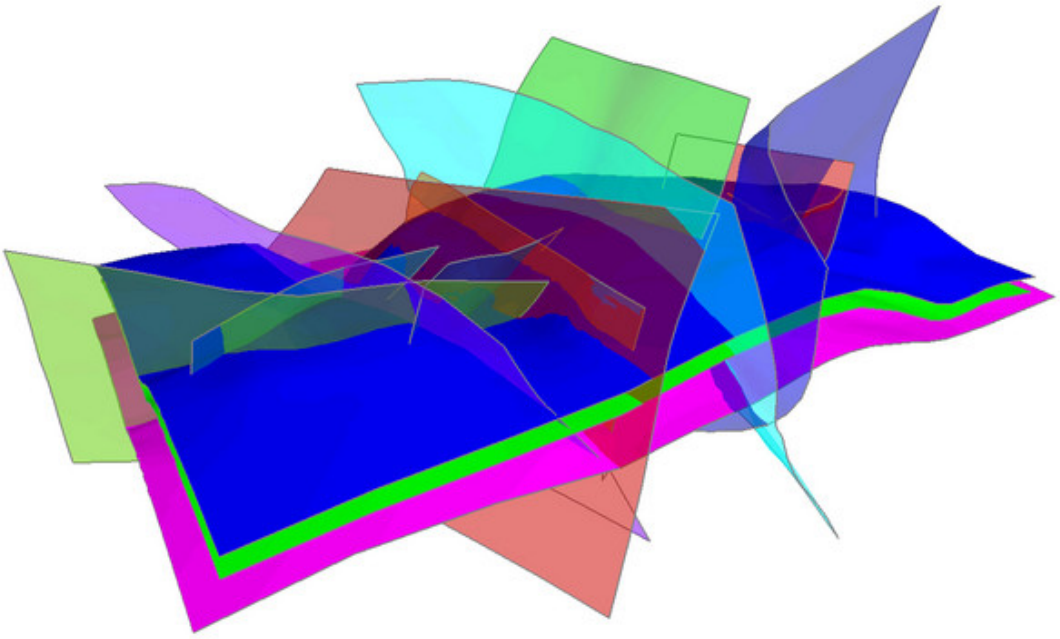


FIG. B.16 – Les surfaces originales du modèle « N'Kossa ».

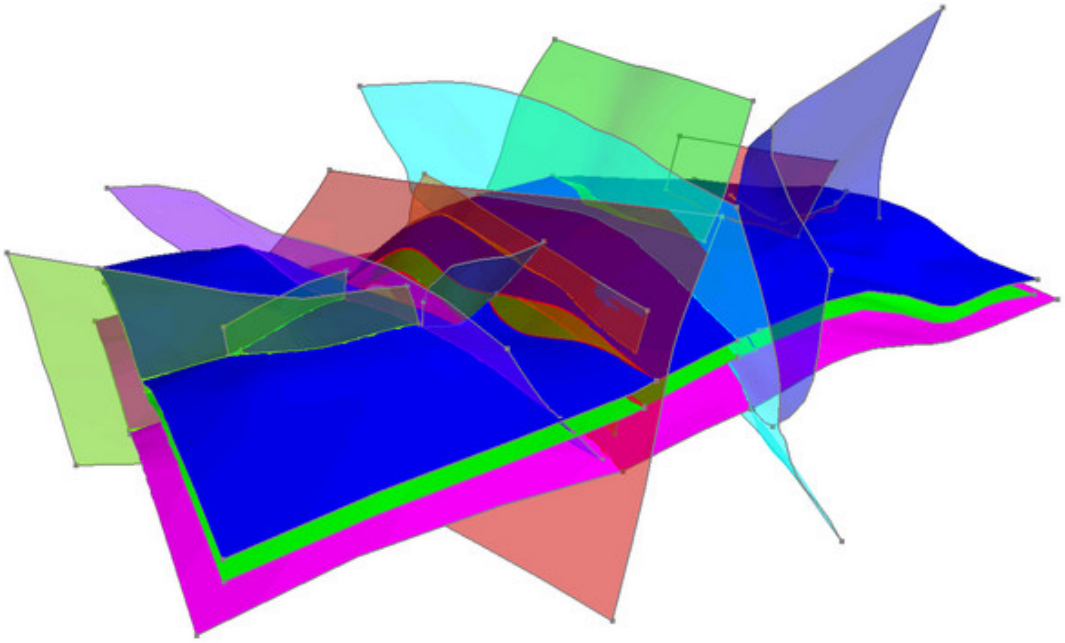


FIG. B.17 – Le modèle « N'Kossa » reconstruit.

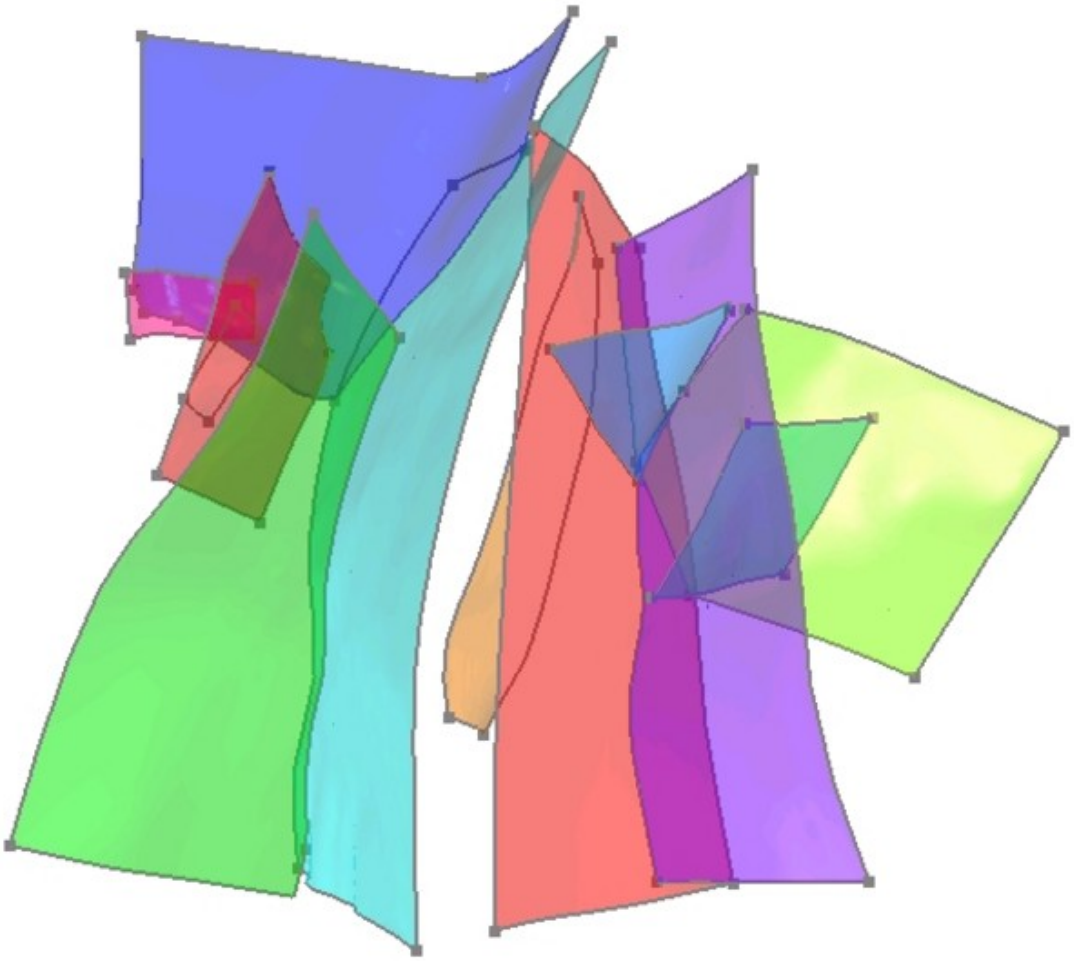


FIG. B.18 – Les réseaux de failles du modèle « N'Kossa » vus du dessus.

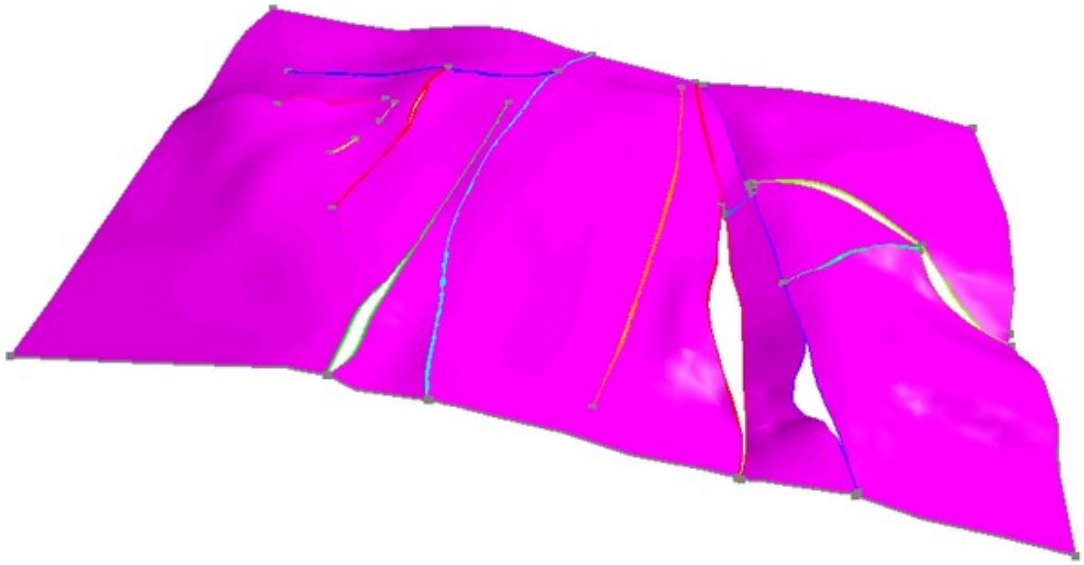


FIG. B.19 – Un horizon du modèle « N'Kossa » reconstruit.

Bibliographie

[BF97] Y. Bertrand and J. Françon. Variétés combinatoires 2d pour la représentation d'objets par leur bord : étude statistique des cellules. Rapport technique, Université Louis-Pasteur, LSIIT, Strasbourg, 1997.

[BSP+04] S. Brandel, S. Schneider, M. Perrin, N. Guiard, J.-F. Rainaud, P. Lienhardt, and Y. Bertrand. Automatic building of structured geological model. In *ACM Symposium on Solid Modelling and Applications*, Genoa (Italy), June 2004.

[BSP+05] S. Brandel, S. Schneider, M. Perrin, N. Guiard, J.-F. Rainaud, P. Lienhardt, and Y. Bertrand. Automatic building of structured geological models. *Journal of Computing & Information Science & Engineering*, 5(2) :138–148, June 2005.

[BY95] J.-D. Boissonnat and M Yvinec. *Géométrie algorithmique*. Ediscience international, 1995.

[Caz97] D. Cazier. *Construction de systèmes de réécriture pour les opérations booléennes en modélisation géométrique*. PhD thesis, Université Louis Pasteur de Strasbourg, 1997.

[CD96] D. Cazier and J.-F. Dufourd. Rewriting-based derivation of efficient algorithms to build planar subdivisions. In *12th Spring Conference on Computer Graphics*, pages 45–54, 1996.

[CD99] D. Cazier and J.-F. Dufourd. A formal specification of geometric refinements. *The Visual Computer*, 15 :279–301, 1999.

[dBDDS97] M. de Berg, O. Devillers, K. Dobrindt, and O. Schwarzkopf. Computing a single cell in the overlay of two simple polygons. *Information Processing Letters*, 63(4) :215–219, 1997.

[FR88] A. Foucault and J.F. Raoult. *Dictionnaire de géologie*. Masson, 3ème edition, 1988.

[GHPT89] M. Gangnet, J.-C. Hervé, T. Pudet, and J.-M. Van Thong. Incremental computation of planar maps. *Computer Graphics*, 23(3) :345–354, 1989.

[GP96] Y. Gardan and E. Perrin. An algorithm reducing 3d boolean operations to a 2d problem : concepts and results. *Computer Aided Design*, 28(4) :277–287, 1996.

[HHK88] C.M. Hoffmann, J.E. Hopcroft, and M.S. Karasick. Robust set operations on polyhedral solids. *IEEE Computer Graphics & Applications*, 9 :55–59, 1988.

[HMW76] B.E. Hobbs, W.D. Means, and P.F. Williams. *An Outline of Structural Geology*. Wiley and sons, 1976.

[KY92] K. Kitajima and M. Yamaguchi. A shell-oriented boolean set operation algorithm suited for the b-reps based on boundary edge loops. *Systems and Computers in Japan*, 23(6) :94–111, 1992.

[Lie91] P. Lienhardt. Topological models for boundary representation : a comparison with n-dimensional generalized maps. *Computer Aided Design*, 23(1) :59–82, 1991.

[Lie94] P. Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal of Computational Geometry and Applications*, 4(3) :275–324, 1994.

[Mal89] J.-L. Mallet. Discrete smooth interpolation in geometric modeling. *ACM Transactions on Graphics*, 8(2) :121–144, 1989.

[Mal92] J.-L. Mallet. Gocad : a computer-aided design program for geological applications. *Three-Dimensional Modeling with Geoscientific Information Systems*, 354 :123–142, 1992.

[MK89] A. Margalit and G.D. Knott. An algorithm for computing the union, intersection or difference of two polygons. *Computer & Graphics*, 13(2) :167–184, 1989.

[MT83] M. Mantyla and M. Tamminen. Localized set operations for solid modeling. *Computer & Graphics*, 13(3), 1983.

[MT88] D. Ma and R. Tang. Realizing the boolean operations in solid modeling technique via directed loops. *Computer & Graphics*, 12(3/4), 1988.

[NP82] J. Nievergelt and F.P. Preparata. Plane-sweep algorithms for intersecting geometric figures. *Communications of the ACM*, 25(10) :739–747, 1982.

[Per97] M. Perrin. éléments de syntaxe géologique en vue de la construction de modèles 3d de terrains. Rapport technique, École Nationale Supérieure des Mines de Paris, Centre de Géologie de l'Ingénieur, 1997.

[Per98] M. Perrin. Geological consistency : an opportunity for safe surface assembly and quick model exploration. *3D Modeling of Natural Objects, A Challenge for the 2000's*, 3(4-5), June 1998.

[PS86] L.K. Putnam and P.A. Subrahmanyam. Boolean operations on n-dimensional objects. *IEEE Computer Graphics & Applications*, 6 :43–51, 1986.

[RF00] M. Rivero and F. R. Feito. Boolean operations on general planar polygons. *Computer & Graphics*, 24(6) :881–896, 2000.

[RPB05] J.-F. Rainaud, M. Perrin, and Y. Bertrand. Innovative knowledge-driven approach for shared earth model building. In *Joint SPE/EAGE Conference*, Madrid, June 2005. Extended Abstract SPE 94172 PP.

[Sch02] S. Schneider. *Pilotage automatique de la construction de modèles géologiques surfaciques*. PhD thesis, Université Jean-Monnet et École des Mines de Saint-Etienne, 2002.

[SPG+04] S. Schneider, M. Perrin, N. Guiard, J.-F. Rainaud, P. Lienhardt, and Y. Bertrand. Methodology for automatic building of structured geological models. In *Conference GEOMOD04*, Emetten (Switzerland), June 2004.

[TN87] W.C. Thibault and B.F. Naylor. Set operations on polyhedra using binary space partitioning trees. *Computer & Graphics*, 21(4) :153–162, 1987.

[Ž00] B. Žalik. Two efficient algorithms for determining intersection points between simple polygons. *Computer & Geosciences*, 26(2) :137–151, 2000.