Une nouvelle approche pour les opérations booléennes : formalisation et mise en oeuvre

Estelle Perrin

THÈSE

Présentée à

l'UNIVERSITÉ de METZ

Pour l'obtention du grade de :
DOCTEUR de l'UNIVERSITÉ de METZ

Spécialité : INFORMATIQUE

Estelle PERRIN

UNE NOUVELLE APPROCHE POUR LES OPÉRATIONS BOOLÉENNES : FORMALISATION ET MISE EN ŒUVRE.

Soutenue à Metz le 19 octobre 1995

Composition du jury :

Directeur de thèse : Yvon GARDAN (Professeur à l'Université de Metz)

Rapporteurs : Jean-Paul HATON (Professeur à l'Université de Nancy I)
Michel LUCAS (Professeur à l'École Centrale de Nantes)

Examinateurs : Jean-Pierre JUNG (Professeur à l'Université de Metz)
Maurice MARGENSTERN (Professeur à l'Université de Metz)

...FORMATIQUE DE METZ

THÈSE

Présentée à

l'UNIVERSITÉ de METZ

Pour l'obtention du grade de :
DOCTEUR de l'UNIVERSITÉ de METZ

Spécialité : INFORMATIQUE

Estelle PERRIN

UNE NOUVELLE APPROCHE POUR LES OPÉRATIONS BOOLÉENNES : FORMALISATION ET MISE EN ŒUVRE.

Soutenue à Metz le 19 octobre 1995

Composition du jury :

Directeur de thèse : Yvon GARDAN (Professeur à l'Université de Metz)

Rapporteurs : Jean-Paul HATON (Professeur à l'Université de Nancy I)
Michel LUCAS (Professeur à l'École Centrale de Nantes)

Examinateurs : Jean-Pierre JUNG (Professeur à l'Université de Metz)
Maurice MARGENSTERN (Professeur à l'Université de Metz)

LABORATOIRE DE RECHERCHE EN INFORMATIQUE DE METZ

A mes parents, merci pour tout et pour beaucoup.
A Benoit, ton amour, ton soutien et la confiance que tu me portes sont mon entrain et ma résistance.

Ne me dites pas que ce problème est difficile. S'il n'était pas difficile,
ce ne serait pas un problème.
Maréchal F. Foch

REMERCIEMENTS.

Mes premiers remerciements sont, naturellement, pour mon directeur de thèse Yvon Gardan. Ce manuscrit est l'occasion, pour moi, de le remercier encore de la confiance qu'il m'a témoignée en m'acceptant dans son laboratoire. Je tenais tout particulièrement à souligner la disponibilité d'esprit et de temps dont il a fait preuve tout au long de ces trois années, malgré les nombreuses fonctions dont il a la charge. Se sentir correctement encadrée et savoir qu'il existe quelqu'un pour nous écouter quand un problème survient, sont, pour ma part, des facteurs non négligeables intervenant dans la motivation à mener à bien une thèse.

Ensuite, je tiens à remercier tous ceux qui ont accepté de juger mon travail. Je pense naturellement à mes deux rapporteurs Jean-Paul Haton et Michel Lucas ainsi qu'à mes deux examinateurs Jean-Pierre Jung et Maurice Margenstern.

Je tiens également à exprimer mon amitié à l'ensemble des personnes du Laboratoire de Recherche en Informatique de Metz : *Cathy*, *Isabelle*, *Sandrine et Frédéric :* À tous les quatre, merci pour nos fous rires partagés ; ainsi que : *Jean-Pierre*, *Yann, Benoit*, *Christian*, *Myriam*, *Robin*, *Kamel*, *Stéphane*, *Pascal*, *Abdou*, *Lynda*, *Martine*, *Catherine*, *Jocelyne*, *Dominique*, ...

Mes pensées vont aussi à tous mes amis d'études qui sont restés en contact avec moi, *Pierre et Sandrine*, *Michel*, *Jean-Marie*, *Laurent*, *Jean-Luc*, ...

A tous ceux que je viens de nommer et à tous ceux que j'ai oubliés, merci.

Estelle

SOMMAIRE.

Chapitre 2. Opérations booléennes sur des objets polyédriques : la méthode des sections.

Chapitre 3. Opérations booléennes sur des objets à surfaces quelconques : extension de la méthode des sections.

Chapitre 4. Réalisation, mise en œuvre et complexité.

INTRODUCTION.

Un système de C.F.A.O. (Conception et Fabrication Assistées par Ordinateur) s'appuie principalement sur un modèle qui sert de description virtuelle d'un objet que l'on veut concevoir puis fabriquer.

Les évolutions de la modélisation ont permis d'effectuer de réels progrès pour prévoir le comportement des pièces que ce soit pour étudier une déformation possible, pour préparer de façon automatique leur fabrication ou même pour déterminer leur forme à partir de leurs fonctions.

Il apparaît clairement qu'il est essentiel pour mener à bien ces opérations, de pouvoir stocker la forme de l'objet. Historiquement, deux modèles solides ont été utilisés à cette fin : le modèle B-Rep (Boundary Representation) qui est une représentation de la frontière de l'objet et le modèle CSG (Constructive Solid Geometry) qui est un arbre de construction. De plus en plus, ces deux modèles sont maintenus conjointement dans le système, le modèle B-Rep étant généralement calculé automatiquement à partir de l'arbre de construction. Les possibilités offertes par ce dernier en matière de paramétrage et de remise en cause de l'historique de construction sont alors associées aux avantages amenés par le fait que la représentation par les limites est évaluée (visualisation rapide, cotations, ...). L'outil permettant de convertir une description CSG d'un objet en une suite de faces est communément appelé évaluation booléenne. Ce mémoire est consacré à cet opérateur.

Le travail présenté dans ce mémoire, s'inscrit dans le projet SACADO (Système Adaptatif de Conception et d'Aide au Développement par Ordinateur) développé au Laboratoire de Recherche en Informatique de Metz. Notre objectif à terme est de fournir au système SACADO un modèle hybride s'appuyant sur un arbre CSG étendu et évolué.

Dans ce cadre, nous avons étudié une nouvelle approche des opérations booléennes. Le chapitre 1 montre que de nombreux algorithmes sont connus. Cependant ils sont souvent sensibles aux imprécisions de calcul. En outre, leur temps d'exécution est important pour parfois arriver à un résultat erronné. De plus, ces algorithmes doivent gérer de nombreux cas particuliers et ont des difficultés à s'adapter à d'autres familles d'objets que les polyèdres, notamment aux objets non-Eulériens qui comportent des parties sans volume.

1. INTRODUCTION.

La diversité des besoins en C.A.O. a fait progressivement apparaître trois types d'objets :

- les objets à facettes planes ou polyèdres,
- les objets comportant des surfaces gauches,
- les objets possédant des parties sans volume appelés objets non-Eulériens.

Les algorithmes d'évaluation des opérations booléennes dont le but est, d'une part la construction d'objets et, d'autre part, de permettre la liaison entre le modèle CSG et le modèle B-Rep, se sont adaptés à ces trois types d'objets.

Dans ce chapitre, nous étudions les différentes méthodes que nous avons recensées pour ces trois types d'objets. Dans un premier temps, nous proposons de rappeler les conditions sur les opérateurs et les modèles par les frontières (section 2), puis les principales familles d'algorithmes existants sont détaillées (sections 3 à 5).

Dans la section 3, nous résumons les principaux algorithmes pour les objets polyédriques, la section suivante relatant ceux existants pour les objets à surfaces quelconques (section 4). Le chapitre se termine par une rapide présentation des approches pour des modèles non-Eulériens (section 5).

2. LES MODÈLES DE SOLIDES PAR LES FRONTIÈRES.

Formellement, les opérateurs booléens sur deux solides A et B peuvent être définis par :

- UNION : ensemble des points appartenant à l'un ou à l'autre solide (notée $A \cup B$),
- DIFFÉRENCE : ensemble des points appartenant au premier solide et n'appartenant pas au second (notée A-B),
- INTERSECTION : ensemble des points appartenant aux deux solides (notée $A \cap B$).

Dans la mesure où ces opérateurs ne sont pas des lois de composition interne dans l'ensemble des solides, les opérateurs régularisés (figure 1) doivent être mis en œuvre. Ils sont définis par [GAR 83], [RET 78] : $A \cup^{*} B = r\,(A \cup B)$ où r désigne l'application qui à un sous-ensemble de $\mathbb{R}^3$ associe l'adhérence de son intérieur.

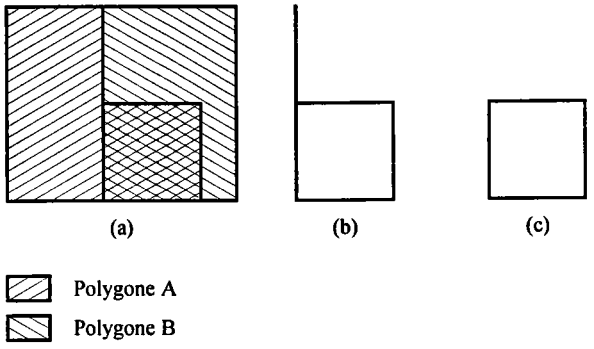


Figure 1 : (a) : Placement des polygones A et B.
(b) : intersection non régularisée
(c) : intersection régularisée.

La validité d'un modèle de solides est assurée par les conditions suivantes :

- rigidité : le solide décrit dans le modèle a une forme invariante, quelles que soient sa position et son orientation,
- finitude : le volume du solide décrit dans le modèle doit être fini,
- homogénéité : tout point à l'intérieur du solide ne peut appartenir à aucun autre objet.

Ces conditions très générales sont indépendantes de la famille de modèles utilisée. Elles doivent être converties en conditions vérifiables du point de vue informatique, qui dépendront du modèle utilisé. Dans un modèle Eulérien, la vérification de validité peut être faite soit par le traitement de chaque fonction appliquée sur un objet, soit par vérification sur les données géométriques et topologiques de l'objet après application de la fonction. Cette seconde solution reste difficile à mettre en œuvre (il est plus aisé de contrôler une modification d'une partie d'un objet que l'objet après modification) et seule la vérification de la validité topologique est relativement aisée.

Dans les algorithmes mettant en œuvre les opérations booléennes, la première solution est choisie. Toute opération construisant ou modifiant un solide se charge de la validité. Les opérations booléennes régularisées, qui permettent de composer des objets, sont supposées ainsi assurer la validité du résultat. Cette propriété est extrêmement importante, dans la mesure où certaines opérations deviendraient impossibles. Prenons par exemple, la fonction "appartenance d'un point au solide", qui est l'un des traitements de base utilisé dans de nombreux algorithmes ; cette fonction ne pourra être implantée en utilisant la méthode de la demi-droite et en comptant le nombre d'intersections significatives que si l'on est certain que l'objet est un solide. Il n'est cependant pas obligatoirement judicieux d'assurer la validité en permanence pendant le déroulement de la fonction. En particulier dans le cas des opérations booléennes, on peut accepter des états intermédiaires non valides, à condition que l'état final le soit.

Ainsi l'utilisation d'opérateurs d'Euler peut ne pas être la panacée, même si ces derniers sont à la base de certains algorithmes. On peut utiliser des primitives qui ne vérifient pas la validité topologique, laissant cette responsabilité à la fonction de construction qui les utilise.

Les modèles par les frontières s'appuient sur un graphe FAS (Faces, Arêtes, Sommets). Les conditions topologiques (figure 2) utilisées sur ce graphe sont les suivantes :

- a. une arête est partagée par deux faces exactement,
- b. une arête est parcourue une fois dans un sens, et une fois dans l'autre pour les deux faces la partageant,
- c. deux arêtes ne se coupent qu'en un sommet.

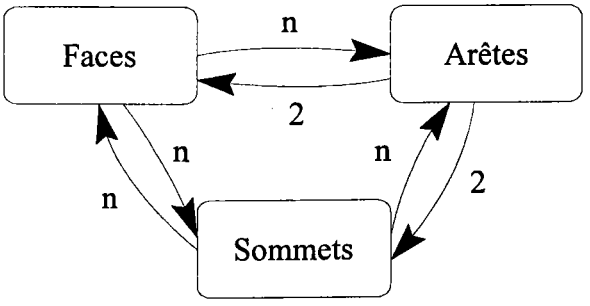


Figure 2 : Relations topologiques entre les entités géométriques.

Les conditions géométriques se réduisent à :

- a. deux faces ne peuvent se couper qu'en une arête partagée,
- b. la surface définie par le contour de la face ne peut avoir de recoupement (sans importance pour des modèles à facettes planes).

3. MISE EN ŒUVRE DES OPÉRATIONS BOOLÉENNES SUR DES OBJETS POLYÉDRIQUES.

Les opérations ensemblistes sont l'un des moyens très utilisés pour construire des solides. Leur mise en œuvre sur des modèles par des facettes planes est relativement difficile. Pourtant leur définition mathématique est très simple.

Les opérations booléennes sont utilisées dans deux types d'algorithmes [BRO 88] :

- ceux effectuant une opération booléenne sur deux objets B-Rep et dont le résultat est un objet B-Rep. Ils sont recensés sous le nom de "fusion des frontières" ("boundary merging"),
- ceux évaluant un arbre de construction CSG et réunis sous le terme "calcul des frontières" ("boundary evaluation").

L'idée sous-jacente de ces deux types d'algorithmes est cependant la même. Elle consiste à évaluer le résultat d'une opération booléenne sur deux objets B-Rep. L'évaluation d'un arbre de construction devient alors triviale : il suffit de transformer les primitives en objets B-Rep puis d'évaluer l'arbre de construction du modèle CSG. C'est pourquoi notre étude va essentiellement porter sur les algorithmes "fusion des frontières".

Pour étudier les algorithmes d'évaluation d'opérations booléennes sur les objets polyédriques, on peut classer les différentes méthodes (figure 3) en deux grandes familles :

- les méthodes utilisant un modèle intermédiaire spécifique (essentiellement les arbres octaux),
- les méthodes travaillant directement sur le modèle par les frontières.

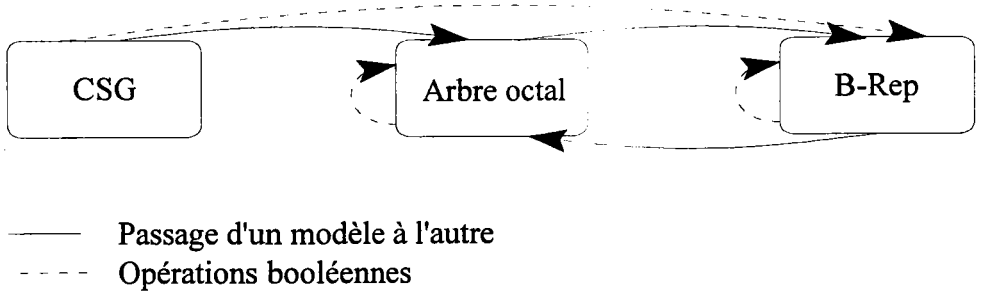


Figure 3 : Méthodes d'évaluations d'opérations booléennes.

Nous allons faire un survol exhaustif des différentes méthodes travaillant directement sur le modèle par les frontières puisque la méthode que nous présentons dans le chapitre 2 appartient à cette famille. Cependant nous expliciterons tout de même quelques méthodes appartenant à la première famille citée.

3.1. Méthodes d'évaluation directe sur des B-Rep.

3.1.1. Méthode d'intersection arêtes-faces.

Cette méthode [BRA 75], fait partie des premières méthodes publiées sur les algorithmes d'opérations booléennes. Elle ne concerne qu'une classe d'objets restreints. Ces derniers ne peuvent être composés que de faces planes et de surfaces cylindriques. Un objet est dit négatif s'il n'est considéré comme n'étant constitué que d'antimatière sinon il est dit positif. Les objets négatifs servent notamment à supprimer de la matière dans un objet positif. Les points de l'espace euclidien 3D sont classés suivant leur appartenance aux objets.

Soit un objet Q positif, alors pour tout point p de l'espace :

- si Q(p) = 1 alors le point est à l'intérieur de l'objet Q,
- si Q(p) = 1/2 alors le point est sur la surface de l'objet Q,
- si Q(p) = 0 alors le point est à l'extérieur de l'objet Q.

Soit un objet Q négatif, alors pour tout point p de l'espace :

- si Q(p) = -1 alors le point est à l'intérieur de l'objet Q,
- si Q(p) = -1/2 alors le point est sur la surface de l'objet Q,
- si Q(p) = 0 alors le point est à l'extérieur de l'objet Q.

Cette caractérisation permet la définition d'une opération d'addition de deux objets. L'addition de deux objets va donner un objet positif et/ou un objet négatif. Cette opération est définie par les tables ci-dessous (figure 4). La première table représente l'objet résultat positif et la seconde l'objet résultat négatif.

		Q1(P)				
	+	1	1/2	0	-1/2	-1
	1	1	1	1	1/2	0
Q2(P)	1/2	1	?	1/2	?	0
	0	1	1/2	0	0	0
	-1/2	1/2	?	0	0	0
	-1	0	0	0	0	0

		Q1(P)				
	+	1	1/2	0	-1/2	-1
	1	0	0	0	0	0
Q2(P)	1/2	0	0	0	?	-1/2
	0	0	0	0	-1/2	-1
	-1/2	0	?	-1/2	?	-1
	-1	0	-1/2	-1	-1	-1

Figure 4 : Tables d'opérations d'addition entre deux objets. La première table donne l'objet résultat positif et la seconde l'objet résultat négatif. L'ensemble des "?" inclus dans les tables est défini suivant les normales sortantes aux deux objets au point testé.

Cette définition mathématique d'un objet assure une classification rapide d'un point de l'espace mais ne permet pas une représentation informatique de l'objet (un objet ne peut être représenté informatiquement par l'ensemble des points qui le compose). C'est pourquoi la méthode s'appuie sur cette définition mais ne l'utilise pas directement.

La méthode diffère suivant la position des deux objets et donne deux cas :

- les deux objets sont juxtaposés, l'opération d'addition est locale aux deux faces de la juxtaposition,
- les deux objets ont une intersection.

Le second cas peut être décrit simplement par :

- calcul de toutes les intersections arêtes / faces, caractérisation et éclatement éventuels,
- construction de l'objet final en fonction de tous les cas particuliers qui peuvent se poser.

Cependant la mise en œuvre de l'algorithme décrit effectivement tous les cas particuliers se présentant ; ce qui rend la compréhension lourde et ardue et son implémentation difficile.

Bien que cela n'ait pas été le cas dans la première implémentation de ce type d'algorithme, la détermination des faces peut se faire par recherche de cycles dans un graphe.

3.1.2. Méthode du polygone de section.

La méthode étudiée dans cette section a été élaborée dans [MAN 82], [MAT 83] puis améliorée dans [MMP 87a] [MMP 87b]. Elle est basée sur le polygone de section.

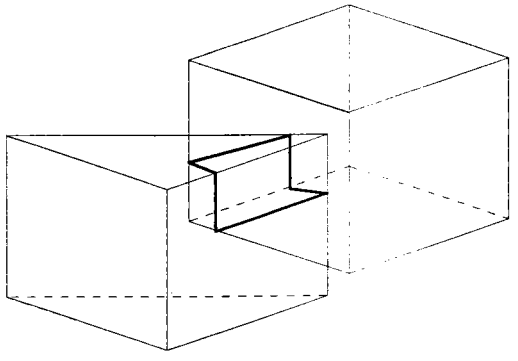


Figure 5 : Ligne de section

Le principe est le suivant : l'intersection de deux polyèdres se fait le long d'une ou plusieurs lignes brisées (figure 5) qui permettent de partager chaque polyèdre en deux catégories (figure 6) :

- la partie du polyèdre qui est extérieure à l'autre solide,
- la partie du polyèdre qui est intérieure à l'autre solide.

On obtient donc quatre sous-ensembles qu'il suffit d'assembler selon les règles suivantes :

Union	= Coller (A extérieur à B , B extérieur à A)
Intersection	= Coller (A intérieur à B , B intérieur à A)
Différence	= Coller (A extérieur à B , B intérieur à A)

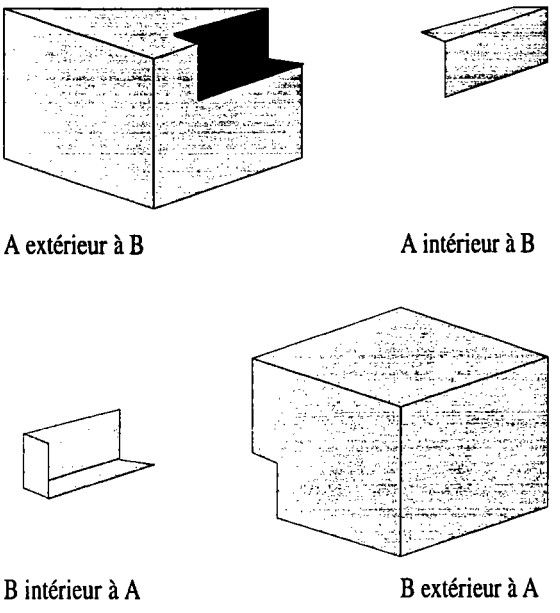


Figure 6 : Les différentes parties des objets A et B suivant leur classification par rapport à l'autre objet.

Globalement l'algorithme est le suivant :

Calculer les intersections de A avec B et réciproquement (intersection arêtes faces)
Construire la ligne de section joignant les points d'intersection
Partager A en (A intérieur à B) et (A extérieur à B)
et
Partager B en (B intérieur à A) et (B extérieur à A)
Choisir selon l'opérateur :

Union	:	Coller (A extérieur à B , B extérieur à A)
Intersection	:	Coller (A intérieur à B , B intérieur à A)
Différence	:	Coller (A extérieur à B , B intérieur à A)

Les calculs d'intersection arêtes / faces impliquent le traitement de tous les cas particuliers pouvant se présenter (ce qui était également le cas pour la méthode d'intersection arêtes-faces [BRA 75]). La difficulté essentielle de cette approche réside dans la détermination de la ligne de section. La ligne de section est formée par les points d'intersection entre arêtes et faces qui doivent être correctement reliés.

L'idée générale est que l'on peut relier deux points d'intersection si ils appartiennent à une même face sur A et sur B pour des faces convexes [MAN 82]. Dans le cas de faces non convexes [MMP 87a] [MMP 87b], on ordonne les points d'intersections communs à deux faces ce qui donne un certain nombre de segments candidats. Un segment candidat constitue une arête si le milieu du segment appartient à chaque face.

On insère les éléments de la ligne de section dans chaque objet en deux étapes :

- insertion des points d'intersection dans les contours de faces auxquelles ils appartiennent,
- pour une telle face, on la parcourt jusqu'à un point d'intersection, puis on parcourt la ligne de section tant que l'on est sur la même face. Le parcours se pousuit sur le contour de la face jusqu'à un nouveau point d'intersection ou jusqu'à ce que le parcours donne un contour fermé.

La partition en intérieur et en extérieur est assez simple puisqu'une face d'un ensemble (extérieur et intérieur) ne peut pas traverser l'autre solide. Par adjacence d'arêtes initiales et test de classification d'un point, on détermine facilement l'état des sous-ensembles de faces.

Les opérateurs d'Euler sont en général utilisés [HIG 93], car ils permettent de traiter facilement des opérations consistant à couper des arêtes, créer des arêtes nulles...

Cette famille d'algorithmes fonctionne sur des faces planes et impose un traitement particulier pour les faces coplanaires.

3.1.3. Méthode de découpage de faces.

La troisième méthode [PIL 89] [LTH 86] aborde les opérations booléennes par un découpage de faces.

Chaque face de A est étudiée par rapport aux faces de B. Lorsqu'une face de A coupe une face de B, elle est partagée en plusieurs "sous-faces" à partir des points d'intersections avec les arêtes de la face B (figure 7).

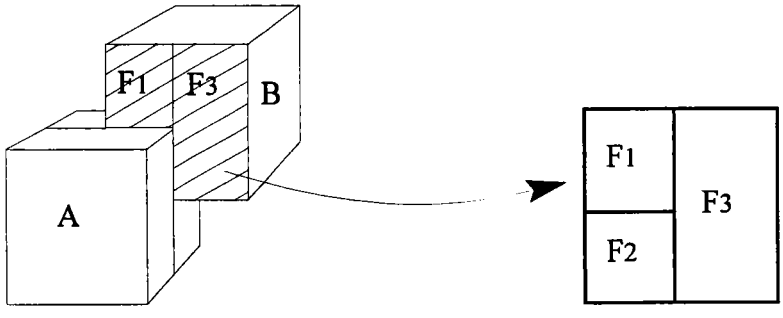


Figure 7 : La face du solide B est découpée en sous-faces F1, F2 et F3. Les faces F1 et F3 sont extérieures au solide A, par contre la face F2 est intérieure au solide A. F1 et F3 vont donc appartenir par exemple à l'union de A et de B. Il faut les réunir pour obtenir une des faces de l'objet résultat.

Ensuite, les caractéristiques de chaque sous-face sont déterminées.

Une sous-face peut être :

- à l'intérieur du solide résultat,
- sur le solide résultat,
- à l'extérieur du solide résultat.

Cette détermination est faite de la manière suivante : chaque sommet d'une sous-face est évalué par rapport au solide ; s'il est intérieur ou extérieur au solide, la sous-face ne peut appartenir à la frontière du solide final et elle est abandonnée. Si tous les sommets d'une sous-face sont sur le solide, il suffit de tester un point intérieur à la sous-face pour s'assurer que la face est sur le solide ou pas (dans la mesure où aucune sous-face ne peut couper le solide et ne peut donc le toucher que sur la frontière). Enfin, pour éviter les faces ne séparant pas l'intérieur de l'extérieur (n'appartenant pas à la fermeture de l'intérieur), deux points de part et d'autre de la sous-face à une petite distance sont testés : la sous-face n'est considérée comme appartenant au solide que si l'un des deux points est intérieur et l'autre extérieur.

La seule partie algorithmique restant à définir est la détermination du fait qu'un point donné soit intérieur, extérieur ou sur la frontière du solide. Dans [PIL 89], les auteurs évaluant un arbre CSG proposent naturellement de tester cette propriété sur les feuilles (primitives), ce qui est relativement trivial, puis de donner la propriété sur l'arbre en évaluant ce dernier récursivement. Dans le cas où les deux objets sont connus par leurs frontières, il suffit de mettre en œuvre un algorithme de type demi-droite [LTH 86] pour déterminer cette propriété (sur A, sur B et sur l'opération booléenne de A et B).

Les sous-faces faisant partie de la frontière de l'objet final doivent être ensuite regroupées afin d'obtenir l'ensemble des faces du solide résultat. Les sous-faces appartenant à un même plan sont fusionnées. En supprimant les arêtes de contact, la face finale est formée.

Parmi les difficultés liées à cette famille d'algorithmes, on peut citer :

- le nombre important de sous-faces créées au fur et à mesure du déroulement de l'algorithme, ce qui peut influer sur les performances,
- l'existence de sous-faces très petites qui peuvent impliquer des problèmes de précision.

3.1.4. Méthode du pan.

La quatrième méthode [MIC 87] [BEN 93] utilise une structure de représentation différente du graphe FAS du B-Rep afin de faciliter les calculs d'intersection. Nous allons expliquer quelle est la structure utilisée avant de décrire la méthode.

L'élément de base de la représentation que proposent les auteurs est le *pan*. Un pan est une portion de face incidente à une arête donnée d'un coté bien déterminé de l'arête. A un pan, sont associées trois informations (voir figure 8) :

- la face *f* sur laquelle se situe le pan,
- l'arête *a* sur laquelle se rapporte le pan : elle est orientée de telle manière que son origine précède son extrémité dans l'ordre (XYZ),
- une valeur *s* égale à +1 ou -1.

Le triplet (*a,f,s*) définit un pan de f incident à *a* et de sens *s*. Si *N* est le vecteur normal à la face *f* et *E* le vecteur allant de l'origine à l'extrémité de *a* alors le vecteur *V* (nommé vecteur indicateur) définit par *s* * (*E* ^ *N*) se dirige vers l'intérieur de la face.

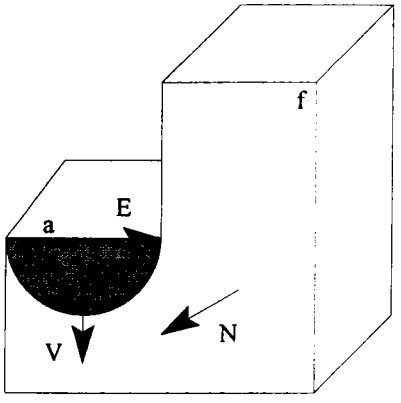


Figure 8 : Notion de pan.

Un solide est un ensemble de faces maximales. Une face maximale est le plus grand sous-ensemble de la frontière du solide porté par un même plan orienté. Une face est composée d'un ensemble de pans. La frontière d'un solide est donc simplement composée d'un ensemble de pans.

La méthode sur ce type de représentation d'objets est la suivante :

- comme pour la méthode précédente, chaque face de A est étudiée par rapport aux faces de B. L'ensemble des couples susceptibles d'avoir une intersection est réduit par l'utilisation de boîtes englobantes.
- pour un couple candidat de faces, l'ensemble des arêtes d'intersection n'est pas calculé directement. Comme ces arêtes appartiennent à la droite d'intersection D entre les deux plans des faces, les segments d'intersection entre chaque face et la droite D sont générés séparément. Ces segments sont considérés comme identiques suivant leur classification (Sur, Dans, Hors) par rapport aux faces dont ils sont issus pour former les arêtes d'intersection (figure 9).

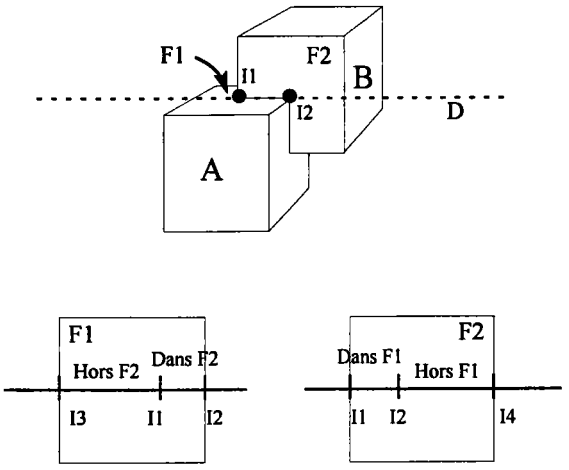


Figure 9 : Le segment $[I_1,I_2]$ est classé "DansF"$_2$ "Sur F_1" et "DansF_1" sur F2. Il fait donc partie des arêtes d'intersection entre l'objet A et l'objet B.

Puisque l'on calcule les arêtes d'intersection (elles ne peuvent pas être formées à partir d'un segment "Hors"), ne sont retenus que les couples de segments (Sur,Sur), (Sur,Dans) et (Dans,Dans). A chaque nouvelle arête sont associés exactement quatre pans : deux pans de l'objet A de sens opposé et réciproquement pour B. Le découpage d'une arête originelle d'un des deux solides opérandes donne des arêtes homogènes. Chaque sous-arête homogène est créée avec autant de pans de A que de pans originaux incidents à l'arête originelle. Si de plus elle est incluse dans une face de l'autre solide, alors elle reçoit deux autres pans supplémentaires héritant du plan de la face de l'autre solide.

L'ensemble des pans de l'objet final est donc inclus dans l'ensemble des pans des arêtes d'intersection, des arêtes homogènes et des arêtes banales (i.e. dont tous les pans appartiennent au même solide). Il reste à déterminer l'ensemble des pans utiles au solide résultat.

Si un pan provient d'une arête banale qui est intérieure à l'autre solide, alors le pan est utile. La détermination de la classification de l'arête par rapport au solide se fait par un test de classification du milieu de l'arête par rapport au même solide (lancer de demi-droite).

Si un pan est incident à une arête non banale (arête d'intersection ou arête homogène), il est utile s'il détermine l'intérieur du solide résultat. Cette détermination s'effectue par les informations données par le vecteur indicateur du pan et la normale sortante à la face du pan.

La description finale de la frontière du solide peut alors être générée à partir de tous les pans trouvés utiles. Les faces maximales sont reconstituées avec les pans de plans identiques.

3.1.5. Méthode des domaines.

Cette méthode [ZHA 92] utilise plus spécifiquement les informations topologiques du B-Rep pour mettre en œuvre les opérations booléennes.

Un solide est défini par un ensemble de faces appelé composante connexe ("shell"). Une composante connexe d'un solide S peut être représentée par un graphe G dont les sommets du solide correspondent aux sommets du graphe (figure 10). Les arêtes du solide S correspondent aux arcs du graphe G et les faces du solide sont les régions du graphe G.

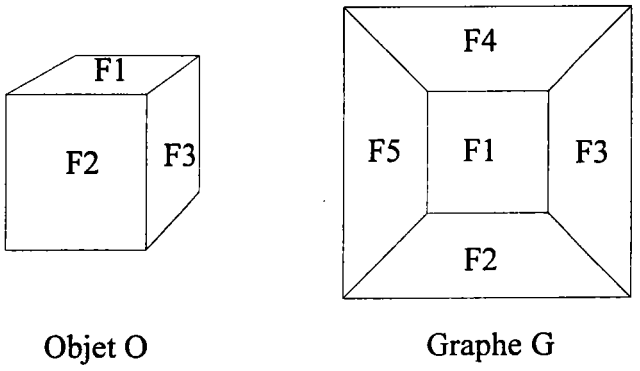


Figure 10 : Un objet et son graphe.

Etant donné deux objets O_1 et O_2 et leurs graphes associés G_1 et G_2, si les deux objets O_1 et O_2 ont une intersection alors G_2 découpe G_1 et réciproquement. Les intersections entre G_1 et G_2 peuvent se faire à plusieurs niveaux :

- les objets O_1 et O_2 possèdent deux faces qui ont une intersection, les graphes G_1 et G_2 vont hériter d'un nouvel arc qui représentera l'arête d'intersection,
- les objets O_1 et O_2 possèdent deux faces coplanaires qui ont des parties communes, les graphes G_1 et G_2 vont hériter de deux nouvelles régions correspondant à la partie d'intersection entre les faces coplanaires.

L'ensemble de ces intersections va former des domaines dans G_1 et G_2. Un domaine est un ensemble de régions connectées du graphe qui a la même classification par rapport à l'autre graphe. Chaque domaine est formé à partir des intersections des données géométriques des solides. Un domaine peut donc être issu :

- d'un cycle d'arêtes d'intersection. L'intersection des deux objets n'est constitué que d'une suite d'arêtes,
- d'un cycle d'intersection mixte. Les deux objets ont deux faces coplanaires : l'intersection des deux objets est donc formée à partir des arêtes d'intersection et de la partie de face commune aux deux faces coplanaires,
- d'un cycle d'intersection de faces. Les objets n'ont que des faces communes en intersection,
- d'aucun cycle. Les objets n'ont qu'une seule face commune et il n'existe pas de matière commune aux deux objets.

Chaque domaine doit ensuite être classé par rapport à l'autre solide. Un domaine étant formé d'un ensemble de régions, il suffit de connaître la position d'une région pour connaître la position du domaine par rapport à l'autre solide. Une région d'un graphe représente une face ou une sous-face du solide. La position d'une face est déterminée par la position d'un point de la face.

La classification d'un domaine implique donc la classification d'une face ou d'une sous-face du solide et non de l'ensemble des faces ou sous-faces faisant partie du domaine.

En réunissant les domaines nécessaires à une opération booléenne donnée, on reconstruit le graphe de l'objet résultat.

La méthode se résume donc par :

- construction des cycles d'intersection et découpage des composantes connexes en domaine,
- classification des domaines,
- regroupement des domaines.

Dans la formalisation de son approche, cette méthode peut sembler différente de la méthode de polygone de section [MAN 82]. Cependant leurs finalités sont identiques puisqu'elles consistent à classer les différentes parties d'un solide par rapport à l'autre solide afin de les unifier de façon adéquate pour former l'objet résultat.

3.1.6. Méthodes par faces.

D'autres méthodes préconisent de traiter les opérations booléennes face par face et de ne conserver que les portions de faces intervenant dans la composition de l'objet résultat. Notamment, dans [MAT 88], il est indiqué qu'il est possible de reprendre l'idée générale de la méthode des opérations booléennes en deux dimensions pour implémenter celle des opérations booléennes en trois dimensions en considérant la position relative des vecteurs normaux aux faces. L'article ne donne cependant aucune description de l'algorithme. De même, [HOF 89] propose une approche qui transforme le problème des opérations booléennes tridimensionnelles en une gestion de graphes.

La méthode publiée dans [KIY 92] donne un bon aperçu de cette famille d'algorithmes. Les auteurs précisent qu'il n'est pas nécessaire d'exécuter les opérations booléennes d'union et de différence puisqu'elles peuvent être facilement déduites de l'intersection par les lois de De Morgan :

$A \cup B = \overline{\overline{A} \cap \overline{B}}$

$A - B = A \cap \overline{B}$ avec $\overline{E}$ définissant le complémentaire de l'objet E.

Cependant cette interprétation des opérations booléennes demande de traiter des objets négatifs et infinis.

L'algorithme présenté est le suivant :

1. Exécuter les actions suivantes pour un solide A donné :
   1. Etendre une face F_A du solide A à son plan infini S_A, obtenir une section C_A entre le plan S_A et les faces d'un autre solide B.
   2. Trouver les points d'intersection entre la face F_A et la section C_A et découper les arêtes en fonction de leur classification (Dans, Hors, Sur).
   3. En suivant les arêtes découpées en *2*, obtenir la face d'intersection I_A entre F_A et la section C_A.
   4. Exécuter les étapes *1* à *3* pour toutes les faces du solide A et obtenir toutes les faces d'intersection. Unifier ces faces pour générer une composante connexe ouverte Sh_A.
2. De la même manière, exécuter 1 pour un autre solide B et générer une deuxième composante connexe Sh_B.
3. Unifier les composantes connexes ouvertes Sh_A et Sh_B pour obtenir une composante connexe fermée Sh_C qui est le solide intersection résultat.

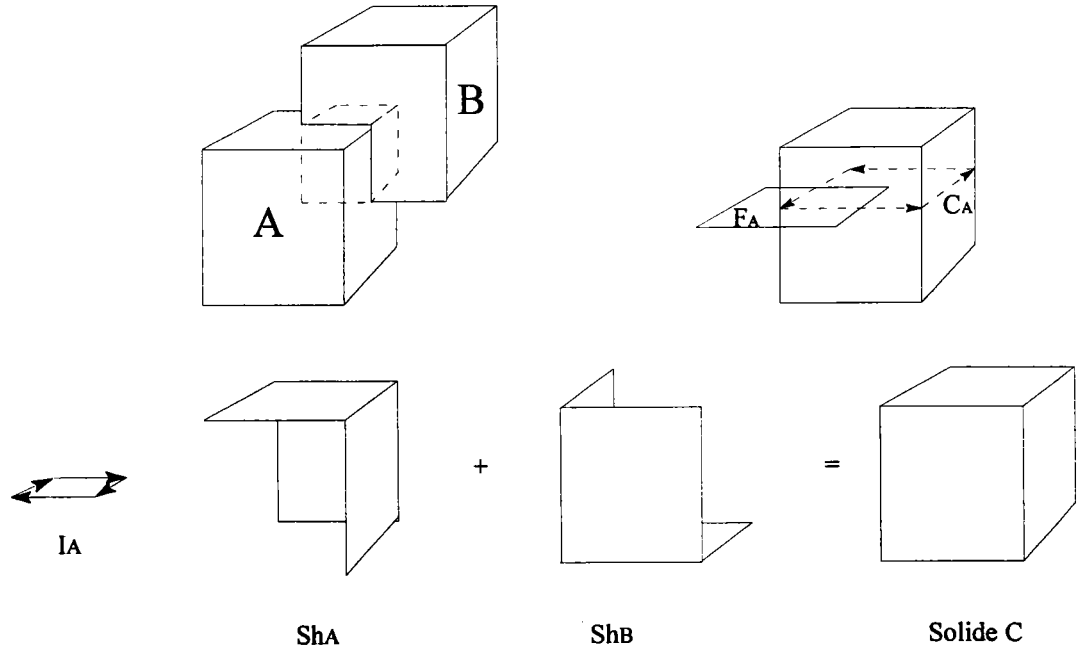


Figure 11 : Les différentes étapes de l'algorithme de la méthode par faces.

La recherche de la section du solide B suivant le plan S_A de la face F_A s'effectue en recherchant tous les points d'intersection entre les arêtes de B et le plan S_A. Les points d'intersection ainsi trouvés doivent ensuite être reliés correctement afin de former la section. Une façon adéquate de relier les points est de suivre le contour du solide B. Ceci est possible grâce aux entités reliées aux arêtes de B ayant donné une intersection avec le plan S_A et les faces adjacentes à ces arêtes.

A partir de la section les auteurs ont détaillé de nombreux cas particuliers pour générer la face d'intersection entre la face F_A et la section C_A :

(1) le plan S_A n'a pas d'intersection avec le solide B, la section C_A n'existe pas,

(2) le plan S_A a une intersection avec le solide B et la section C_A existe,

(3) la face F_A est coplanaire avec une face de B.

A travers ces trois cas, les auteurs distinguent encore des cas particuliers. Par exemple, ils expliquent comment résoudre le cas (2) lorsque il existe plusieurs contours extérieurs pour la section ou lorsque la section est formée de plusieurs contours intérieurs. Le cas (3) est aussi découpé en huit sous-cas par rapport à la position des deux objets. Enfin ils terminent en expliquant comment résoudre le problème du cas (2) et du cas (3) simultanément (la face F_A est coplanaire avec une face de B et il existe une section C_A).

Face à cette abondance de détails qu'il aurait peut-être été possible de généraliser, les auteurs ne présentent guère d'explications sur la manière dont ils unifient les parties des faces qu'ils trouvent faisant partie de l'intersection. Ce point nous paraît pourtant bien plus important puisqu'il permet de générer des objets cohérents par rapport au modèle utilisé. Nous ne savons donc pas si les auteurs recherchent les points de l'objet final de manière géométrique dans les différentes parties de face trouvées ou si cette recherche s'exécute par rapport à la topologie des deux objets. L'article ne donne également aucune indication pour expliquer si les auteurs ont rencontré des difficultés lors de cette unification et quelles sont les solutions qu'ils auraient pu y apporter.

3.1.7. Conclusion.

Nous venons de recenser les différentes méthodes d'opérations booléennes sur deux polyèdres A et B. Les méthodes sont les suivantes :

(1) intersection entre les faces et les arêtes afin de générer les arêtes et les faces de l'objet résultat,

(2) création d'un polygone de section permettant de séparer les deux objets en parties distinctes suivant leur classification par rapport à l'autre objet,

(3) découpage de faces et classification de ces faces par rapport à l'autre objet,

(4) modélisation différente des polyèdres par la notion de pan attaché à une arête,

(5) passage de l'objet à la notion de graphe et classification des domaines du graphe,

(6) méthode par construction des faces de l'objet résultat à partir des faces des deux objets opérandes.

Ces méthodes procèdent en quatre étapes :

1- Calcul des intersections entre des entités de la frontière de A et des entités de la frontière de B. Éclatement de ces entités.

2- Classification de chaque élément provenant du pas 1 par rapport à l'autre objet.

3- Sélection suivant l'opération booléenne des éléments à retenir pour la composition de l'objet résultat.

4- Fusionnement des différentes entités trouvées au pas 3 pour obtenir l'objet résultat.

L'approche de la méthode (5) est similaire à celle de la méthode (4) mais elle permet une amélioration notable en introduisant la notion de domaine d'un graphe. Ceci permet de ne tester qu'une sous-face d'un domaine au lieu de tester toutes les sous-faces créées.

Les différences majeures proviennent du choix des entités sur lesquelles s'effectue une intersection et de la façon de classifier les entités par rapport à l'objet résultat. Pour les méthodes (1) (2) et (5), des intersections arête-face sont effectuées, par contre pour les méthodes (3), (4) et (5) ce sont des intersections face-face. La classification en intérieur-extérieur se réalise avec un polygone de section pour la méthode (2), avec une recherche de section pour la méthode (6) et plus classiquement par test d'un point avec un lancer de demi-droite pour les méthodes (1), (3), (4) et (5).

Toutes ces méthodes sont difficiles à implémenter, il faut tenir compte de tous les cas particuliers pouvant se présenter ; ce qui ne donne généralement pas des algorithmes faciles à écrire et à construire. Elles sont plus ou moins gourmandes en temps suivant la méthode employée pour le calcul d'intersection. De plus, puisqu'elles calculent des intersections, elles sont sensibles aux erreurs de calcul dues à l'imprécision du codage des réels sur les ordinateurs utilisés pour leur implémentation. Certains algorithmes tendent à résoudre ce dernier problème et les solutions proposées sont diverses :

- utilisation d'une arithmétique rationnelle paresseuse [BEM 93] [JAI 93] permettant de coder les réels par un rationnel proche,
- en réévaluant les différents tests sur un sommet, une arête ou une face [HOF 89].

3.2. Méthodes utilisant un modèle intermédiaire.

La section précédente présente les différentes méthodes recensées qui travaillent directement sur les objets opérandes. Nous abordons, dans cette section, les méthodes qui utilisent un modèle intermédiaire. Ces méthodes prennent souvent un modèle arbre octal pour évaluer les opérations booléennes. Un arbre octal (ou octree) est une décomposition récursive de l'espace considéré comme un grand cube dont les arêtes sont parallèles aux axes du repère. Un objet de cet univers est représenté par un ensemble de cubes plus petits. Chaque cube est le résultat d'une subdivision récursive d'un cube donné en huit sous-cubes identiques. A chaque pas de la division un cube peut être noir s'il est totalement intérieur à l'objet ou blanc s'il est complètement extérieur à l'objet. Si une partie de la frontière appartient au cube alors le cube est gris. L'ensemble des cubes gris est soumis à une division. La division s'arrête lorsque les cubes gris sont devenus suffisamment petits. Les feuilles de l'arbre octal sont les cubes noirs, blancs ou gris ayant atteint le niveau maximum de division.

Si la méthode appartient à la famille "fusion des frontières" alors elle se décompose de la manière suivante :

- passage du modèle B-Rep au modèle arbre octal,
- évaluation de l'opération booléenne sur les deux arbres octaux,
- passage du modèle arbre octal au modèle B-Rep.

Par contre, si la méthode appartient à la famille "calcul des frontières" alors elle ne comporte plus que deux parties qui sont :

- passage du modèle CSG au modèle arbre octal, l'évaluation des opérations booléennes étant nécessaire pour la construction du modèle arbre octal,
- passage du modèle arbre octal au modèle B-Rep.

Cependant l'arbre octal présente le problème suivant : lors du passage du modèle B-Rep au modèle arbre octal (ou du modèle CSG au modèle arbre octal), l'ensemble des informations géométriques du modèle sont perdues. C'est pourquoi les auteurs proposent d'utiliser un arbre octal étendu [NAB 86] (extended octree ou enriched octree) qui permet le stockage de certaines données géométriques.

L'arbre octal étendu diffère de l'arbre octal dans le sens où chaque feuille de l'arbre peut stocker une information géométrique (figure 12). Une feuille peut donc être :

- homogène : elle correspond aux cubes noirs ou blancs de l'arbre octal,
- une feuille "face" : la feuille contient une portion de face de l'objet mais ne peut contenir ni une portion d'arête, ni un sommet de cette face,
- une feuille "arête" : la feuille contient une portion d'arête mais ne contient aucune portion de faces non adjacentes à cette arête,
- une feuille "sommet" : la feuille ne contient que des portions d'arêtes partant de ce sommet.
- une feuille "plans" : la feuille contient plusieurs plans.

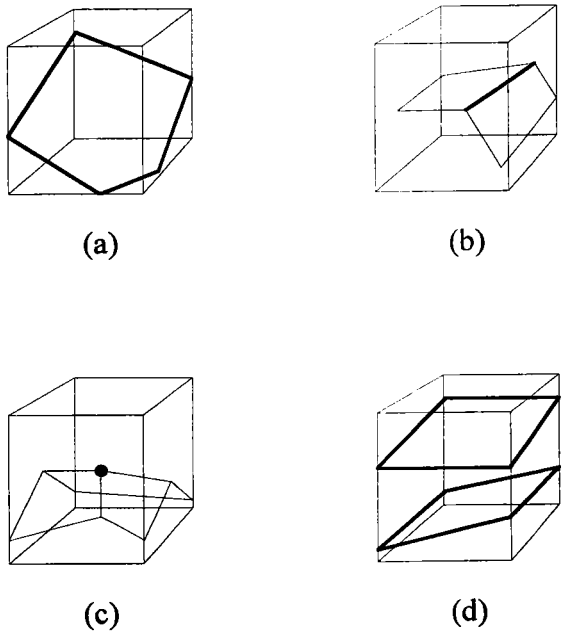


Figure 12 : Les différentes feuilles de l'arbre octal étendu.
(a) : feuille "face", (b) : feuille "arête", (c) : feuille "sommet",
(d) : feuille "plans".

Le passage du modèle B-Rep au modèle arbre octal [NFB 87] se fait par divisions successives de l'espace. Chaque cube ainsi trouvé est classé par rapport à la frontière de l'objet. Si un cube ne peut être classé, il est subdivisé en octants et le test est poursuivi sur les huit octants jusqu'à obtention d'une classification possible.

L'évaluation d'une opération booléenne sur deux arbres octaux s'effectue de la manière suivante [WAL 89], [PLA 93] :

Étant donné deux arbres octaux étendus A_1 et A_2, l'algorithme impose que la racine de l'arbre A_1 représente la même portion d'espace que celle de l'arbre A_2. La première feuille de A_1 est étudiée avec la première feuille de A_2. Si les deux feuilles occupent le même espace alors la première feuille de l'arbre octal résultat est générée à partir de la configuration géométrique des deux feuilles étudiées. Sinon la feuille la plus grande est subdivisée jusqu'à obtention de la taille de la plus petite feuille. La seconde feuille de A_1 est ensuite étudiée avec la seconde feuille A_2 et ce, jusqu'à parcours complet des feuilles de A_1 et de A_2. Etant donné qu'il existe cinq types de feuilles différentes et trois opérations booléennes, il y a quinze combinaisons possibles pour un couple de feuilles. Les combinaisons les plus difficiles sont celles où une des feuilles contient un sommet.

Pour le passage du modèle CSG au modèle arbre octal [NFB 87], l'arbre CSG est d'abord transformé en un arbre CSG étendu (expanded CSG). Un arbre CSG étendu est un arbre CSG dont les feuilles peuvent contenir un plan définissant alors un demi-espace. Si l'arbre CSG est un arbre CSG dont les feuilles sont des primitives classiques, il est aisé de trouver une boîte englobante de l'arbre CSG. Ensuite, de la même façon que pour un modèle B-Rep, le modèle CSG est transformé en un arbre octal.

Il reste ensuite à convertir l'arbre octal en un modèle B-Rep [AYA 88]. Les équations des faces de l'objet peuvent être obtenues facilement à partir de l'arbre octal. Les informations géométriques restantes sont obtenues en deux traitements :

- le premier parcourt l'arbre octal et recense les différents sommets pour former une liste associée à chaque face. Cette liste est composée de couples de faces adjacentes provenant des cônes de faces adjacentes des sommets.
- le second trie l'ensemble de ces listes pour obtenir l'objet résultat.

L'inconvénient majeur d'un arbre octal est qu'il a besoin d'un grand espace mémoire. De plus l'ensemble de ces méthodes nécessitent deux traitements supplémentaires en plus du calcul des opérations booléennes : le passage du modèle B-Rep vers le modèle arbre octal et inversement.

4. MÉTHODES SUR DES OBJETS CONTENANT DES SURFACES GAUCHES.

L'ensemble des méthodes qui exécutent les opérations booléennes sur des objets polyédriques a été présenté dans la section précédente. Dans cette section, nous nous intéressons aux méthodes d'opérations booléennes sur des objets à surfaces quelconques. Ces méthodes reprennent le schéma vu à la section 2.1 de ce chapitre :

(1) Calcul des intersections entre des entités de la frontière de A et des entités de la frontière de B. Éclatement de ces entités en vue de leur classification.

(2) Classification de chaque élément provenant du cas (1) par rapport à l'autre objet.

(3) Sélection suivant l'opération booléenne des éléments à retenir pour la composition de l'objet résultat.

(4) Fusionnement des différentes entités trouvées au cas (3) pour obtenir l'objet final .

L'ajout de surfaces non planes intervient surtout dans le point (1). Les entités dont on recherche l'intersection ne sont plus aussi simples que des arêtes rectilignes ou des faces planes. Il faut y ajouter un calcul d'intersection entre surfaces.

Dans un premier temps, nous allons décrire brièvement les différentes méthodes de calcul d'intersection de surfaces, puis nous décrirons les méthodes recensées pour les opérations booléennes.

4.1. Intersection de surfaces.

La fiabilité des algorithmes d'intersection de surfaces est difficile à assurer en vue de leur utilisation dans un système de C.A.O. Elle est liée à leur comportement lors de recouvrement de surfaces ou lors de la présence de très petites courbes fermées d'intersection. De plus l'ensemble de ces algorithmes est très sensible aux erreurs numériques dues à une précision finie des opérandes lors des calculs.

Divers algorithmes d'intersection de surfaces existent qui peuvent être classés en quatre catégories [PAT 93] :

- les méthodes analytiques [FAR 87b],
- les méthodes d'intersection des isoparamétriques [MIC 92],
- les méthodes de suivi [LUK 89] [BAH 88] [BER 95],
- les méthodes de subdivision [HOE 85] [AZB 90].

Dans les sections suivantes, nous allons décrire de façon succincte ces quatre familles d'algorithmes d'intersection de surfaces.

4.1.1. Méthodes analytiques.

Les méthodes analytiques dérivent une équation principale afin d'obtenir la courbe d'intersection entre deux surfaces. Pour les surfaces polynomiales, l'équation de la courbe d'intersection est de la forme $f(u,v) = 0$ où f est un polynôme en u et v. Cette équation peut être obtenue en substituant les coordonnées cartésiennes d'une surface polynomiale R(u,v) dans l'équation algébrique d'une surface telle que $f(R) = 0$.

En théorie, il est possible de traiter l'intersection de deux surfaces représentables par des polynômes en cherchant une représentation algébrique pour une des deux surfaces. En général, la substitution des coordonnées cartésiennes dans l'équation algébrique donne une équation algébrique de degré élevé. Plus le degré est élevé, plus la détection topologique d'une telle courbe devient difficile.

4.1.2. Méthodes incluant une intersection des isoparamétriques.

L'ensemble de ces méthodes réduit la complexité du problème d'intersection de surfaces en partant de l'intersection des isoparamétriques d'une surface avec l'autre surface.

Les méthodes se déroulent en trois parties :

- calcul des intersections des isoparamétriques d'une surface avec l'autre surface,
- obtention d'un ensemble de points appartenant à la courbe d'intersection entre les surfaces,
- formation de la courbe d'intersection en reliant les points d'intersection.

Si le choix des isoparamétriques n'est pas assez fin, la méthode peut omettre des petites boucles d'intersection ou des points isolés qui indiquent une tangence des deux surfaces.

4.1.3. Méthodes de suivi.

Ces méthodes génèrent un ensemble de points de la courbe d'intersection en avançant pas à pas à partir d'un point donné sur la courbe et dans une direction donnée par la différenciation locale de la courbe. L'ensemble de ces méthodes est délicat à mettre en œuvre dans le sens où elles demandent un point de départ pour chaque portion de la courbe d'intersection. De plus, le pas d'avancement doit être judicieusement choisi sinon il risque d'induire des boucles infinies.

4.1.4. Méthodes de subdivision.

Ces méthodes impliquent la décomposition du problème d'intersection de surfaces en problèmes de degré de difficulté moindre (méthode de Newton). La résolution du problème global est donc fragmentée en solutions de problèmes simples. Les solutions "simples" doivent être reliées pour former la solution du problème global.

Les techniques de subdivision ne demandent pas de point de départ contrairement à la méthode "pas à pas". Cependant, il est difficile de garantir une connexion correcte des branches de la courbe d'intersection près de points singuliers.

4.1.5. Problèmes communs.

L'ensemble de ces méthodes présente les problèmes suivants :

- difficulté de détecter les singularités des courbes d'intersection comme le recouvrement de surfaces, des petites courbes d'intersection ou des points d'intersection,
- sensibilité aux imprécisions de calculs,
- temps de calcul parfois trop importants,
- impossibilité, en général, d'obtenir la courbe dans le même modèle mathématique que les surfaces.

4.2. Évaluation d'opérations booléennes.

L'inclusion des surfaces gauches dans la composition des objets nécessite l'insertion de ces données géométriques dans un B-Rep nommé B-Rep exact. Un B-Rep exact est toujours un graphe Faces-Arêtes-Sommets dont les supports des faces ne sont pas forcément plans et dont les arêtes ne sont pas forcément rectilignes. Les données topologiques ne diffèrent pas d'un B-Rep représentant des polyèdres.

Traiter les opérations booléennes sur des objets non polyédriques demande la résolution de problèmes tels que l'intersection de surfaces.

Tout au long de cette section, les objets A et B sont les objets opérandes de l'opération booléenne et l'objet C est l'objet résultat.

4.2.1. Méthode par calcul direct.

La méthode présentée dans [MIL 93] implique la résolution de l'intersection de deux surfaces.

L'auteur reprend le schéma classique qui est :

- générer des ensembles de faces, arêtes et sommets contenant l'ensemble des faces, arêtes et sommets de C,
- tester les éléments de ces ensembles afin de déterminer quelles sont les diverses entités appartenant à la frontière de l'objet final.

L'ensemble des arêtes de l'objet résultat est donc contenu dans l'ensemble des arêtes de A, dans l'ensemble des arêtes de B et dans l'ensemble des arêtes d'intersection de A et B. Les arêtes de C jointes correctement forment les faces de C. L'auteur ne s'intéresse donc qu'au calcul des arêtes et de leur classification pour obtenir l'objet final. Les données géométriques (faces et sommets) et topologiques sont déduites des données géométriques et topologiques des arêtes.

De plus, seule l'intersection est implémentée, puisque l'union et la différence peuvent être déduites de l'intersection en utilisant la complémentarité d'un objet.

Les arêtes de C peuvent donc provenir des arêtes de A ou de B (type1), des courbes d'intersection entre les surfaces de A et de B (type2) ou de la combinaison des deux (une partie d'une arête de A peut appartenir à une courbe d'intersection entre une surface de A et une surface de B) (type3) (figure 13).

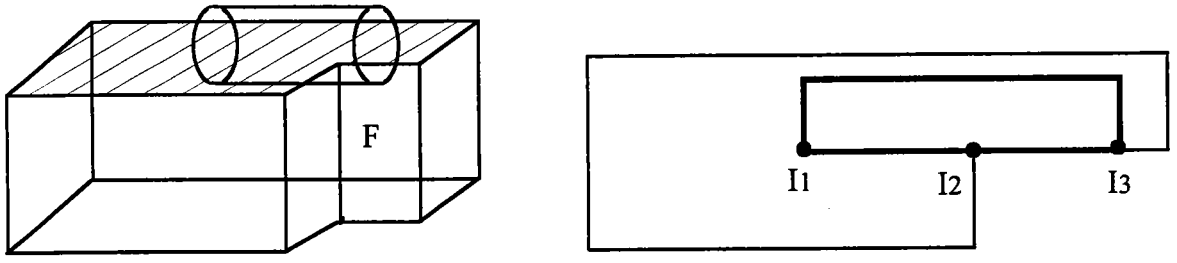


Figure 13 : Le cylindre est tangent à la boîte tronquée sur la face F. La face représentée à droite est la face grisée de la boîte tronquée. L'arête I_1I_3 est une arête d'intersection entre la face grisée et le cylindre. Elle se compose de deux parties. Une partie type1 qui est la portion I_1I_2 et une partie type3 qui est la portion I_2I_3.

Après le calcul des arêtes d'intersection, l'ensemble des arêtes peut être classé par rapport à C de la façon qui suit :

- une arête type1 de A appartient à l'intersection des deux objets si elle est intérieure à B (et réciproquement pour une arête type1 de B). Cette classification est équivalente à la classification d'un point intérieur de l'arête type1 par rapport à l'objet B,
- une arête type2 appartient forcément à l'intersection des deux objets,
- la classification d'une arête type3 nécessite plus d'informations pour déterminer si elle appartient à l'intersection. On recherche les sections des solides par rapport à un plan perpendiculaire à l'arête type3 et passant par son milieu. Si les intérieurs des deux sections sont disjoints alors l'arête type3 n'appartient pas à l'intersection.

A ce stade, l'ensemble des arêtes de C est connu. Ces arêtes sont ensuite reliées correctement afin de former les faces de C qui vont former l'objet final.

Pour calculer les arêtes d'intersection, il est nécessaire de calculer le support de ces arêtes qui est une intersection surface-surface. Malheureusement, dans cet article, l'auteur n'indique pas quel modèle il a employé ni quelle méthode d'intersection de surfaces il a utilisée. Des détails sur le modèle auraient permis de savoir s'il conservait et calculait les données géométriques de l'arête. Cependant, on peut se référer à un article du même auteur [MIL 87] où l'intersection des quadriques est traitée de façon exacte et paramétrique.

4.2.2. Méthode d'intersection de carreaux restreints.

Comme il est difficile d'obtenir des résultats corrects dans toutes les configurations possibles, les opérations booléennes basées directement sur les calculs d'intersection de surfaces ne peuvent être fiables.

De plus, même si les courbes d'intersection peuvent être calculées correctement, elles n'appartiennent pas forcément en totalité aux parties de surfaces définies dans les solides. Dans de tels cas, on utilise la notion de carreau restreint [CAS 89] [FAR 87].

Un carreau restreint (figure 14) est un carreau découpé par un ou plusieurs contours. Un carreau restreint est donc un carreau dont on ne conserve qu'une partie. Le carreau restreint est représenté par la fonction suivante :

$$r : D \subset \Re^2 \rightarrow \Re^3$$
$$(u,v) \mapsto r(u,v)$$

à laquelle est associée une liste de contours 2D.

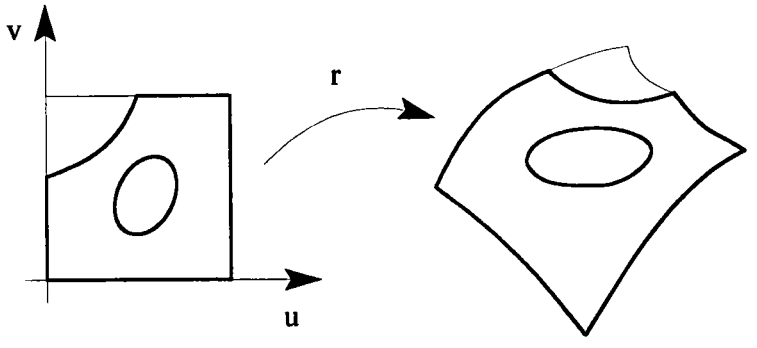


Figure 14 : Carreau restreint

La frontière bO d'un objet O représenté par un ensemble de carreaux restreints est définie par : $bO = \bigcup_{i=1}^{n} r_i(D_i)$.

Effectuer une opération booléenne sur A et B représentés par un ensemble de carreaux restreints pour obtenir un objet C est équivalent à chercher les parties des carreaux restreints de A et de B intervenant dans la frontière de C [CAB 89] c'est-à-dire :

$$bC = \left(\bigcup_{i=1}^{n} r_i(D_i^*)\right) \cup \left(\bigcup_{j=1}^{m} s_j(E_j^*)\right)$$

$$\text{avec } X_k^* = \{p \in X_k \,/\, p \in bC\}$$

L'algorithme est donc le suivant :

(1) intersection entre les carreaux de A et les carreaux de B,

(2) à partir des données de (1), construction des contours 2D sur chaque carreau et construction des domaines D_i^* et E_j^*.

(3) formation de l'objet final avec les domaines calculés en (2).

Les points les plus importants de cet algorithme sont le calcul de l'intersection de deux carreaux et la classification d'un domaine du carreau par rapport à l'autre objet.

Le calcul de l'intersection de deux carreaux se fait principalement par deux méthodes : par un suivi de courbes qui est particulièrement adapté au traitement des points singuliers [BAH 88] ou par la méthode de subdivision qui transforme un problème global (ici, l'intersection de deux carreaux restreints) en plusieurs sous-problèmes de difficulté moindre. Dans [CAS 87] on utilise cette dernière méthode en approchant les carreaux restreints par des triangles. Selon l'auteur, le choix de cette méthode s'explique parce qu'elle convient à d'autres parties de son algorithme. La méthode de calcul d'intersection de carreaux restreints est donc la suivante :

- approcher les carreaux par des triangles,
- calculer les polylignes d'intersection entre les carreaux à partir des intersections entre les triangles,
- raffiner les polylignes d'intersection afin de former les courbes d'intersection.

Le point le plus délicat de ce calcul est de relier correctement les segments d'intersection entre les triangles afin de constituer les polylignes d'intersection entre les carreaux. L'auteur indique que relier deux segments qui ont deux de leurs extrémités qui appartiennent au même voisinage peut conduire à des erreurs. La méthode choisie consiste à relier deux extrémités de segments si leur provenance topologique le permet.

L'ensemble des courbes d'intersection découpe le carreau en plusieurs domaines. Ces domaines doivent être classés par rapport à l'autre solide pour déterminer quels sont ceux qui participent à la frontière de l'objet final. Le classement de tous les domaines d'un carreau s'effectue à partir du classement d'un point du carreau.

Etant donné la classification d'un point P d'un domaine par rapport à un solide, l'ensemble des points dans le voisinage de P a la même classification. Par extension radiale, tous les points du domaine de P ont la même classification. Donc la classification d'un domaine se fait par classification d'un point du domaine. Lorsque l'on change de domaine c'est-à-dire lorsqu'une courbe d'intersection est traversée, la classification s'inverse et ainsi de suite jusqu'à parcourir l'ensemble des domaines du carreau. Ceci devient faux lorsque les objets ne sont pas réguliers. En résumé, tous les domaines du carreau peuvent être classés si l'on connaît la classification d'un point du domaine (voir figure 15). Sur les B-Rep, la classification d'un point de l'espace euclidien par rapport à un objet se traite par un lancer de demi-droite à partir du point considéré. Si le nombre d'intersections entre la demi-droite est l'objet est pair alors le point est extérieur sinon le point est intérieur. Cet algorithme doit tenir compte de tous les cas particuliers pouvant se présenter (par exemple, un sommet de l'objet appartient à la demi-droite).

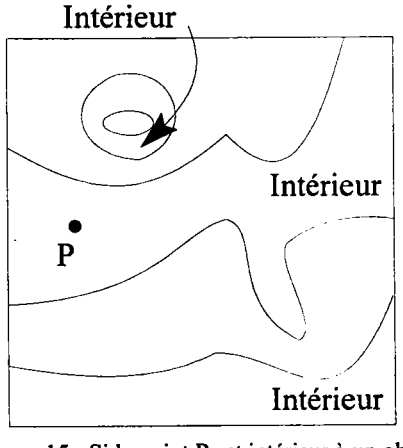


Figure 15 : Si le point P est intérieur à un objet O alors les domaines du carreau intérieurs à O sont rapidement trouvés.

4.2.3. Méthode par approximation polyédrique

Les avantages d'une approximation polyédrique sont les suivants :

- elle permet d'adapter facilement les opérations booléennes aux surfaces gauches ; l'extension des méthodes vues à la section 2 de ce chapitre n'est pas toujours évidente et peut demander de lourds traitements informatiques,
- les inconvénients associés à une intersection directe de surfaces disparaissent (par exemple, mauvaise connexion des courbes d'intersection).

De telles méthodes enrobent les opérations booléennes sur des objets polyédriques par un pré-traitement qui consiste à approcher les objets opérandes par des polyèdres, et un post-traitement qui modifie la frontière de l'objet final pour mettre à jour les surfaces intervenant dans la composition de cet objet.

La méthode de [TOT 91] s'appuie sur une approximation polyédrique pour donner une représentation topologique de l'objet final. Les données géométriques de l'objet résultat sont calculées à partir des courbes d'intersection entre les objets. La méthode s'effectue en quatre parties :

(1) calcul des courbes d'intersection (figure 16b),
(2) approximation polyédrique (figure 16c) à partir d'une approximation des arêtes curvilignes,
(3) opération booléenne entre les deux objets opérandes (figure 16d),
(4) création des surfaces de l'objet final (figure 16e) en utilisant les informations géométriques calculées dans la partie 1.

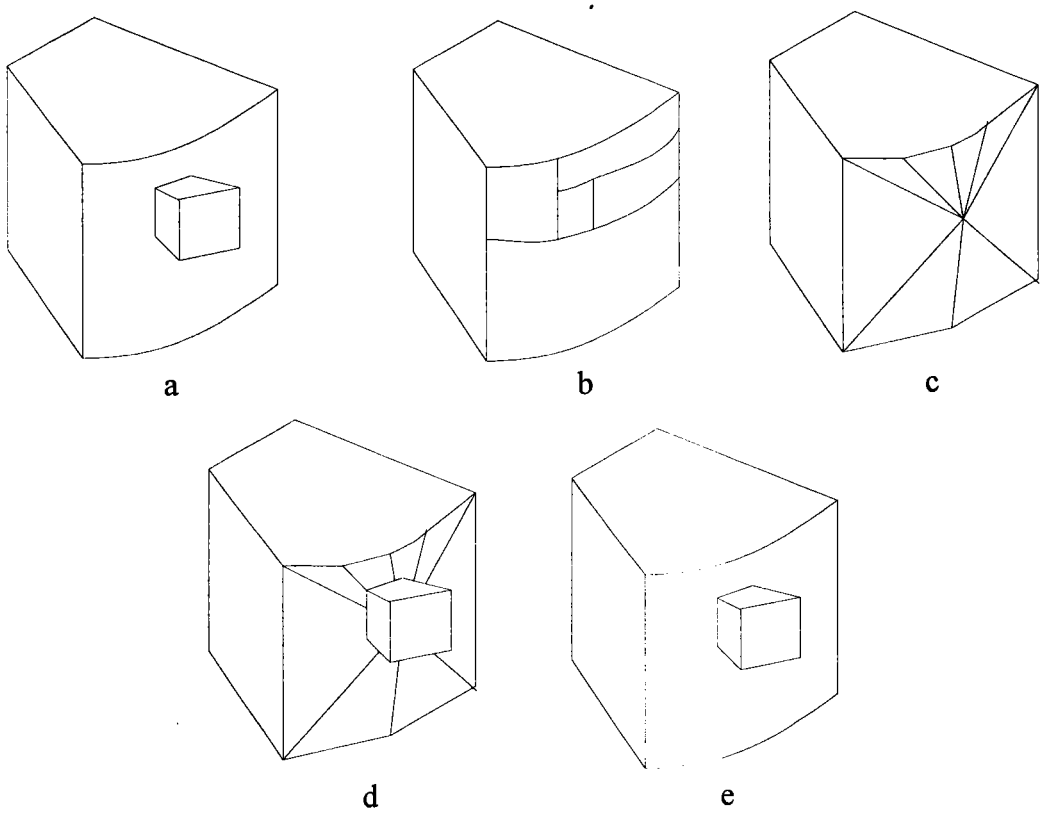


Figure 16 : Les différentes étapes des opérations booléennes par approximation polyédrique.
(a) les deux objets en vue cachée (b) courbes d'intersection pour un objet
(c) approximation polyédrique (d) objet résultat approché
(e) objet résultat

Pour la première partie, les points d'intersection entre les arêtes de l'objet A et les faces de l'objet B sont calculés (et réciproquement). À partir de ces points, les arêtes d'intersections entre les faces sont générées. Lorsque les supports des faces sont des quadriques, les données géométriques sont utilisées pour calculer les courbes d'intersection [JOH 91] [ELV 94], d'autant plus qu'au moins deux points de la courbe d'intersection sont connus. Les données topologiques des différentes intersection sont également stockées.

L'approximation polyédrique est indépendante de l'ensemble de ces informations : elle consiste pour chaque face à approcher les arêtes courbes par des segments. Pour ce faire, on construit un ensemble de points plus ou moins dense suivant la précision désirée. En reliant ces points, on obtient une polyligne qui est une approximation de l'arête courbe. Le centre de la face est ensuite calculé et relié à l'ensemble des points approchant les arêtes courbes afin de former plusieurs faces planes.

Après application de l'opération booléenne sur les deux objets polyédriques, l'objet résultat est obtenu. Ce dernier est une approximation de l'objet réel. L'objet comprenant les surfaces gauches est obtenu à partir de l'objet polyédrique en trois traitements :

- les arêtes provenant de l'approximation polyédrique sont supprimées,

- les sommets de l'approximation polyédrique sont également supprimés et les arêtes courbes sont réactualisées ; les sommets générés par l'exécution de l'opération booléenne et ne faisant pas partie des points d'intersection de la partie (1) de l'algorithme sont supprimés,
- les arêtes provenant de l'opération booléenne sont comparées avec les arêtes calculées dans la partie (1) de l'algorithme et les arêtes rectilignes sont transformées en arêtes courbes.

Les problèmes inhérents à cette famille de méthodes sont dus à l'approximation polyédrique. Dans [TOT 91], la surface exacte de l'objet est incluse dans l'approximation polyédrique. De ce fait, deux objets n'ayant aucune intersection avant l'approximation peuvent en avoir une après (et inversement si l'approximation polyédrique se fait en dessous de la surface de l'objet).

4.2.4. Conclusion

Les opérations booléennes sur des objets comprenant des surfaces gauches s'effectuent principalement de deux façons :

- soit par calcul direct des intersections surface-surface et/ou des intersections courbe-surface avec ou sans la notion de carreau restreint,
- soit par approximation polyédrique des deux objets opérandes de l'opération booléenne.

Les méthodes par calcul direct des intersections sont tributaires des inconvénients liés au calcul d'intersection de surfaces, à savoir :

- difficulté de détecter les singularités des courbes d'intersection comme le recouvrement de surfaces, des petites courbes d'intersection ou des points d'intersection,
- sensibilité aux imprécisions de calculs,
- temps de calcul parfois trop importants,
- impossibilité, en général, d'obtenir la courbe dans le même modèle mathématique que les surfaces.

Ces méthodes engendrent donc des problèmes de fiabilité dans les systèmes employant un calcul direct d'intersection de surfaces. Cependant lorsque les surfaces sont des quadriques, les intersections sont mieux maîtrisées.

Les méthodes par approximation polyédrique permettent une extension plus facile des opérations booléennes sur des objets comportant des surfaces gauches, en réutilisant les algorithmes existant sur des polyèdres. Les inconvénients de ces méthodes sont donc les inconvénients de la méthode employée sur des polyèdres. De plus, elles nécessitent un pré-traitement et un post-traitement pour rendre l'objet polyédrique et pour remettre l'objet dans la représentation initiale des objets opérandes (avec des surfaces quelconques). De par le pré-traitement, elles peuvent engendrer un changement géométrique des deux objets. Si l'approximation polyédrique est incluse dans la frontière de l'objet initial alors deux objets ayant une intersection avant l'approximation peuvent ne plus en avoir après (et inversement si la frontière de l'objet est incluse dans l'approximation).

5. OPÉRATIONS BOOLÉENNES SUR DES OBJETS NON-EULÉRIENS.

5.1. Définition.

Les objets définis à la section 2.1 sont aussi également appelés des objets Eulériens (une arête est partagée par deux faces uniquement). Les "r-sets" englobent les objets Eulériens et sont plus permissifs que ces derniers (une arête est partagée par plus de deux faces) [DES 92]. Au-dessus des "r-sets", se trouvent les objets non-Eulériens (figure 17). Ceux-ci permettent des faces ou des arêtes isolées dans la composition d'un objet.

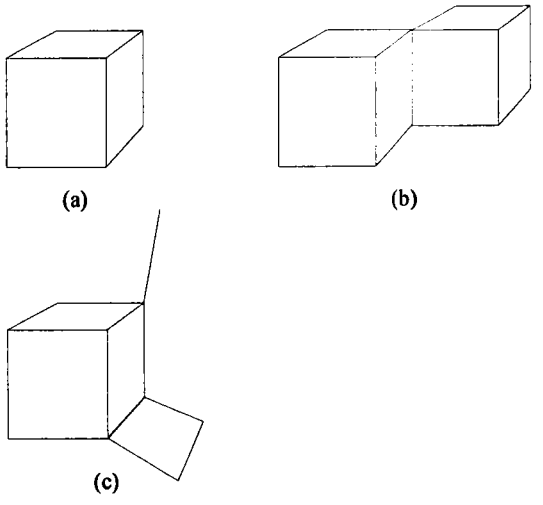


Figure 17 : Objets non-Eulérienes.
(a) : objet "r-set" avec une frontière Eulérienne
(b) : objet "r-set" avec une frontière non-Eulérienne
(c) : objet non-Eulériens

Il peut être utile en C.A.O. de détecter si deux objets sont en contact et dans l'affirmative vouloir connaître la région de contact. Un contact existe si les frontières des deux objets ont une intersection et si leurs intérieurs sont disjoints. Dans un tel cas, avoir formalisé une opération d'intersection régularisée peut être un inconvénient. C'est pourquoi les objets non Eulériens et les opérations booléennes associées doivent être implémentés pour résoudre ce type de problème.

De ce fait, les modèles supportant ce type d'objets doivent être définis. Le modèle B-Rep supporte très bien l'extension [MAS 93]. Un objet est composé d'une ou plusieurs faces. Une face est un polygone plan comportant plusieurs frontières connexes. Une frontière connexe peut être une suite fermée d'arêtes, une arête ou un sommet. Une arête est représentée par deux sommets et un sommet est un point de l'espace euclidien. A l'ensemble de ces données géométriques, il faut ajouter les relations topologiques. Un sommet est partagé par une ou plusieurs face(s), une arête peut être partagée par plusieurs faces.

Le modèle CSG fait aussi l'objet d'études pour s'adapter à ce type d'objets [ROR 91]. Les primitives ne sont plus seulement des volumes mais peuvent être aussi un sommet, une arête ou un polygone plan. Les opérations booléennes sur ces primitives ne sont plus régularisées et un nouvel opérateur est ajouté qui permet de "coller" deux primitives.

5.2. Opérations booléennes.

Les opérations booléennes reprennent le schéma classique des opérations booléennes sur des objets Eulériens (voir la section 2 de ce chapitre) :

- intersection entre les différentes entités,
- fusionnement et éclatement suivant l'opération booléenne effectuée.

L'intersection entre les différentes entités s'effectue dans l'ordre croissant de leurs dimensions [GCP 91]. Selon cet ordre, on trouve : les sommets, les arêtes et les faces. La méthode d'intersection se déroule en six étapes :

- intersection sommet-sommet (figure 18) : chaque sommet S_A de l'objet A est examiné avec chaque sommet S_B de l'objet B. Si les deux sommets sont très proches (c'est-à-dire si la distance $[S_A S_B]$ est inférieure à une certaine tolérance), les deux sommets sont marqués comme identiques ;

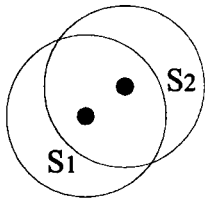


Figure 18 : intersection sommet-sommet.

- intersection sommet-arête (figure 19) : si la distance entre un sommet et le projeté orthogonal de ce sommet sur une arête est inférieure à une certaine tolérance alors les deux points sont marqués comme identiques. L'arête est divisée en ce point et donne deux nouvelles arêtes ;

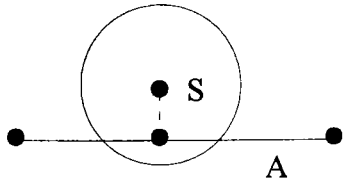


Figure 19 : intersection sommet-arête.

- intersection arête-arête (figure 20) : si deux arêtes possèdent leurs origines et leurs extrémités marquées comme identiques alors les deux arêtes sont considérées comme identiques. Si aucune extrémité n'est identique avec une autre alors la plus petite distance entre les arêtes est calculée. Ceci s'effectue en cherchant la droite perpendiculaire aux deux arêtes. Si les points d'intersection entre cette droite et les arêtes appartiennent respectivement aux arêtes alors les deux points d'intersection sont marqués comme identiques. Les arêtes sont découpées en ces points ;

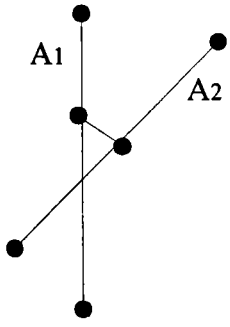


Figure 20 : intersection arête-arête.

- "intersection" sommet-face (figure 21) : à ce stade de l'étude des cas d'intersections, il n'est pas nécessaire de considérer l'intersection sommet-face sur la frontière de la face puisque toutes les intersections sommet-sommet et sommet-arête ont été effectuées. Si la distance entre le sommet considéré et le projeté orthogonal de ce sommet sur la face est inférieure à une certaine tolérance alors les deux points sont marqués comme identiques et un sommet isolé est créé sur la face ;

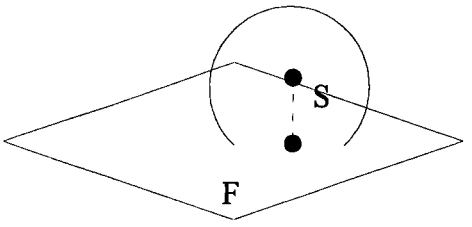


Figure 21 : intersection sommet-face

- intersection arête-face (figure 22) : si les extrémités de l'arête appartiennent à la face (c'est-à-dire si les extrémités de l'arête ont été marqués comme identiques au test précédent) alors il faut tester si l'arête est intérieure à la face. Dans ce cas, une nouvelle arête est créée sur la face et les frontières de la face sont mises à jour en conséquence. Si aucune extrémité de l'arête n'a de point identique avec ceux de la face alors le point d'intersection entre le support de l'arête et la face est cherché. Si ce point d'intersection appartient à l'arête alors un sommet isolé est créé sur la face et les deux sommets sont marqués comme identiques ;

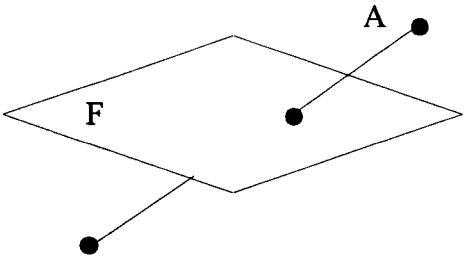


Figure 22 : intersection arête-face.

- intersection face-face (figure 23) : si toutes les entités de la première face sont identiques avec les entités de la seconde alors les deux faces sont marquées comme identiques. S'il existe plusieurs entités identiques, et si les deux faces ne sont pas coplanaires alors la droite d'intersection entre les deux faces est construite et les sommets marqués comme identiques sur cette droite sont triés. Les arêtes entre chaque sommet sont alors construites et les frontières des faces sont mises à jour.

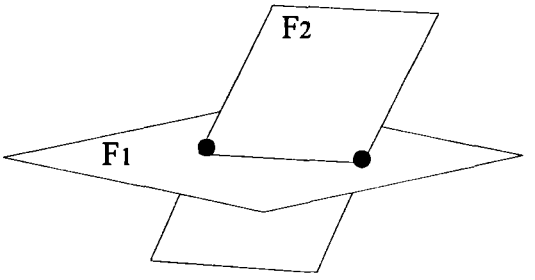


Figure 23 : intersection face-face.

Lors de la phase de fusionnement, toutes les entités marquées comme identiques sont fusionnées. Ensuite, chaque entité est choisie en fonction de l'opération booléenne appliquée.

Comme les opérations booléennes ne sont pas régularisées pour ce type d'objets, elles sont plus aisées à réaliser. En effet, de par la définition des objets, des cas particuliers disparaissent (notamment les faces coplanaires). Par contre, elles sont aussi soumises à l'imprécision des calculs : dans les intersections décrites plus haut, la coïncidence entre deux entités est testée avec une certaine tolérance. Cette dernière dépend de la position géométrique des deux objets et de leurs tailles.

6. CONCLUSION.

Actuellement, les deux modèles principaux sont le modèle B-Rep et le modèle CSG. Dans le modèle CSG, de nombreuses informations sur le frontière de l'objet sont implicites. Dès que l'on veut contraindre une entité de la peau de l'objet (par exemple donner une longueur à une arête), le modèle CSG n'est plus suffisant et le modèle B-Rep devient nécessaire. C'est pourquoi, il faut posséder des liaisons qui permettent de passer d'un modèle à l'autre. Les modeleurs actuels intègrent une évaluation du modèle CSG vers le modèle B-Rep. Quelques études existent sur la liaison B-Rep vers CSG [SHV 91] [SHV 93]. L'évaluation du modèle CSG vers le modèle B-Rep passe par la réalisation des opérations booléennes sur deux objets B-Rep. Les primitives de l'arbre CSG sont alors transformées en objets B-Rep. Ensuite, un parcours de l'arbre en évaluant à chaque nœud l'opération booléenne sur les deux objets fils permet d'obtenir la frontière de l'objet final.

De par la variété des besoins en C.A.O., les opérations booléennes doivent s'adapter aux différents types d'objets existants, à savoir :

- les objets à facettes planes (ou polyèdres),
- les objets comportant des surfaces gauches,
- les objets comportant des arêtes ou faces isolées et/ou dont les arêtes sont partagées par plus de deux faces (ou objet non-Eulériens).

Dans ce chapitre, nous venons d'expliquer les différentes méthodes que nous avons recensées pour chaque type d'objet nommé ci-dessus. Ces méthodes peuvent être résumées par le tableau suivant :

	0bjets à faces planes	Objets à faces quelconques	Objet non-Eulériens
Intersection arête-face	*oui*	*oui avec approximation*	*non*
Polygone de section	*oui*	*oui avec approximation*	*non*
Découpage de faces	*oui*	*oui avec approximation*	*non*
Méthode du pan	*oui*	*oui avec approximation*	*oui*
Méthode des domaines	*oui*	*oui avec approximation*	*oui*
Méthode par faces	*oui*	*oui avec approximation*	*oui*
Méthode par arbre octal	*oui avec passage arbre octal*	*oui avec passage arbre octal*	*oui avec passage arbre octal*
Méthode par calcul direct	*non*	*oui*	*non*
Intersection de carreaux restreints	*non*	*oui*	*non*
Intersection par ordre croissant des dimensions	*oui*	*non*	*oui*

Les méthodes s'adressant plus particulièrement aux polyèdres se regroupent en deux familles : celles travaillant directement sur la frontière de l'objet et celles utilisant un modèle intermédiaire (arbre octal). Les méthodes qui permettent l'intégration des surfaces gauches dans les objets utilisent les algorithmes sur les polyèdres en ajoutant une phase de polygonalisation de l'objet ou bien travaillent directement sur la peau de l'objet. Dans ce dernier cas, ces méthodes sont tributaires des inconvénients liés aux algorithmes de calcul d'intersection de surfaces.

La plupart de ces méthodes s'appuie sur la suite d'opérations "calculer les intersections", "combiner", "classer" et "extraire" l'ensemble des entités (sommets, arêtes et faces) formant les deux objets opérandes. Les intersections ne portent pas sur les mêmes entités (intersections arête-face, intersection face-face ou intersection surface-surface) et le traitement pour classer les entités diffère également (polygone de section permettant de partager les objets en extérieur-intérieur, classement de sous-faces en intérieur-extérieur par classement d'un point, transformation d'un objet en graphe pour classer toutes les faces appartenant à un même domaine, tests d'entités identiques par ordre croissant des dimensions des entités pour les objets non-Eulériens).

Les principaux inconvénients de ces méthodes sont :

- leur sensibilité aux imprécisions de calcul ; les méthodes sont confrontées au problème de la gestion d'epsilon notamment lors de la comparaison de deux réels ou du calcul d'intersection ;
- leur temps d'exécution important pour arriver parfois à un résultat erroné ;
- la gestion de nombreux cas particuliers ;
- leur difficulté à s'adapter à d'autres familles d'objets que les objets polyédriques (comportant des surfaces gauches ou non-Eulériens).

Dans le chapitre suivant, nous proposons une méthode permettant de résoudre une partie de ces problèmes (la gestion des cas particuliers, la robustesse du résultat et l'extension possible à des objets autres que des polyèdres). Le problème de la gestion d'erreurs ne fait pas partie de notre propos car il est en cours de traitement dans notre laboratoire.

CHAPITRE 2.

OPÉRATIONS BOOLÉENNES SUR DES POLYÈDRES : LA MÉTHODE DES SECTIONS.

1. INTRODUCTION.

Afin de résoudre les problèmes communs aux diverses méthodes d'opérations booléennes existantes (à savoir la gestion de nombreux cas particuliers, la robustesse du résultat et l'extension possible à des objets autres que des polyèdres), notre objectif est de définir et mettre en œuvre un algorithme d'opérations booléennes sur des objets polyédriques, en tenant compte des contraintes suivantes :

- les objets initiaux sont définis par leurs frontières,
- l'objet final doit être défini par ses frontières,
- la théorie développée pour les objets polyédriques doit tenir compte des objets comportant des surfaces gauches (voir chapitre 3) et à des objets non-Eulériens.

L'algorithme que nous décrivons dans ce chapitre répond aux deux premières contraintes en ne traitant que des objets à faces planes. Bien que la troisième contrainte pour les objets non-Eulériens ne soit pas traitée dans ce manuscrit, la méthode a été définie de telle sorte qu'elle puisse s'adapter à ce type d'objets.

2. IDÉE PRINCIPALE ET BASES THÉORIQUES.

Dans cette section, nous allons donner une approche de la méthode quel que soit le type des objets opérandes et décrire les bases théoriques qui permettent d'y accéder. Pour ce faire, nous donnons quelques notations.

Notons $\cup$ l'opérateur booléen "union" trois dimensions. Si l'opérateur booléen est régularisé nous le noterons $\cup^*$.

Notons $\cap$ l'opérateur booléen "intersection" trois dimensions. Si l'opérateur booléen est régularisé nous le noterons $\cap^*$.

Notons $-$ l'opérateur booléen "différence" trois dimensions. Si l'opérateur booléen est régularisé, nous le noterons $-^*$.

A l'aide de ces notations, nous pouvons définir notre but. Soient deux objets O_1 et O_2 définis par les frontières, nous voulons obtenir un objet O_3 également défini par les frontières tel que O_3 soit le résultat d'une opération booléenne sur O_1 et O_2 c'est à dire :

$$O_3 = O_1 \cup O_2$$
ou $$O_3 = O_1 \cap O_2$$
ou $$O_3 = O_1 - O_2$$
ou $$O_3 = O_2 - O_1.$$

Le fait de s'imposer que les objets soient définis par leurs frontières nous a conduit à proposer une méthode générale basée sur l'étude de chaque face.

Notre intérêt se porte donc sur la frontière des deux objets et plus précisément sur les faces de chaque objet opérande. Soit $F(O_1)$ et $F(O_2)$ les faces de l'objet O_1 et les faces de l'objet O_2 respectivement. Notons $M(O_1)$ et $M(O_2)$ la matière de O_1 et la matière de O_2 respectivement.

L'objet O_1 (respectivement l'objet O_2) est déduit de $F(O_1)$ et $M(O_1)$ (respectivement $F(O_2)$ et $M(O_2)$) :

$O_1 = F(O_1) \cup M(O_1)$

$O_2 = F(O_2) \cup M(O_2)$.

Montrons que l'ensemble des faces de l'union ou de l'intersection ou de la différence de O_1 et O_2 peut s'exprimer en fonction de $F(O_1)$, $F(O_2)$, $M(O_1)$ et $M(O_2)$. Pour ce faire, nous indiquons que le terme $[F(O_1) \subset M(O_1)]$ doit être traduit par "*l'ensemble des parties des faces de l'objet O_1 incluses dans la matière de l'objet O_2*".

Par définition de l'union :

$$O_1 \cup O_2 = (O_1 \not\subset O_2) \cup (O_2 \not\subset O_1)$$
$$O_1 \cup O_2 = \left([F(O_1) \cup M(O_1)] \not\subset [F(O_2) \cup M(O_2)]\right) \cup \left([F(O_2) \cup M(O_2)] \not\subset [F(O_1) \cup M(O_1)]\right)$$

Cependant, les faces de l'objet résultat ne peuvent provenir que des faces des objet O_1 et O_2 (les matières des objets ne peuvent pas produire de faces). Donc :

$$F(O_1 \cup O_2) = [F(O_1) \not\subset M(O_2)] \cup [F(O_2) \not\subset M(O_1)] \cup [F(O_1) \cup F(O_2)]$$
$$F(O_1 \cup O_2) = [F(O_1) - (F(O_1) \cap M(O_2))] \cup [F(O_2) - (F(O_2) \cap M(O_1))] \cup [F(O_1) \cup F(O_2)]$$
$$F(O_1 \cup O_2) = [F(O_1) - M(O_2)] \cup [F(O_2) - M(O_1)] \cup [F(O_1) \cup F(O_2)]$$

Par définition de l'intersection :

$$O_1 \cap O_2 = (O_1 \subset O_2) \cup (O_2 \subset O_1)$$
$$O_1 \cap O_2 = \left([F(O_1) \cup M(O_1)] \subset [F(O_2) \cup M(O_2)]\right) \cup \left([F(O_2) \cup M(O_2)] \subset [F(O_1) \cup M(O_1)]\right)$$

De même :

$$F(O_1 \cap O_2) = [F(O_1) \subset M(O_2)] \cup [F(O_2) \subset M(O_1)] \cup [F(O_1) \cap F(O_2)]$$
$$F(O_1 \cap O_2) = [F(O_1) \cap (F(O_1) \cap M(O_2))] \cup [F(O_2) \cap (F(O_2) \cap M(O_1))] \cup [F(O_1) \cap F(O_2)]$$
$$F(O_1 \cap O_2) = [F(O_1) \cap M(O_2)] \cup [F(O_2) \cap M(O_1)] \cup [F(O_1) \cap F(O_2)]$$

Par définition de la différence :

$$O_1 - O_2 = (O_1 \not\subset O_2) \cup (O_2 \subset O_1)$$
$$O_1 - O_2 = \left([F(O_1) \cup M(O_1)] \not\subset [F(O_2) \cup M(O_2)]\right) \cup \left([F(O_2) \cup M(O_2)] \subset [F(O_1) \cup M(O_1)]\right)$$

Enfin :

$$F(O_1 - O_2) = [F(O_1) \not\subset M(O_2)] \cup [F(O_2) \subset M(O_1)] \cup [F(O_1) - F(O_2)]$$
$$F(O_1 - O_2) = [F(O_1) - (F(O_1) \cap M(O_2))] \cup [F(O_2) \cap (F(O_2) \cap M(O_1))] \cup [F(O_1) - F(O_2)]$$
$$F(O_1 - O_2) = [F(O_1) - M(O_2)] \cup [F(O_2) \cap M(O_1)] \cup [F(O_1) - F(O_2)]$$

L'ensemble des faces de l'union des objets O_1 et O_2 ou de l'intersection des objets O_1 et O_2 ou de la différence de l'objet O_1 par l'objet O_2 (c'est-à-dire respectivement $F(O_1 \cup O_2)$, $F(O_1 \cap O_2)$ et $F(O_1 - O_2)$) est composé de trois termes :

$$F(O_1 \otimes O_2) = [F(O_1) \otimes M(O_2)] \cup [F(O_2) \otimes M(O_1)] \cup [F(O_1) \otimes F(O_2)]$$

(Équation (1))

dans lesquels $\otimes$ est une opération booléenne en trois dimensions. Lorsque cette opération booléenne s'applique à des objets réguliers, elle doit être régularisée.

Le premier et le deuxième termes de l'équation (1) représentent une opération booléenne entre les faces d'un objet et la matière de l'autre. Comme spécifié dans l'équation (1), l'opération booléenne est en trois dimensions. Cependant, algorithmiquement, l'opération booléenne s'effectue sur chaque face, c'est donc une opération booléenne en deux dimensions. Cependant les deux opérandes de cette opération doivent être des entités deux dimensions. Déterminer la matière de l'objet par une entité deux dimensions revient à chercher la matière de l'objet suivant le plan de la face traitée. Ceci implique qu'il faut trouver la section d'un objet par un plan donné. Le premier et le deuxième termes sont donc des opérations booléennes entre les faces des deux objets et les sections de ces objets par rapport au plan des faces. Le troisième et dernier terme de l'équation (1) représente, quant à lui, le cas particulier où des faces du premier objet et des faces du second objet sont coplanaires (ou cas de contact entre les frontières).

Les différentes étapes de l'algorithme sont donc les suivantes :

Début
Pour chacune des faces de O_1 et O_2
    déterminer la section du second objet par le plan de la face;
    calculer une opération booléenne 2D appliquée à la face et à la section;
FinPour
construire O_3 avec l'ensemble de faces généré par la boucle;
Fin

Il est à noter que l'intérêt majeur de l'algorithme réside dans le fait que la complexité de l'algorithme est ramenée en deux dimensions. De plus, le fait de proposer une méthode générale basée sur l'étude de chaque face permet de résoudre facilement le cas des faces isolées que les objets non-Eulériens peuvent comporter.

En résumé :

L'opération booléenne en deux dimensions est choisie en fonction de l'opération booléenne en trois dimensions de la manière suivante :

- $O_1 \cup O_2$: est l'ensemble des parties des faces de O_1 extérieures à O_2 et des parties des faces de O_2 extérieures à O_1. Les opérations booléennes en deux dimensions sont une série de différences.

- $O_1 \cap O_2$: est l'ensemble des parties des faces de O_1 intérieures à O_2 et des parties des faces de O_2 intérieures à O_1. Les opérations booléennes en deux dimensions sont une série d'intersections.

- $O_1 - O_2$: est l'ensemble des parties des faces de O_1 extérieures à O_2 et des parties des faces de O_2 intérieures à O_1. Les opérations booléennes en deux dimensions sont une série de différences lors du traitement des faces de O_1 et une série d'intersections lors du traitement des faces de O_2.

Si l'on note $\cap_{2D}$ l'opérateur booléen "intersection" en deux dimensions, $-_{2D}$ l'opérateur booléen "différence" en deux dimensions et $\cup$ l'opérateur "union" ensembliste alors (figure 24) :

$$\cup = \{-_{2D}\} \cup \{-_{2D}\}$$
$$\cap = \{\cap_{2D}\} \cup \{\cap_{2D}\}$$
$$- = \{-_{2D}\} \cup \{\cap_{2D}\}$$

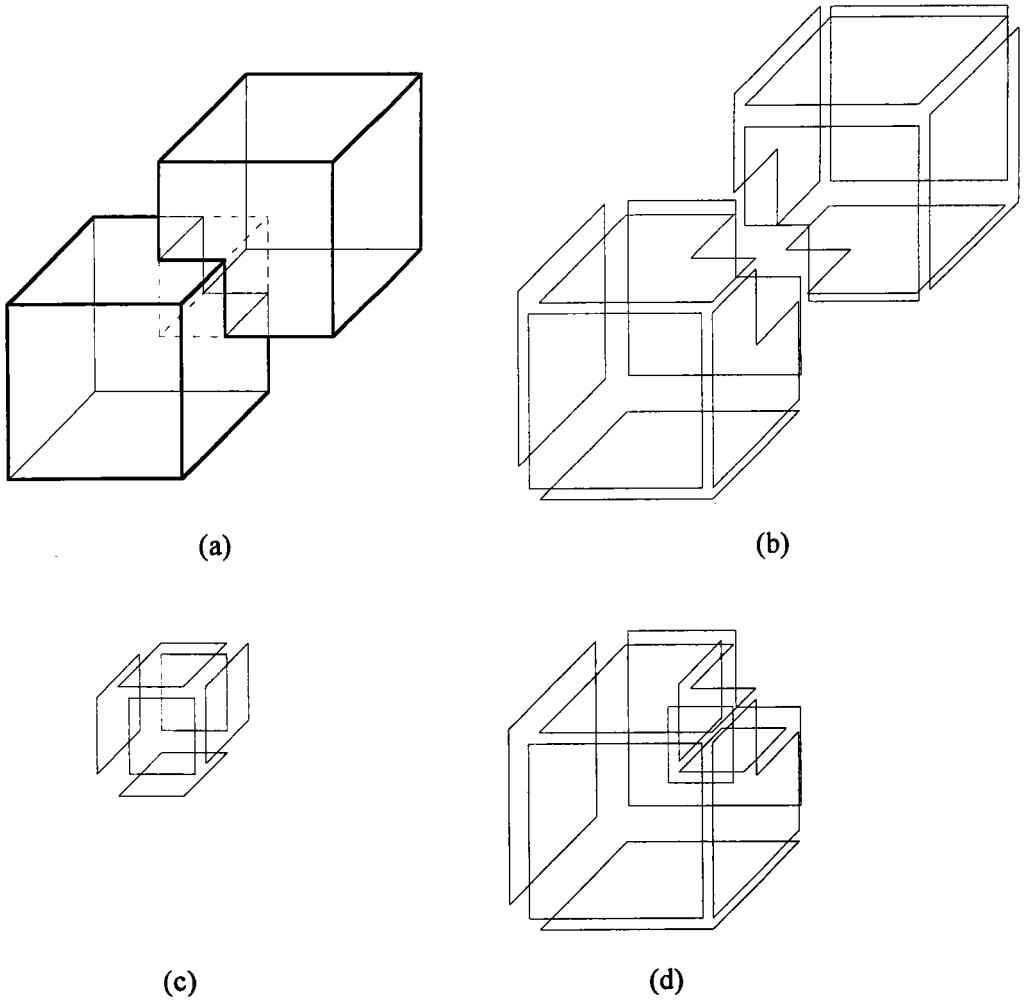


Figure 24 : Les faces du résultat d'une opération booléenne sur deux cubes.
(a) : positionnement des deux objets
(b) : union
(c) : intersection
(d) : différence

L'algorithme général est donc de la forme :

Début {OpérationBooléenneTroisDimensions}
    O_1, O_2 : objet;
    TraiterObjet (O_1 , O_2 , Opérateur , $\underline{EnsFace_1}$);
    TraiterObjet (O_1 , O_2 , Opérateur , $\underline{EnsFace_2}$);
    CréerObjet ($EnsFace_1$, $EnsFace_2$, $\underline{O_3}$);
Fin

Début {TraiterObjet}
    EnsFace := ∅ ;
    Pour chaque face F de l'objet O_1 **Faire**
        CréerSection (O_1 , Section);
        OpérationBooléenneDeuxDimensions (F , Section , Opérateur , Face);
        EnsFace := EnsFace ∪ {Face};
    FinPour
Fin

Dans la suite de ce chapitre, nous décrivons les procédures CréerSection (§3), OpérationBooléenneDeuxDimensions (§4) et CréerObjet (§6) qui s'appliquent aux objets polyédriques réguliers. Nous étudions aussi le troisième terme de l'équation (1) qui correspond au cas de faces coplanaires entre les deux objets (§5).

3. RECHERCHE DE LA SECTION.

La recherche de la section permet de trouver la matière du deuxième objet opérande pour une face donnée du premier objet opérande. Ce qui va permettre de résoudre, face par face, l'équation (1) qui est :

$$F(O_1 \otimes^* O_2) = \left[F(O_1) \otimes^* M(O_2)\right] \cup \left[F(O_2) \otimes^* M(O_1)\right] \cup \left[F(O_1) \otimes^* F(O_2)\right].$$

Soit F une face traitée de l'objet O_1. Soit O_2 le second objet opérande. On recherche la matière de l'objet O_2 afin de résoudre une des opérations booléennes du premier terme de l'équation (1) :

$$F \otimes^*_{2D} M(O_2).$$

Chercher la matière de l'objet O_2 pour exécuter l'opération booléenne $\otimes^*_{2D}$, revient à chercher la section de l'objet O_2 par le plan de la face F. Il faut donc suivre le contour de l'objet O_2 sur le plan de F. Pour ce faire, comme les supports des faces sont des plans, on calcule les intersections entre les plans des faces de l'objet O_2 avec le plan de la face F. Le résultat de ces intersections est un ensemble de droites. Deux droites provenant de deux faces adjacentes donnent un point d'intersection susceptible d'appartenir à la section. Ce point d'intersection n'est retenu que s'il appartient à l'arête commune entre les deux faces. Le contour de l'objet est formé en reliant ces points de façon adéquate.

L'intersection entre deux plans donne trois résultats possibles (figure 25) :

- les plans se coupent, la droite d'intersection est ajoutée à l'ensemble de droites servant à former la section. Cependant, la droite peut contenir ou non une portion du contour de la section (voir explication ci-dessus),
- les plans sont parallèles, le plan de la face suivante de l'objet O_2 est soumis au test d'intersection,
- les plans sont identiques. Il existe au moins deux faces (la face F et une face de l'objet O_2) qui sont coplanaires. L'intersection se poursuit pour chercher les autres faces de l'objet O_2 susceptibles d'être coplanaires avec la face F. Nous traitons ce cas particulier de coplanéarité dans la section 5.

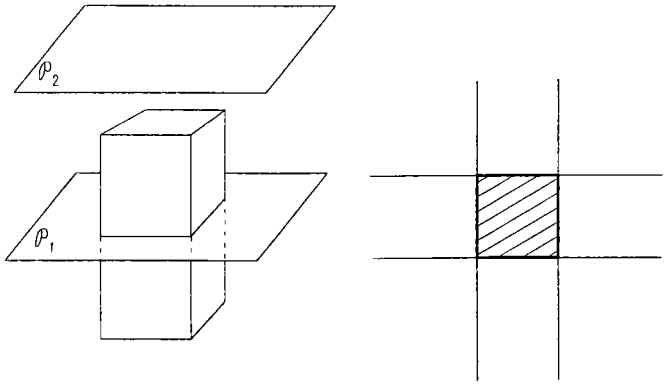


Figure 25 : Face et section. Le plan $\mathcal{P}_1$ donne une section contrairement au plan $\mathcal{P}_2$.

L'algorithme d'intersection entre le plan de la face traitée et les plans de l'objet est le suivant :

Début {IntersectionFaceObjet}
EnsDroite := ∅;
Pour chaque face F_j de l'objet O_j
Faire **Si** intersection = Droite **Alors**
EnsDroite := EnsDroite ∪ Droite;
FinSi
Si intersection = ∅ **Alors**
rien à faire;
FinSi
Si les plans sont identiques **Alors**
F et F_j sont regroupées;
FinSi
FinPour
Fin

Avec les intersections plan-plan, on construit la section de l'objet. Les intersections entre le plan de la face traitée et les plans de l'objet donnent un ensemble de droites. Chacune de ces droites peut porter, ou non, un segment [α , β] appartenant au contour de l'objet (figure 26).

Soit $\mathcal{D}$ une droite comportant une portion du contour de l'objet, alors il existe une droite $\mathcal{D}_\alpha$ et une droite $\mathcal{D}_\beta$ telles que α soit l'intersection des droites $\mathcal{D}$ et $\mathcal{D}_\alpha$ et β soit l'intersection des droites $\mathcal{D}$ et $\mathcal{D}_\beta$.

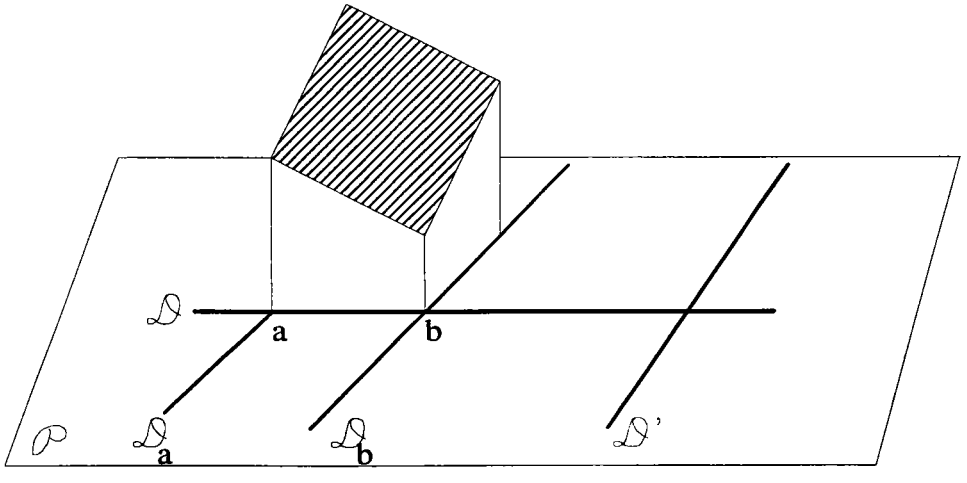


Figure 26 : Recherche de la section.
La droite $\mathcal{D}'$ qui provient de l'intersection entre le plan $\mathcal{P}$ et la face traitée ne contient aucune partie du contour de la section. Par contre, la droite $\mathcal{D}$ contient un segment de la section.

Les points α et β appartiennent au contour de l'objet sur lequel est réalisée la section.

Trouver la section de l'objet par rapport au plan de la face F revient à suivre les intervalles $[\alpha_j^i , \beta_j^i]$ où $i \in \mathbb{N}$ est le nombre de cycles dans la section et $j \in \mathbb{N}$ le nombre de points contenus dans le cycle i avec :

$\alpha_1^i = \beta_n^i$ et $\beta_j^i = \alpha_{j+1}^i$.

On parcourt les intervalles pour former l'ensemble des cycles. Un cycle commence en un des points d'intersection entre les droites. On suit le segment jusqu'au point extrémité puis on parcourt la droite suivante. On recommence jusqu'à revenir au point de début de cycle. Tant que tous les points d'intersection entre les droites n'ont pas été parcourus, on cherche un autre point de départ de cycle. Le parcours des segments $[\alpha_j^i , \beta_j^i]$ donne un ensemble de cycles qui sont tous considérés au départ comme des contours extérieurs ne comportant pas de contours intérieurs.

Cependant, de par la définition d'un modèle B-Rep, une section non vide est constituée d'au moins un contour extérieur et éventuellement d'un ou plusieurs contours intérieurs. La section recherchée est formée à partir des cycles et chaque cycle doit être correctement évalué (contour extérieur ou contour intérieur). On compare tous les cycles deux à deux en testant leur inclusion, tant qu'il y a un changement d'inclusion, et en ajoutant au fur à mesure des contours intérieurs de la façon suivante (figure 27) :

Début {ordonnancement de deux cycles C1 et C2}
  Si C1 et C2 ne sont pas disjoints **Alors**
    Si C1 $\subset$ C2 **Alors**
      Si C1 est inclus dans un trou de C2 **Alors**
        C1 est un contour extérieur;
      Sinonsi C1 entoure un ou plusieurs trous de C2 **Alors**
        C1 est un contour intérieur de C2;
        Les trous deviennent des contours extérieurs;
      Sinon C1 est un contour intérieur de C2;
      FinSi
    FinSi
  Sinon inverser les rôles de C1 et C2;
  FinSi
FinSi
Fin

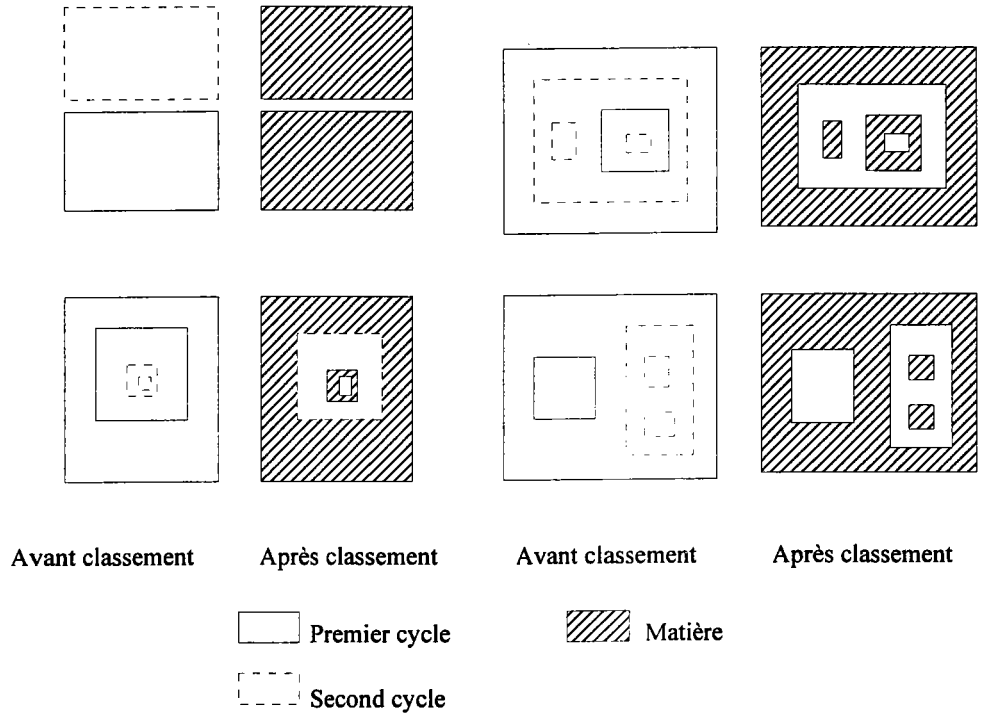


Figure 27 : Ordonnancement de deux cycles.

4. OPÉRATIONS BOOLÉENNES EN DEUX DIMENSIONS.

Dans la section précédente, la recherche de la section d'un objet par rapport au plan de la face traitée a été expliquée. Le traitement suivant, donné par l'algorithme de la section 2, consiste à calculer une opération booléenne en deux dimensions sur la face traitée et la section. Cette opération booléenne en deux dimensions dépend de l'opération booléenne en trois dimensions (voir section 2). L'opération booléenne, une fois appliquée sur la face et la section, permet de définir les parties de la face de l'objet opérande qui appartiennent à l'objet final.

Dans cette section, nous allons indiquer de manière sommaire comment calculer les opérations booléennes sur des polygones éventuellement troués (donc les opérations booléennes en deux dimensions). L'implémentation de ces opérations booléennes est basée sur l'algorithme d'Atherton-Weiler [WEA 77] [BMP 91].

Considérons deux polygones A et B représentés en fonction de leurs arêtes et de leurs sommets (B-Rep en deux dimensions) et définissons les opérations booléennes en deux dimensions.

4.1. Calcul de l'union.

L'algorithme de l'union de deux polygones consiste à calculer les intersections segment-segment entre les deux polygones. À partir de ces points d'intersection et des sommets des polygones, on cherche un point de A qui soit extérieur à B (ou réciproquement). On parcourt les deux polygones à tour de rôle suivant les cas d'intersection qui se présentent et les cas de tangence entre les segments des polygones pour former le polygone résultat (figure 28).

Début

Tant que ((A et B n'ont pas été parcourus) et (que l'on peut trouver un point P)) **Faire**

Choisir un point P du polygone A qui soit extérieur au polygone B (ou inversement);

$I_1 \leftarrow P$

Répéter

À partir de I_1, parcourir A jusqu'à une intersection I_2 avec B;

À partir de I_2, parcourir B jusqu'à une intersection I_1 avec A;

jusqu'à arriver au point P;

FinTantQue

Fin

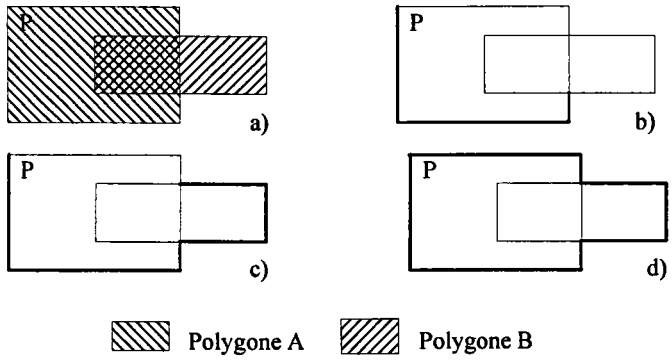


Figure 28 : Union de deux polygones A et B.
a) : les deux polygones
b), c), d) : itérations de la boucle de l'algorithme de l'union

4.2. Calcul de l'intersection.

L'algorithme de l'intersection de deux polygones diffère de celui de l'union par la recherche du point de départ. Ce dernier doit être intérieur aux deux polygones. Le parcours des deux polygones est ensuite identique à celui de l'union.

Début

Tant que ((A et B n'ont pas été parcourus) et (que l'on peut trouver un point P)) **Faire**
Choisir un point P du polygone A qui soit intérieur au polygone B (ou inversement);
$I_1 \leftarrow P$
Répéter
À partir de I_1, parcourir A jusqu'à une intersection I_2 avec B;
À partir de I_2, parcourir B jusqu'à une intersection I_1 avec A;
jusqu'à arriver au point P;
FinTantQue
Fin

4.3. Calcul de la différence.

L'algorithme de la différence du polygone A par le polygone B diffère de l'algorithme de l'union de ces deux polygones par la recherche de point de départ. Ce dernier doit appartenir au polygone A et être extérieur au polygone B. De plus, le sens de parcours du polygone B doit être inversé afin que le parcours donne bien une différence.

Début
Inverser le sens du polygone B.
Tant que ((A et B n'ont pas été parcourus) et (que l'on peut trouver un point P)) **Faire**
Choisir un point P du polygone A qui soit extérieur au polygone B ;
$I_1 \leftarrow P$
Répéter
À partir de I_1, parcourir A jusqu'à une intersection I_2 avec B;
À partir de I_2, parcourir B jusqu'à une intersection I_1 avec A;
jusqu'à arriver au point P;
FinTantQue
Fin

4.4. Conclusion.

Les sections 3 et 4 de ce chapitre permettent de calculer l'opération booléenne entre les faces du premier objet et la matière du second objet (c'est-à-dire la section), et réciproquement.

Toutefois, les opérations booléennes en deux dimensions sur les faces et les sections ne prennent pas en compte les cas de contact entre les frontières des deux objets polyédriques. Ce cas de contact se traduit par des faces des deux solides qui appartiennent au même plan. La section suivante permet de résoudre ce cas particulier.

5. TRAITEMENT DES FACES COPLANAIRES.

Le seul cas particulier pour la résolution des opérations booléennes sur des objets polyédriques concerne les faces coplanaires regroupées lors de l'intersection plan-plan de la section 3 de ce chapitre.

5.1. Résolution théorique.

Ce cas particulier est formalisé plus particulièrement par le troisième terme de l'équation (1) de la section 2 à savoir :

$$F(O_1 \otimes^* O_2) = \left[F(O_1) \otimes^* M(O_2)\right] \cup \left[F(O_2) \otimes^* M(O_1)\right] \cup \left[F(O_1) \otimes^* F(O_2)\right].$$

Considérons deux faces F_1 et F_2 coplanaires telles que F_1 est une face du premier objet et F_2 une face du second objet.

Soit S_1 (respectivement S_2) la section du second (respectivement premier) objet par rapport au plan de F_1 (respectivement F_2).

L'équation (1) devient : $\left[F_1 \otimes^*_{2D} S_1\right] \cup \left[F_2 \otimes^*_{2D} S_2)\right] \cup \left[F_1 \otimes^*_{2D} F_2\right]$ **Équation (2).**

La résolution de ce cas procède en deux parties :

a- résolution de $F_1 \otimes^*_{2D} F_2$: puisque les faces sont coplanaires, elles représentent en fait un point de contact entre les deux objets. Donc, elles doivent être prises en compte pendant l'opération booléenne car ces faces (ou des parties de ces faces) peuvent appartenir à la frontière de l'objet résultat.

L'opération booléenne en deux dimensions doit être adéquate par rapport à l'opération booléenne en trois dimensions appliquée aux deux objets opérandes. Pour déterminer quelle opération booléenne en deux dimensions il faut appliquer aux faces coplanaires, il est nécessaire de savoir situer la matière des deux objets par rapport à ces deux faces. Pour ce faire, il est possible de connaître où se trouve la matière d'un objet par rapport à une face de deux façons :

- par information de proximité (ou voisinage) [TIL 80] [REV 85]. A chaque face d'un objet est associée une sphère dont le centre se situe au milieu de la face. La frontière du solide découpe la sphère en deux parties. La partie à l'intérieur du solide représente l'information de proximité de la face,
- par la normale sortante de la face. C'est cette seconde solution que nous avons choisie car elle est directement liée au sens de parcours du contour extérieur de la face.

L'opération booléenne en deux dimensions est donc fonction des deux normales aux faces afin de déterminer s'il y a seulement contact entre les deux objets ou si, au contraire, une réelle intersection de matières est présente (figure 29). Ceci garantit une régularisation de l'opération booléenne en trois dimensions ;

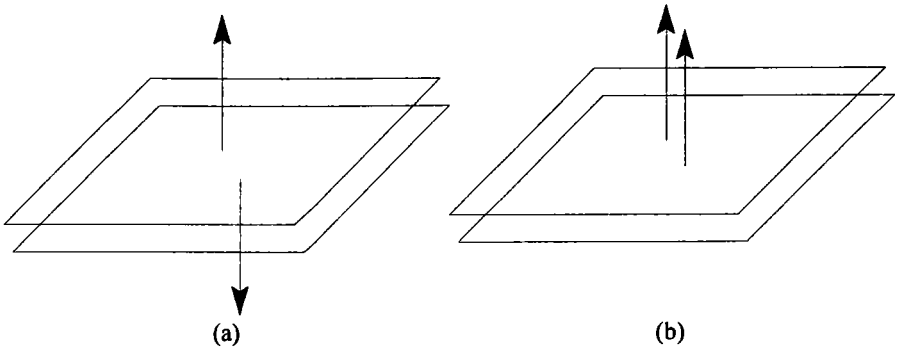


Figure 29 : Contact avec ou sans intersection de matière.
(a) : contact uniquement
(b) : contact et intersection de matière

b- résolution de $\left[F_1 \otimes^*_{2D} S_1\right] \cup \left[F_2 \otimes^*_{2D} S_2\right]$: la résolution de a- n'est pas suffisante car seules les frontières des objets ont été considérées. Il faut aussi examiner le résultat de (1) par rapport aux matières des deux objets et donc par rapport aux sections pour obtenir les parties exactes des faces intervenant dans la composition de l'objet résultat. Dans le cas où les faces coplanaires ne sont pas identiques aux sections des deux objets, il est nécessaire d'ajouter (ou de retrancher) au résultat du traitement (1) les parties des sections qui interviennent (ou qui n'interviennent pas) dans l'opération booléenne principale.

En résumé, l'opération booléenne en deux dimensions dépend de celle appliquée en trois dimensions en tenant compte des normales aux deux faces. Soit {F} l'ensemble des faces issues de la résolution du cas particulier de coplanairité. À partir de l'équation (2) et des sens des deux normales, on obtient deux cas :

- si les deux normales sont dans la même direction (contact avec intersection de matière), alors :

$$\cup^* \Rightarrow \{F\} = \{\{\{F_1 \cup^*_{2D} F_2\} -^*_{2D} S_1\} -^*_{2D} S_2\}$$

$$\cap^* \Rightarrow \{F\} = \{F_1 \cap^*_{2D} F_2\} \cup^* \{F_1 \cap^*_{2D} S_1\} \cup^* \{F_2 \cap^*_{2D} S_2\}$$

$$-^* \Rightarrow \{F\} = \{F', F''\} \text{ t.q. } \{F'\} = \{\{F_1 -^*_{2D} F_2\} -^*_{2D} S_1\}$$
$$\{F''\} = \{F_2 \cap^*_{2D} S_2\}$$

- sinon (contact sans intersection de matière), les résultats sont les suivants :

$\cup^* \Rightarrow \{F\} = \{F',F''\}$ t.q. $\{F'\} = \{\{F_1 -^*_{2D} F_2\} -^*_{2D} S_1\}$

$\{F''\} = \{\{ F_2 -^*_{2D} F_1\} -^*_{2D} S_2\}$

$\cap^* \Rightarrow \{F\} = \{F',F''\}$ t.q. $\{F'\} = \{\{F_1 -^*_{2D} F_2\} \cap^*_{2D} S_1\}$

$\{F''\} = \{\{ F_2 -^*_{2D} F_1\} \cap^*_{2D} S_2\}$

$-^* \Rightarrow \{F\} = \{F',F''\}$ t.q. $\{F'\} = \{F_1 -^*_{2D} S_1\}$

$\{F''\} = \{\{ F_2 -^*_{2D} F_1\} \cap^*_{2D} S_2\}$

5.2. Exemples de faces coplanaires.

5.2.1. Faces égales aux sections.

Nous donnons ci-après l'ensemble des parties des faces entrant dans la peau de l'objet résultat lorsque les faces coplanaires sont égales aux sections (i.e. $F_1 = S_2$ et $F_2 = S_1$).

Si l'opération booléenne en trois dimensions est une union et si les normales sortantes sont dans la même direction, alors l'opération booléenne en deux dimensions est une union (figure 30a). Si les normales sortantes ont des directions opposées alors on obtient deux faces : la première est le résultat d'une différence entre F_1 et F_2, la seconde est une différence entre F_2 et F_1 (figure 31b).

Si l'opération booléenne en trois dimensions est une intersection et si les normales sortantes sont dans la même direction, alors l'opération booléenne en deux dimensions est une intersection (figure 30b). Si les normales sortantes ont des directions opposées, alors le résultat est vide (figure 31b).

Si l'opération booléenne en trois dimensions est une différence (du premier objet par le second objet) et si les normales sont dans la même direction, alors l'opération booléenne en deux dimensions est une différence de la face appartenant au premier objet par la face appartenant au second objet (figure 30c). Si les normales ont des directions opposées, alors la face du premier objet reste inchangée (figure 31c).

Le tableau suivant permet de résumer les différentes opérations booléennes en deux dimensions qu'il faut appliquer sur les deux faces (lorsqu'elles sont égales aux sections) en fonction du sens de la normale.

Opération booléenne 3D	Normales dans le même sens	Normales de sens opposé
Union	$F_1 \cup F_2$	$F_1 - F_2$ et $F_2 - F_1$
Intersection	$F_1 \cap F_2$	$\varnothing$
Différence	$F_1 - F_2$	F_1

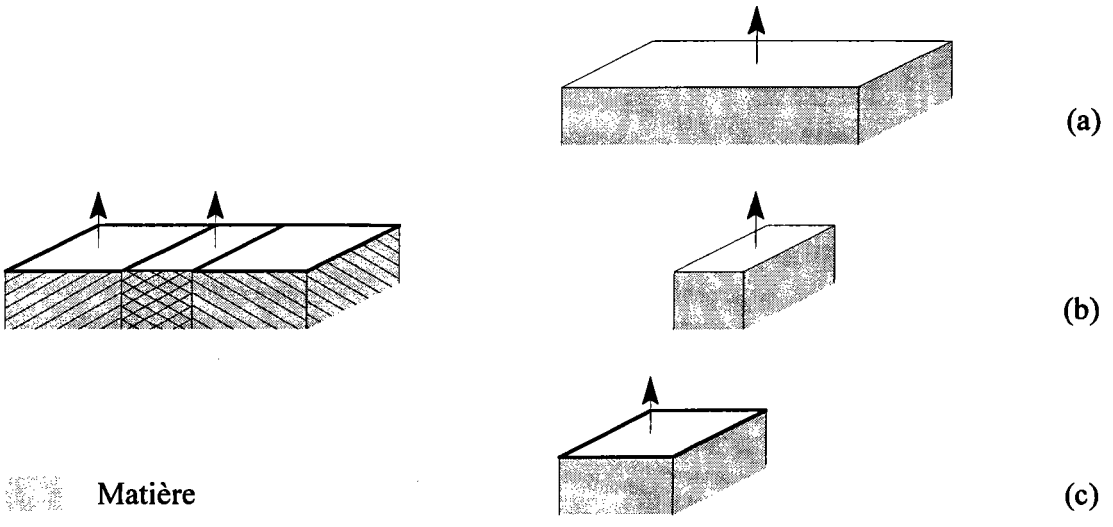


Figure 30 : Lorsque les normales aux faces sont dans la même direction.
(a) union,
(b) intersection,
(c) différence.

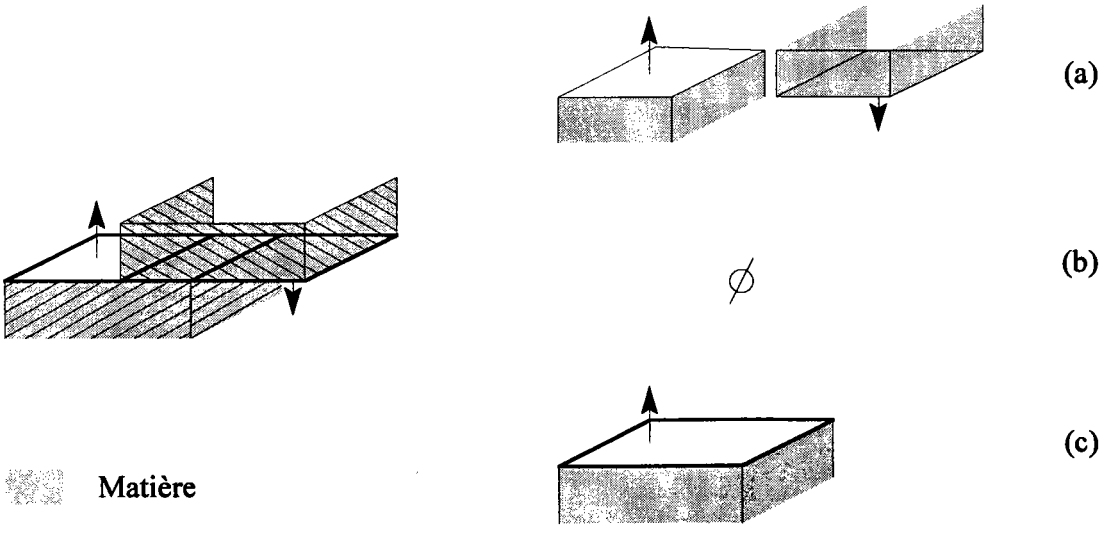
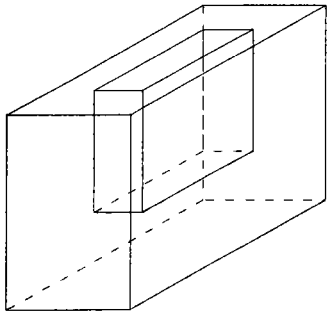


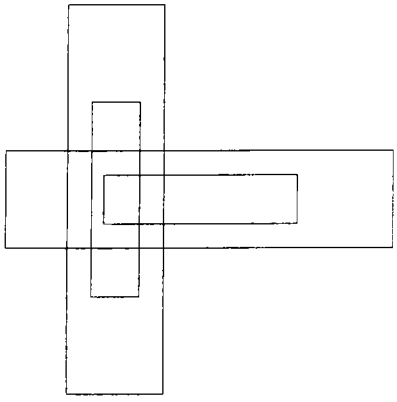
Figure 31 : Lorsque les normales aux faces n'ont pas la même direction.
(a) union,
(b) intersection,
(c) différence.

5.2.2. Faces non égales aux sections.

Prenons l'objet O suivant vu en filaire :

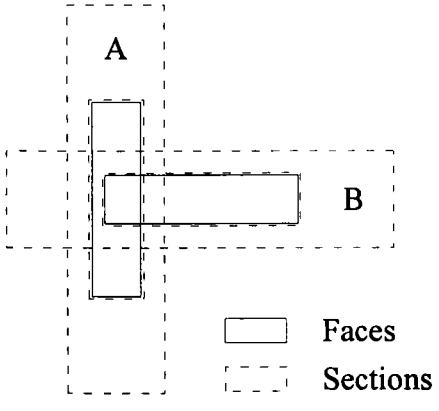


Prenons les deux objets opérandes de l'opération booléenne identiques à O et positionnés comme suit (vue de dessus) :



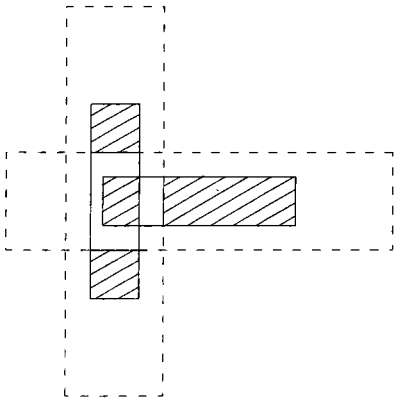
Les deux faces inférieures des trous (ayant la coordonnée en z la plus basse) sont des faces coplanaires. Les sections des deux objets passant par le plan des deux faces coplanaires ne sont pas égales aux faces (les faces correspondent aux trous des sections).

Donnons les résultats du traitement de ces deux faces pour les trois opérations booléennes possibles appliquées aux deux objets sachant que les deux normales sont dans le même sens.

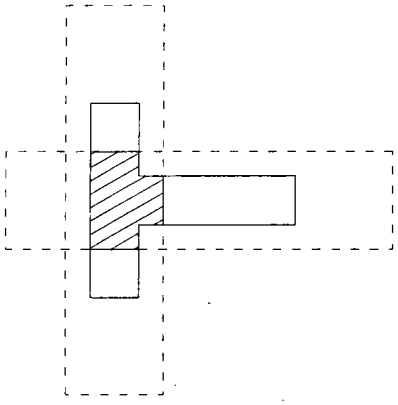


Dans les schémas suivants, les parties hachurées représentent les parties de faces entrant dans la composition de l'objet final.

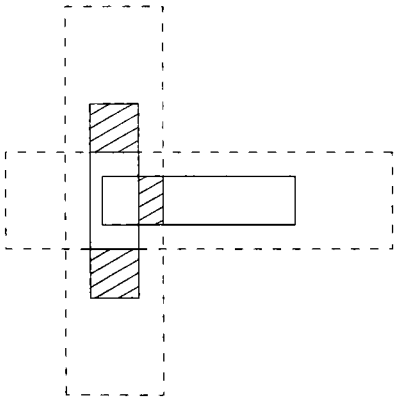
Le schéma suivant donne le résultat pour l'union 3D (A∪B) :



Le schéma suivant donne le résultat pour l'intersection 3D (A∩B) :



Le schéma suivant donne le résultat pour la différence 3D (A-B) :



6. RECONSTRUCTION DE L'OBJET RÉSULTAT SUIVANT LA DÉFINITION DU MODÈLE B-REP.

A ce stade, l'ensemble des parties de faces intervenant dans la frontière de l'objet final a été calculé. Il reste à représenter l'objet final suivant le modèle B-Rep à partir de l'ensemble de ces parties de faces.

Les points des faces de cet ensemble doivent être correctement reliés pour former l'objet résultat. En effet, l'ensemble des données géométriques de l'objet résultat a déjà été calculé : ce sont les points des faces. Il reste à former les relations topologiques de l'objet final. Le but est donc d'éliminer la redondance sur la création des points (un sommet d'un même objet ne peut être déclaré plusieurs fois dans le modèle). Une des solutions aurait été de comparer les coordonnées des faces de l'objet résultat afin de permettre la liaison topologique. Mais cette solution n'est pas souhaitable car elle amène forcément à des erreurs dues à l'imprécision des calculs sur des réels. C'est pourquoi les points identiques du solide vont être retrouvés dans l'ensemble des faces grâce à la topologie dont ils sont issus.

Seuls les cas Eulériens sont décrits puisque les faces et les sections sont des contours réguliers. Les différents cas proviennent des intersections possibles entre la face et la section et ils sont au nombre de quatre :

- intersection entre une arête de la face et un segment de la section,
- un sommet de la face appartient à un segment de la section,
- un point de la section appartient à une arête,
- un sommet de la face correspond à un point de la section.

Afin d'obtenir un algorithme robuste, nous avons formalisé ces différents cas de façon rigoureuse en utilisant différentes notations mathématiques que nous explicitons dans la section suivante.

6.1. Notations mathématiques, définitions et propriétés.

Afin de formaliser de façon rigoureuse la construction d'un objet à partir d'un ensemble de faces, nous avons choisi les notations et les définitions suivantes :

Notons $\mathcal{A}$: l'ensemble des arêtes créées dans le modèle,
$\mathcal{D}$: l'ensemble des droites d'intersection,
$\mathcal{F}$: l'ensemble des faces créées dans le modèle.

A partir de ces ensembles, définissons les fonctions suivantes :

FaceAdj : $(\mathcal{F}\times\mathcal{A}) \rightarrow \mathcal{F}$
$(f_i, a) \mapsto f_j$
telle que f_j est la face adjacente à la face f_i en l'arête a.

ArêteAdj : $(\mathcal{F}\times\mathcal{F}) \rightarrow \mathcal{A}$
$(f_i, f_j) \mapsto a$
telle que a soit l'arête du modèle partagée par les faces f_i et f_j. a prend une valeur par défaut lorsque les faces f_i et f_j ne sont pas adjacentes.

Face : $\mathcal{D} \rightarrow \mathcal{F}$
$d \mapsto f$
telle que f soit la face dans le modèle ayant engendré l'intersection de numéro d avec le plan de la face traitée.

Droite : $\mathcal{F} \rightarrow \mathcal{D}$
$f \mapsto d$
telle que d soit le numéro de l'intersection provoquée par la face f dans le plan de la face traitée.

ArêteSuiv : $(\mathcal{A}\times\mathcal{F}) \rightarrow \mathcal{A}$
$(a_i, f) \mapsto a_j$
telle que a_j soit l'arête qui suit l'arête a_i sur la face f.

ArêtePrec : $(\mathcal{A}\times\mathcal{F}) \rightarrow \mathcal{A}$
$(a_i, f) \mapsto a_j$
telle que a_j soit l'arête qui précède l'arête a_i sur la face f.

A l'aide de ces fonctions, on a les propriétés suivantes :
Soient $f_1 \in \mathcal{F}, f_2 \in \mathcal{F}, d \in \mathcal{D}$ et $a \in \mathcal{A}$.

1. *Droite* (*Face* (d)) = d
2. *Face* (*Droite* (f_1)) = f_1
3. *FaceAdj* (f_1 , *ArêteAdj* (f_1 , f_2)) = f_2
4. *ArêteAdj* (f_1 , *FaceAdj* (f_1 , a)) = a
5. *ArêteSuiv* (*ArêtePrec* (a , f_1) , f_1) = a
6. *ArêtePrec* (*ArêteSuiv* (a , f_1) , f_1) = a

6.2. Propriétés topologiques.

La méthode traite deux objets face par face. Pour chaque face du premier objet, la section du second objet par rapport au plan de la face du premier objet est calculée. Pour chaque face des deux objets, une opération booléenne en deux dimensions est appliquée sur la face et la section. Les intersections face-section sont calculées segment par segment. À ce stade, les informations topologiques de l'intersection sont sauvegardées afin de rechercher ultérieurement les points identiques.

Comme les objets sont réguliers, les sections de ces objets et leurs faces sont des polygones réguliers. En particulier, un sommet d'une face n'est partagé que par deux arêtes uniquement et un point de la section n'est partagé que par deux segments de la section. Nous pouvons donc énumérer tous les cas d'intersection entre une arête et un segment et ils sont au nombre de quatre.

Premier cas (figure 32) : une arête a une intersection avec un segment de la section mais ce point d'intersection ne correspond ni aux extrémités de l'arête ni aux extrémités du segment.

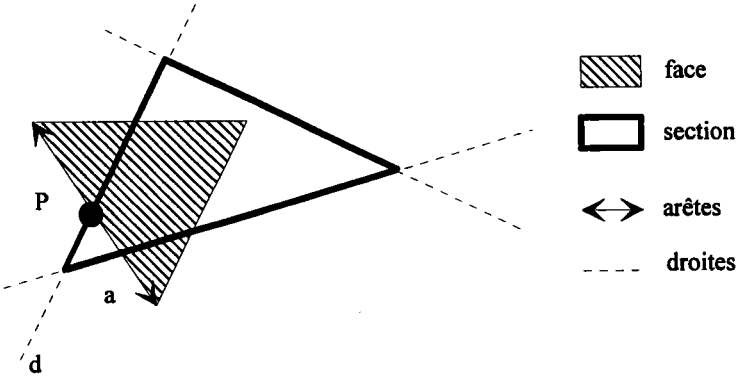


Figure 32 : Cas 1.

Deuxième cas (figure 33) : un sommet de la face appartient à un segment de la section.

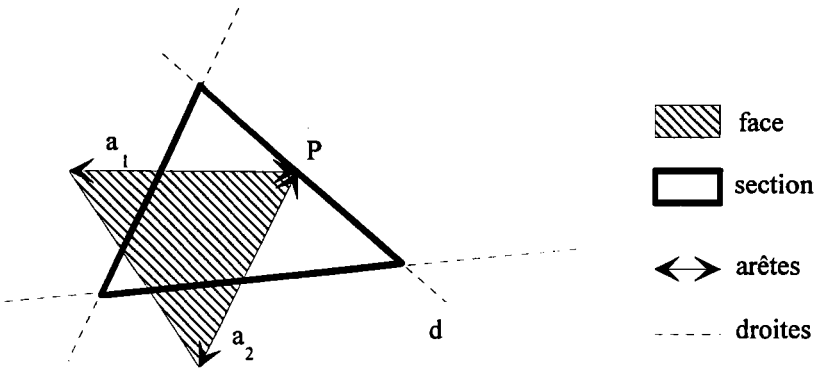


Figure 33 : Cas 2.

Troisième cas (figure 34) : un point de la section appartient à une arête.

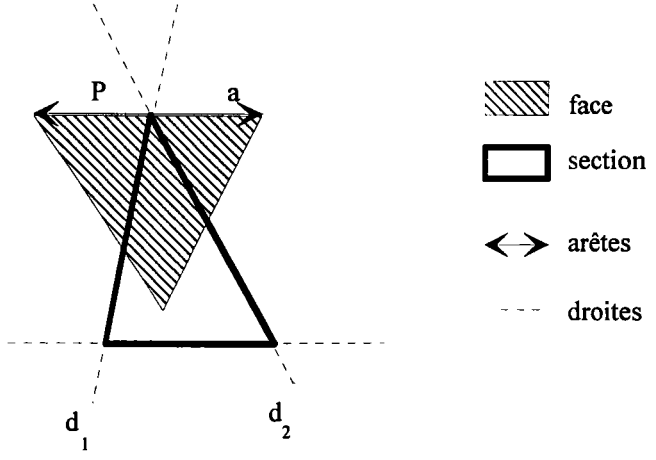


Figure 34 : Cas 3.

Quatrième cas (figure 35) : un sommet de la face correspond à un point de la section.

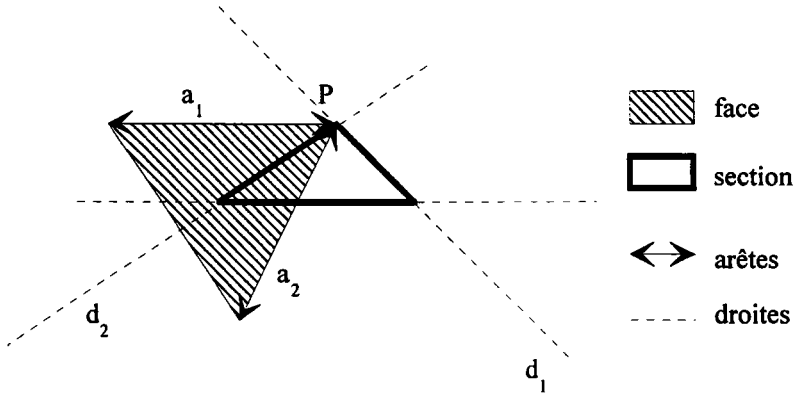


Figure 35 : Cas 4.

Notons *solide1*, le solide auquel appartient la face traitée. Notons *solide2*, le solide auquel appartient la section. Avec les quatre cas présentés ci-dessus, il est possible de retrouver tous les points identiques dans les faces traitées de *solide1* et de *solide2*. Nous donnons maintenant toutes les propriétés topologiques que nous allons utiliser pour résoudre les quatre cas.

- Propriété (a) (figure 36) : si le point d'intersection P appartient à une arête de *solide1* alors il doit se trouver dans la seconde face adjacente à cette arête.

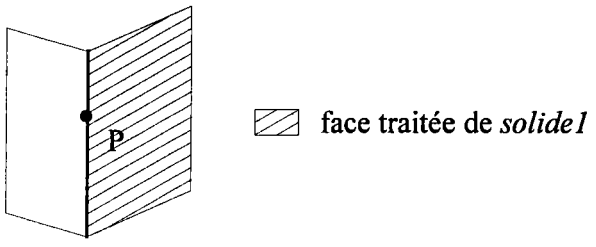


Figure 36 : Propriété (a).

- Propriété (b) (figure 37) : si le point d'intersection P est un sommet de *solide1* alors il doit se trouver dans toutes les autres faces adjacentes à ce sommet.

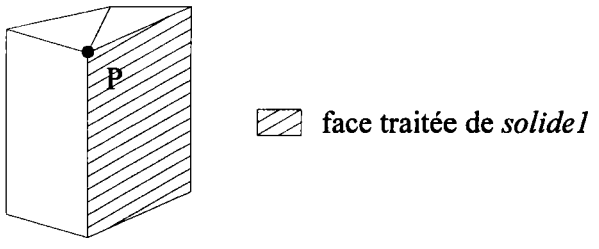


Figure 37 : Propriété (b)

- Propriété (c) (figure 38) : si le point d'intersection appartient à un segment de la section et si ce point n'est aucune des extrémités du segment, alors il doit appartenir à la face qui a donné la droite supportant ce segment. Ce point d'intersection ne peut être un sommet de *solide2*. Sinon il serait un point de la section. Par contre, il peut appartenir à une arête de *solide2*. Dans ce cas, la recherche de point identique se poursuit avec la propriété (a).

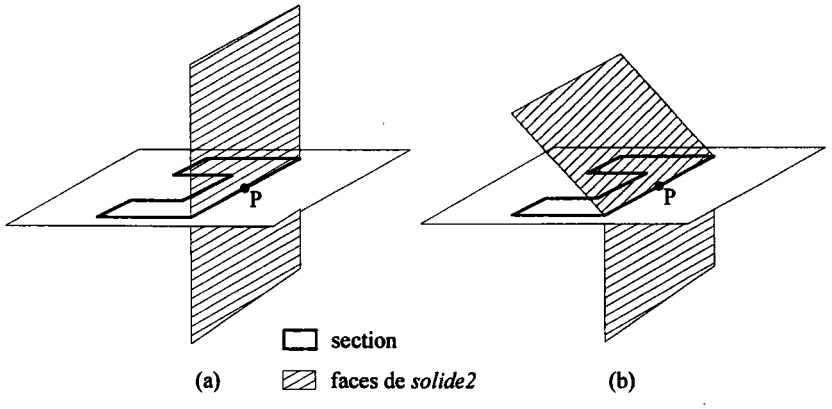


Figure 38 : Propriété (c)

- Propriété (d) (figure 39) : si le point d'intersection est un point de la section, alors il appartient à une arête de *solide2*. De plus, ce point d'intersection peut être un sommet de *solide2*. Dans ce cas, la recherche se poursuit avec les propriétés (a) et (b).

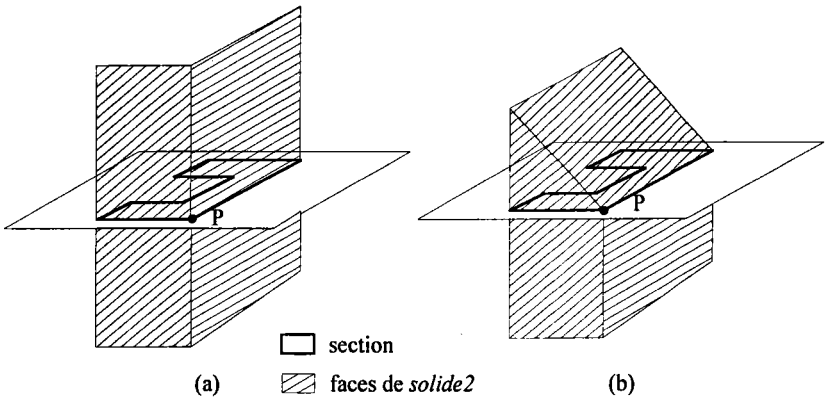


Figure 39 : Propriété (d).

6.3. Démonstrations.

Dans cette section, nous donnons toutes les équations topologiques nécessaires à la recherche des points identiques dans toutes les faces suivant les quatre cas décrits dans la section "Propriétés topologiques". Nous indiquons où se situe le point d'intersection pour les quatre cas dans les autres faces de *solide1* et de *solide2*.

Pour ce faire nous définissons une nouvelle fonction, commutative :
Inter (a_1, ... , a_n) telle que
$a_i \in \mathbb{A}$ ou $a_i \in \mathbb{D}$ avec $n \in \{2, ... ,+\infty\}$
qui détermine l'intersection entre les entités a_i sur une face donnée du traitement.

En utilisant la fonction *Inter*, tous les cas présentés à la section 6.2 de ce chapitre peuvent s'exprimer de la façon suivante :

Cas 1 $\Leftrightarrow$ P = *Inter* (a , d) sur la face *f*. (i)
Cas 2 $\Leftrightarrow$ P = *Inter* (a_1 , a_2 , d) sur la face *f*. (ii)
Cas 3 $\Leftrightarrow$ P = *Inter* (a , d_1 , d_2) sur la face *f*. (iii)
Cas 4 $\Leftrightarrow$ P = *Inter* (a_1 , a_2 , d_1 , d_2) sur la face *f*. (iv)

Comme la recherche de points identiques de ces quatre cas comporte des parties communes, il nous est apparu plus judicieux de transformer ces quatre cas en quatre nouveaux qui prennent en compte les entités de la fonction *Inter*.

Si l'on considère les équivalences (i) à (iv), on s'aperçoit que la fonction *Inter* comprend une ou deux entités qui appartiennent à l'ensemble $\mathcal{A}$ et une ou deux entités qui appartiennent à l'ensemble $\mathcal{B}$. Ce qui donne également quatre cas :

- la fonction *Inter* n'a qu'une seule entité appartenant à l'ensemble $\mathcal{A}$,
- la fonction *Inter* a deux entités appartenant à l'ensemble $\mathcal{A}$,
- la fonction *Inter* n'a qu'une seule entité appartenant à l'ensemble $\mathcal{B}$,
- la fonction *Inter* a deux entités appartenant à l'ensemble $\mathcal{B}$.

Nous allons détailler ces quatre cas car leur combinaison adéquate donne la résolution des cas 1 à cas 4 de la section "Propriétés topologiques".

6.3.1. La fonction Inter *n'a qu'une seule entité appartenant à l'ensemble* $\mathcal{A}$

$P = Inter\ (e_1, \ldots, e_n)$ **sur** f tel que $n \in \{2, 3\}$,

$\exists i / e_i \in \mathcal{A}$ et $\forall j / j \in \{1,\ldots,n\}, j \neq i, e_j \notin \mathcal{A}$.

$\Leftrightarrow$

$P = Inter\ (a, e_1, \ldots, e_n)$ **sur** f tel que $n \in \{1, 2\}$, $a \in \mathcal{A}$, $\forall i \in \{1,\ldots,n\}, e_i \notin \mathcal{A}$. (1)

Dans ce cas, le point *P* appartient à une arête *a* de *solide1*. Donc le point *P* doit se trouver dans l'autre face adjacente à l'arête *a*. Avec l'équation (1) et la propriété (a) on en déduit :

$P = Inter\ (a, e_1, \ldots, e_n)$ **sur** *FaceAdj* (f, a). (2)

6.3.2. La fonction Inter a deux entités appartenant à l'ensemble $\mathcal{A}$

$P = Inter\ (e_1, \ldots, e_n)$ **sur** f tel que $n \in \{3, 4\}$,

$\exists i, j / e_i \in \mathcal{A}, e_j \in \mathcal{A}$ et $\forall k / k \in \{1,\ldots,n\}, k \neq i, k \neq j, e_k \notin \mathcal{A}$.

$\Leftrightarrow$

$P = Inter\ (a_1, a_2, e_1, \ldots, e_n)$ **sur** f tel que $n \in \{1, 2\}$, $a_1 \in \mathcal{A}$, $a_2 \in \mathcal{A}$, $\forall i \in \{1,\ldots,n\}, e_i \notin \mathcal{A}$. (3)

Le point *P* est à l'intersection des arêtes a_1 et a_2. Donc le point P est un sommet du solide *solide1*. Dans ce cas le point *P* doit se trouver dans toutes les autres faces adjacentes au point *P*. Avec l'équation (3) et la propriété (b), on peut déduire :

$P = Inter\ (a_1, ArêteSuiv\ (a_1, FaceAdj\ (f, a_1)), e_1, \ldots, e_n)$ **sur** $FaceAdj\ (f, a_1)$ (4)

et

$P = Inter\ (a_2, ArêtePrec\ (a_2, FaceAdj\ (f, a_2)), e_1, \ldots, e_n)$ **sur** $FaceAdj\ (f, a_2)$ (5)

Si *ArêtePrec* (a_2 , *AdjFace* (f,a_2)) n'est pas égal à *ArêteSuiv* (a_1 , *AdjFace* (f,a_1)) alors il est nécessaire de parcourir toutes les faces adjacentes du point P afin de trouver tous les points identiques.

6.3.3. La fonction Inter *n'a qu'une seule entité appartenant à* $\mathcal{L}$

P = *Inter* (e_1 , ... , e_n) **sur** f tel que $n \in \{2, 3\}$, $\exists i / e_i \in \mathcal{L}$ et $\forall j / j \in \{1,...,n\}, j \neq i, e_j \notin \mathcal{L}$.

$\Leftrightarrow$

P = *Inter* (d , e_1 , ... , e_n) **sur** f tel que $n \in \{1, 2\}$, $d \in \mathcal{L}$, $\forall i \in \{1,...,n\}, e_i \notin \mathcal{L}$. (6)

Le point *P* se trouve sur un segment porté par la droite *d*. Le point *P* se trouve au moins une fois sur l'objet *solide2* sur la face ayant donné la droite *d*. Le point P est à l'intersection entre la droite provenant de la face *f* et les droites provenant des faces adjacentes aux entités e_1, ... , e_n.

Avec l'équation (6) et la propriété (c), on peut déduire :

P = *Inter* (*Droite* (f) , *Droite* (*FaceAdj* (f,e_1)) , ... , *Droite* (*FaceAdj* (f,e_n))) **sur** *Face* (d). (7)

Le point *P* ne peut pas être un sommet de *Face (d)* sinon *P* serait à l'intersection de deux droites. Mais le point *P* peut appartenir à une arête de *Face (d)*. Soit a_d cette arête. Dans ce cas, l'équation (7) est incomplète et est :

P = *Inter* (a_d , *Droite* (f) , *Droite* (*FaceAdj* (f,e_1)) , .. , *Droite* (*FaceAdj* (f,e_n))) **sur** *Face* (d). (8)

A partir de l'équation (8) et de la propriété (a) sur l'arête a_d on peut déduire :

P = *Inter* (a_d , *Droite* (f) , *Droite* (*FaceAdj* (f,e_1)) , ... , *Droite* (*FaceAdj* (f,e_n))) **sur** *FaceAdj* (*Face* (d) , a_d). (9)

6.3.4. La fonction Inter *a deux entités appartenant à l'ensemble* $\mathcal{D}$

P = *Inter* (e_1 , ... , e_n) **sur** f tel que n ∈ {3 , 4} ,

$\exists i, j / e_i \in \mathcal{D}, e_j \in \mathcal{D}$ et $\forall k / k \in \{1,...,n\}, k \neq i, k \neq j, e_k \notin \mathcal{D}$.

⇔

P = *Inter* (d_1 , d_2 , e_1 , ... , e_n) **sur** f tel que n ∈ {1 , 2} , $d_1 \in \mathcal{D}$, $d_2 \in \mathcal{D}$, $\forall i \in \{1,...,n\}, e_i \notin \mathcal{D}$. (10)

Le point *P* appartient à l'arête commune aux faces *Face* (d_1) et *Face* (d_2) car il est à l'intersection des droites d_1 et d_2. Soit a_{12} cette arête.

Avec l'équation (10) et la propriété (d) on peut déduire :

P = *Inter* (*Droite* (f) , a_{12} , *Droite* (*FaceAdj* (f,e_1)) , ... , *Droite* (*FaceAdj* (f,e_n))) **sur** *Face* (d_1) (11)

et

P = *Inter* (*Droite* (f) , a_{12} , *Droite* (*FaceAdj* (f,e_1)) , ... , *Droite* (*FaceAdj* (f,e_n))) **sur** *Face* (d_2). (12)

Si le point *P* n'est aucune des extrémités de l'arête a_{12} alors tous les cas ont été traités. Mais si *P* est l'origine ou l'extrémité de l'arête a_{12} alors l'arête précédente et l'arête suivante de l'arête a_{12} doivent être prises en compte dans l'équation (11) et l'équation (12). Dans ce cas, le point *P* est un sommet de *solide2*.

L'équation (11) est alors incomplète et est :

P = *Inter* (*Droite* (f) , a_{12} , *ArêteSuiv* (a_{12}, *Face* (d_1)) , *Droite* (*FaceAdj* (f,e_1)) , ... , *Droite* (*FaceAdj* (f,e_n))) **sur** *Face* (d_1) (13)

et l'équation (12) devient :

P = *Inter* (*Droite* (f) , a_{12} , *ArêtePrec* (a_{12} , *Face* (d_2)) , *Droite* (*FaceAdj* (f,e_1)) , ... , *Droite* (*FaceAdj* (f,e_n))) **sur** *Face* (d_2). (14)

A partir de l'équation (13) et avec la propriété (a) sur l'arête suivante de a_{12}, on peut déduire :

P = *Inter* (*Droite* (f) , *ArêteSuiv* (a_{12} , *Face* (d_1)) , *ArêtePrec* (*ArêteSuiv* (a_{12} , *Face* (d_1))) , *Droite* (*FaceAdj* (f,e_1)) , ... , *Droite* (*FaceAdj* (f,e_n))) **sur** *FaceAdj* (*Face* (d_1) , *ArêteSuiv* (a_{12} , *Face* (d_1))). (15)

A partir de l'équation (14) et avec la propriété (a) sur l'arête précédente de a_{12}, on peut déduire :

P = *Inter* (*Droite* (f), *ArêtePrec* (a_{12} , *Face* (d_2)) , *ArêteSuiv* (*ArêtePrec* (a_{12} , *Face* (d_2))) , *Droite* (*FaceAdj* (f,e_1)) , ... , *Droite* (*FaceAdj* (f,e_n))) **sur** *FaceAdj* (*Face* (d_2) , *ArêtePrec* (a_{12} , *Face* (d_2))). (16)

Si *FaceAdj* (*Face* (d_1) , *ArêteSuiv* (a_{12} , *Face* (d_1))) n'est pas égal à *FaceAdj* (*Face* (d_2) , *ArêtePrec* (a_{12} , *Face* (d_2))), il est nécessaire de tourner tout autour du point *P* tant que toutes ses faces adjacentes n'ont pas été parcourues afin de retrouver le point P dans toutes les faces.

Les équations (1) à (16) représentent tous les cas topologiques possibles pour un point d'intersection. Si le point de la face n'est pas un point d'intersection, alors il est soit un sommet de la face soit un point de la section. Il n'est pas nécessaire de traiter le point de la section car il est traité par toutes les équations (1) à (16). Par contre, si le point est un sommet de la face, il est nécessaire de tourner tout autour de ce point pour le retrouver sur toutes ses faces adjacentes. L'objet résultat est correctement créé en combinant de manière adéquate tous les cas décrits dans cette section.

7. COMPARAISON AVEC LA MÉTHODE DES FACES.

Cette section est consacrée à la comparaison de la méthode des faces [KIY 92] (voir la section 3.1.6 du chapitre 1) avec la méthode des sections [PER 92] [GAP 93] que nous présentons dans ce chapitre car les auteurs décrivent une méthode similaire à celle que nous avons développée.

Nous ne nous attachons pas ici à souligner les similitudes entre les deux méthodes, nous développons simplement les principales différences que nous avons décelées. Pour ce faire, nous rappelons brièvement l'algorithme de la méthode des faces :

i. Exécuter les actions suivantes pour un solide A donné :

i.1. Etendre une face F_A du solide A à son plan infini S_A, obtenir une section C_A entre le plan S_A et les faces d'un autre solide B.

i.2. Trouver les points d'intersection entre la face F_A et la section C_A et découper les arêtes.

i.3. En suivant les arêtes découpées en *i.2*, obtenir la face d'intersection I_A entre F_A et la section C_A.

i.4. Exécuter les étapes *i.1* à *i.3* pour toutes les faces du solide A et obtenir toutes les faces d'intersection. Unifier ces faces pour générer une composante connexe ouverte Sh_A.

ii. De la même manière, exécuter *i* pour un autre solide B et générer une deuxième composante connexe Sh_B.

iii. Unifier les composantes connexes ouvertes Sh_A et Sh_B pour obtenir une composante connexe fermée Sh_C qui est le solide intersection résultat.

Nous portons notre attention sur la manière dont les auteurs proposent de rechercher la section et sur la façon d'unifier les faces d'intersection afin de former les deux composantes connexes ouvertes Sh_A et Sh_B dans un premier temps, et la composante connexe fermée Sh_C dans un second temps.

La recherche de la section semble plus indiquée que la notre dans la mesure où les auteurs font moins de calcul d'intersections que notre méthode. Mais leur intérêt n'est pas d'avoir un algorithme dont l'extension à d'autres types d'objets soit possible. Effectivement, les auteurs supposent qu'un segment relie deux points de la section. Ils ne prennent donc pas en compte le cas où deux points de la section peuvent être reliés par une courbe. Notre méthode résout ce cas en effectuant un calcul d'instersection de plans qui peut être remplacé par un calcul d'intersection de surfaces dans le cas des objets à surfaces quelconques.

Détaillons les deux algorithmes :

Méthode des sections

Début
{Recherche des points de la section}
EnsDroite:=∅ ;
Pour chaque face F_j de l'objet O_j **Faire**
  Si intersection (F , F_j) = Droite **Alors**
    EnsDroite := EnsDroite∪Droite
  FinSi
  Si intersection = ∅ **Alors**
    face suivante
  FinSi
  Si les plans sont identiques **Alors**
    F et F_j sont regroupées
  FinSi
FinPour
EnsPoint:=∅ ;
Pour toutes les droites D de EnsDroite **Faire**
  Pour toute droite D' de EnsDroite - D **Faire**
  Si D et D' ont une arête commune A **Alors**
    Si (I = Intersection (D,D'))∈ A **Alors**
      EnsPoint := EnsPoint∪ I
    FinSi
  FinSi
  FinPour
  EnsDroite:=EnsDroite-D ;
FinPour
{Formation de la section}
Suivre toutes les droites D de EnsPoint et former la section ;
Fin

Méthode des faces

Début
{Recherche des arêtes de la section}
Etendre la face F à son plan P
EnsArete:=∅ ;
Pour toutes les faces F_j de l'objet Oj **Faire**
  EnsPoint:=∅ ;
  Pour toutes les aretes A_j de F_j **Faire**
    P = Intersection (F,A_j)
      EnsPoint:=EnsPoint ∪ P
  FinPour
  Former les arêtes A_i à partir de EnsPoint et les inclure dans EnsArete
FinPour

{Formation de la section}
Suivre toutes les arêtes de EnsArete afin de former la section
Fin

Dans les deux méthodes, si un point P est un point de la section alors P appartient à une arête A_i de l'objet sur lequel s'applique la recherche de la section. L'arête A_i est partagée par deux faces F_1 et F_2. Soit F la face que l'on traite.

Dans la méthode des sections que nous présentons, le point P est obtenu en faisant deux intersections plan-plan (intersection entre le plan de F et le plan de F_1 et obtention d'une droite D_1 ; intersection entre le plan de F et le plan de F_2 et obtention d'une droite D_2) puis une intersection droite-droite (intersection entre D_1 et D_2).

Dans la méthode des faces, le point P est obtenu par une simple intersection entre l'arête A_i et le plan de F.

La méthode pour obtenir un point de la section est donc beaucoup moins complexe que celle que nous avons développée. Cependant, cette résolution est absolument spécifique aux objets à facettes planes mais ne peut en aucun cas être utilisée pour des objets ayant des supports de faces non plans. Effectivement, dans ce cas, ce n'est plus un segment qui relie deux points de la section, mais une courbe quelconque.

De plus, les auteurs ont détaillé de nombreux cas particuliers que nous avons réussi à généraliser :

- ils distinguent le cas où la section n'est composée que d'un seul contour et celui où elle en comporte plusieurs ;
- ils développent tous les cas particuliers inhérents aux faces coplanaires notamment celui où une face du second objet appartient à la section mais où la section n'est pas égale à cette seule face.

Face à cette abondance de détails qu'il était possible de généraliser, les auteurs ne présentent guère d'explications sur la manière dont ils unifient les parties des faces qu'ils trouvent faisant partie de l'intersection. Ce point nous paraît pourtant bien plus important puisque c'est cette étape qui permet de générer des objets cohérents par rapport au modèle utilisé (voir section 6 de ce chapitre). Nous ne savons donc pas si les auteurs recherchent les points de l'objet final de manière géométrique dans les différentes parties de face trouvées ou si cette recherche s'exécute par rapport à la topologie des deux objets. Nous ignorons également s'ils ont rencontré des difficultés lors de cette unification et quelles sont les solutions qu'ils auraient pu y apporter.

8. IMPLÉMENTATION.

L'algorithme décrit dans ce chapitre a été implémenté sur station SUN en langage C++ sous Unix BSD4.3 SUN OS 4.1.1 [GAP 92]. Il a été réalisé pour des objets Eulériens à facettes planes tels que des boîtes, des cylindres ou des sphères approchés par des faces planes. Le modeleur utilisé est SACADO (Système Adaptatif de Conception et d'Aide au Développement assisté par Ordinateur) développé dans notre laboratoire [GAJ 88].

L'ensemble des algorithmes étant détaillé dans le chapitre 4, nous allons citer les problèmes que nous avons rencontrés et qui sont liés à la réalisation. Les difficultés rencontrées sont dues au fait que les objets mathématiques et informatiques sont différents. Par exemple, comme dans de nombreux autres algorithmes traitant les opérations booléennes, des problèmes demandent à être résolus :

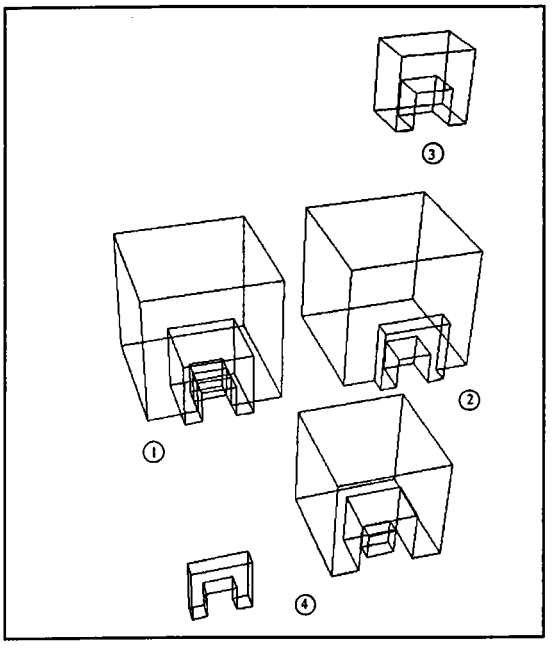
- comment affirmer qu'un point appartient "exactement" à une droite,
- comment affirmer que deux points sont identiques.

Les méthodes mises en œuvre pour résoudre ces problèmes sont classiques et prennent en compte les erreurs provenant des calculs sur ordinateur par la gestion d'un epsilon. Cependant toutes les solutions numériques implémentées restent empiriques car aucune étude ne donne une méthode formelle pour définir et utiliser les epsilons. Nous ne nous sommes pas intéressés à résoudre ce problème car d'une part il constitue à lui seul une étude de thèse et d'autre part car le sujet est en cours de traitement dans notre laboratoire.

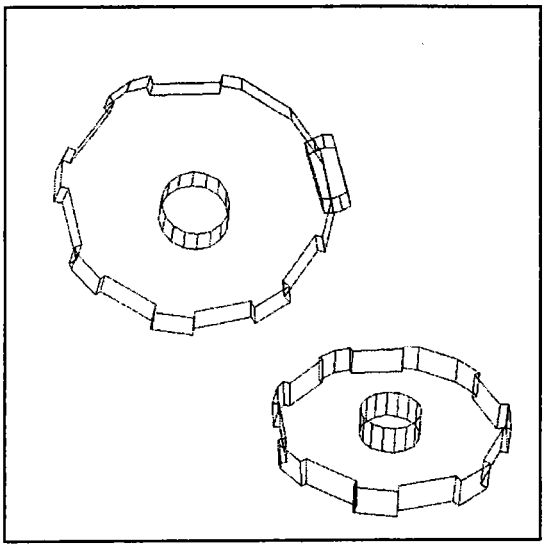
De plus, l'exécution de notre méthode a montré que les temps de calculs étaient trop importants. En effet, nous avons formalisé la recherche de la section (i.e. suivre le contour de l'objet) de telle façon que cette recherche puisse s'adapter à des objets n'ayant pas que des plans en supports de faces. Cependant, le passage de la théorie à la pratique a révélé un temps d'exécution conséquent dès que le nombre de faces devient important. Ce temps d'exécution élevé était dû essentiellement à la recherche de la section.

Nous avons vu à la section précédente, qu'un point de la section est généré par deux intersections plan-plan et une intersection droite-droite. Afin de réduire les temps de calcul pour l'application de la méthode aux objets polyédriques, nous avons modifié ces calculs d'intersection dans le but d'obtenir un algorithme de recherche de section moins général. En effet, il n'est pas nécessaire de calculer les intersections plan-plan puisque deux points de la section sont forcément reliés par une portion de droite puisque ce sont des objets polyédriques. En conséquence, il est suffisant de trouver les points de la section pour obtenir le contour de la section. Pour ce faire, on remplace les deux intersections plan-plan et l'intersection droite-droite par une intersection plan-arête (voir la complexité à la section 2 du chapitre 4).

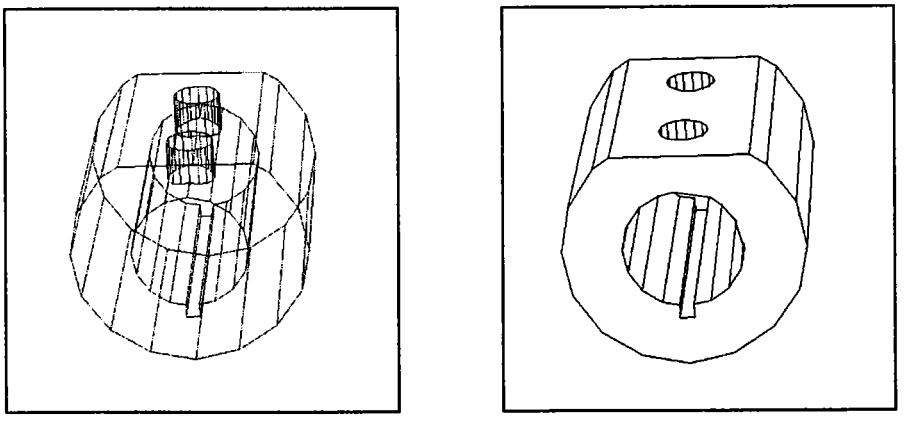
9. EXEMPLES.



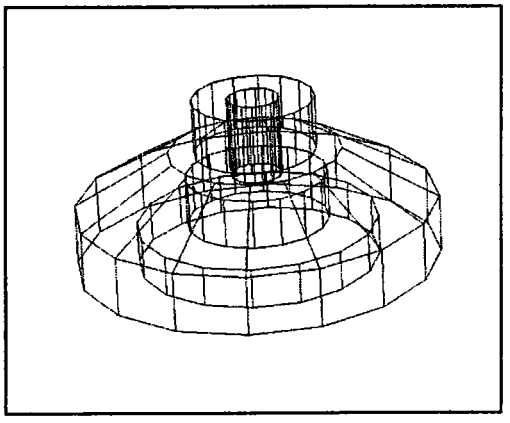
Exemple 1 : Exemple de faces coplanaires.
1. les deux objets positionnés,
2. union,
3. intersection,
4. différence.

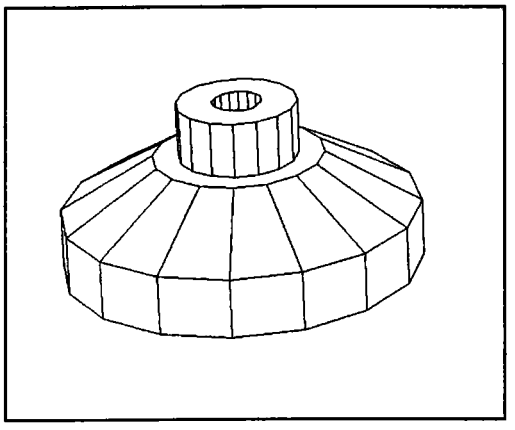


Exemple 2 : Création d'une roue dentée.



Exemple 3 : Création d'un manchon (vue en filaire et vue en parties cachées).





Exemple 4 : Création d'une manivelle (vue en filaire et vue en parties cachées).

10. CONCLUSION.

Notre objectif était de mettre en œuvre une méthode originale pour les algorithmes d'opérations booléennes en tenant compte des contraintes suivantes :

- les objets initiaux sont définis par leurs frontières,
- l'objet final doit être défini par ses frontières,
- la méthode doit pouvoir être étendue à des objets comportant des surfaces gauches et à des objets non-Eulériens.

Ce chapitre a permis de poser les bases théoriques d'une méthode devant s'appliquer à des objets polyédriques mais pouvant s'adapter à d'autres types d'objets.

Le fait d'imposer que les objets initiaux soient définis par les frontières nous a conduits à proposer une méthode traitant les objets face par face. Cette approche a de plus transformé la complexité d'un problème trois dimensions en un problème deux dimensions. L'algorithme traite donc les objets face par face, en cherchant les parties de chaque face qui interviennent dans la composition de l'objet final. Pour ce faire, on applique une opération booléenne en deux dimensions entre la face traitée et la section du second objet qui dépend de celle appliquée sur les objets. Le seul cas particulier de ce traitement est le cas où des faces du premier objet opérande et des faces du second objet opérande sont coplanaires. De plus, le traitement face par face d'un objet permet de prendre en compte les faces isolées qui peuvent exister dans les objets non-Eulériens.

Lorsque l'ensemble des faces faisant partie de l'objet final a été généré, il faut le transformer afin d'obtenir un objet final dont la représentation correspond au modèle de représentation par les frontières. Cette demande nous a amenés à formaliser rigoureusement la reconstruction d'un objet à partir d'un ensemble de faces. Cette formalisation nous a permis d'éviter de trop nombreux cas particuliers et d'obtenir un algorithme robuste.

Nous avons expérimenté cette approche sur des solides à facettes planes. Les résultats obtenus montrent que la solution est viable et que la formalisation que nous en avons faite a réellement débouché sur un algorithme robuste. Les problèmes que nous avons rencontrés sont des problèmes d'implémentation liés aux approximations des réels et aux temps de calculs.

Dans le chapitre suivant, nous proposons une extension de la méthode que nous venons de présenter dans ce chapitre, pour traiter des surfaces non planes. Nous explicitons les différentes façons dont le problème peut être abordé, en conservant la méthode générale proposée pour les facettes planes. Nous détaillons ensuite celle qui a été choisie.

CHAPITRE 3.

OPÉRATIONS BOOLÉENNES SUR DES OBJETS À FACES QUELCONQUES : EXTENSION DE LA MÉTHODE DES SECTIONS.

1. INTRODUCTION.

Dans le chapitre précédent, nous avons proposé une méthode pour traiter les opérations booléennes 3D sur des objets à facettes planes [GAP 93] [GAP 94] [GAP 95]. Les apports fondamentaux de cette méthode sont : la formalisation des différents traitements à effectuer, la simplification consistant à traiter un problème essentiellement 3D en 2D et un traitement basé sur les faces, facilitant la prise en compte des faces isolées de certains objets non Eulériens.

Nous proposons dans ce chapitre une extension de cette méthode pour traiter des surfaces non planes. Nous présentons les différentes façons dont le problème peut être abordé, en conservant la méthode générale proposée pour les facettes planes. Nous détaillons ensuite celle qui a été choisie et nous terminons par des exemples et une critique de notre travail.

Dans l'ensemble de ce chapitre, le terme "facette" désigne toute partie de frontière de l'objet ayant comme support un plan ; le terme "face" indique toute partie de frontière de l'objet ayant comme support une surface quelconque ; le terme "surface" représente la définition mathématique du support d'une face.

2. CHOIX D'UNE SOLUTION.

2.1. Formulation du problème.

L'élément essentiel de l'algorithme que nous proposons pour les polyèdres et que nous voulons étendre aux objets à surfaces quelconques, est de traiter le problème face par face, donc de réduire la complexité de l'algorithme à un problème en deux dimensions. Si un objet est maintenant composé de surfaces quelconques et non plus de faces planes, il faut considérer que la face n'est pas, en général, une entité bi-dimensionnable.

Deux solutions sont possibles :

- trouver un algorithme ayant la même trame que celui pour les objets B-Rep en utilisant la définition mathématique des surfaces,
- adapter les surfaces quelconques pour qu'elles puissent être utilisables par l'algorithme traitant les faces planes.

Dans les deux sections suivantes, nous allons développer les inconvénients et les avantages de ces deux solutions pour décider quelle est la plus adaptée à ce problème.

2.2. Méthode sans modèle intermédiaire.

Dans cette section, nous allons expliciter quels sont les problèmes devant être résolus pour utiliser la définition mathématique des surfaces.

Le choix d'une méthode qui travaille directement sur les surfaces quelconques, implique que cette méthode soit capable de calculer la section d'un objet opérande par le plan d'une face de l'autre objet opérande et d'en déduire un algorithme en deux dimensions.

Pour résoudre ce problème, l'utilisation d'un algorithme d'intersection de surfaces est nécessaire. Le fait d'utiliser un tel algorithme peut contenir de manière cachée le passage par une approximation (en particulier par facettes planes). Cependant, même si cette détermination est faite en utilisant un algorithme par suivi, un certain nombre d'inconvénients peuvent apparaître :

- le temps de calcul de l'intersection peut être important,
- l'exécution n'est pas fiable et peut aboutir à des incohérences ou à des non-résolutions dans les cas de tangence (point singulier),
- le résultat est une suite de points appartenant à la courbe d'intersection. Celle-ci est donc approchée par des segments et l'équation mathématique est perdue, voire impossible à déterminer dans le même modèle que les surfaces elles-mêmes.

Il faut également assurer le passage vers un algorithme en 2D. Ce passage nécessite une projection des objets en cause (section et face). Or cette projection n'est pas possible simplement, sans recouvrement. Par exemple, une sphère pourrait être projetée sur un plan s'appuyant sur un méridien, mais chaque point de projection pourrait correspondre à deux points réels (de chaque hémisphère). Il faudrait découper chaque objet de base en plusieurs sous-objets dont on pourrait assurer une bijection entre les points de la projection et les points de la surface.

2.3. Méthode avec un modèle intermédiaire.

Le modèle intermédiaire le plus naturel pour appliquer la méthode que nous avons développée dans le deuxième chapitre, sans travailler directement sur les surfaces, est un modèle par facettes.

Deux approches sont possibles pour la réalisation d'une approximation des surfaces par des facettes :

- les surfaces de l'objet sont approchées par des facettes ; ce qui revient à considérer l'objet comme s'il était représenté par un modèle B-Rep,
- seules des portions de faces sont concernées par l'approximation qui doit être adaptée à la méthode.

Il est indispensable d'expliciter les critères qui permettent de choisir la méthode qui semble la plus appropriée en tenant compte du passage possible ou non de la théorie à l'implémentation.

Pour ce faire, nous allons essentiellement développer la seconde méthode citée précédemment en précisant quels sont les besoins qu'engendre sa réalisation. En effet, les exigences de réalisation de la première méthode sont incluses dans celles de la seconde. S'il est possible de localiser l'approximation, alors il est forcément possible d'étendre cette approximation à l'ensemble de la surface. De plus, dans ce cas, l'algorithme des faces planes ne nécessite aucune modification pour sa mise en œuvre.

Nous allons donc expliquer pourquoi et comment on peut mettre en œuvre une approximation non systématique et quelles sont ses répercutions sur l'algorithme des faces planes.

L'endroit d'une face qu'il est nécessaire d'approcher par des facettes est principalement le lieu de l'intersection de cette face avec le second objet. En effet, les autres parties de la face sont soit inchangées, soit inutiles dans le résultat de l'application de l'opération booléenne (figure 40). En conséquence, toutes les facettes qui ne sont pas dans le voisinage de la courbe d'intersection sont inchangées ou inutiles dans le résultat final. S'il est possible de trouver le voisinage du lieu de l'intersection sur la face, il est superflu de traiter les facettes qui ne vont pas construire la courbe d'intersection. L'application de l'algorithme sur les facettes se situant au lieu de l'intersection va donner les parties des facettes qui interviennent dans le résultat final. En ne gardant que les arêtes qui séparent deux surfaces différentes, on obtient une approximation de la courbe d'intersection entre ces deux faces.

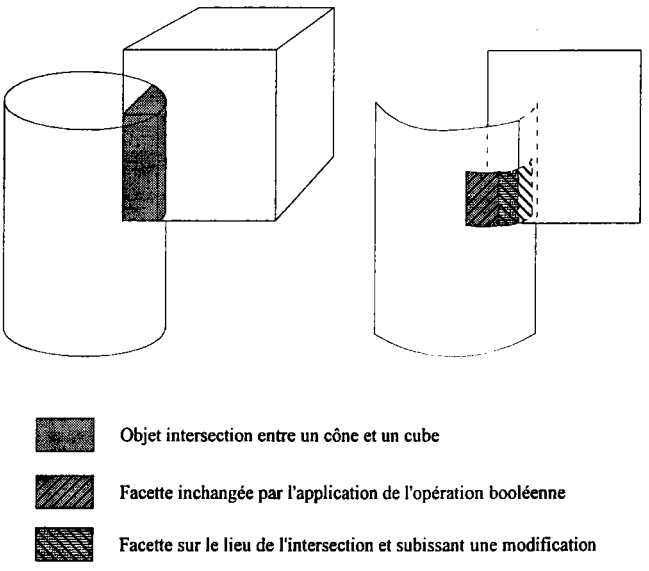


Figure 40 : Facettes utiles et inutiles.

La méthode générale est donc la suivante :

Notons O_1, le premier objet opérande et O_2 le second objet opérande.

Début

Pour toutes les surfaces de O_1 et O_2 **Faire**

Trouver le lieu de l'intersection entre O_1 et O_2

Facettiser le lieu de l'intersection

Fin Pour

Pour toutes facettes de O_1 et O_2 **Faire**

Appliquer l'algorithme des faces planes

Fin Pour

Supprimer les facettes devenues inutiles pour l'objet résultat

Fin

On peut trouver des variantes de cette méthode en travaillant face par face, mais cela ne change pas fondamentalement l'approche.

Pour réaliser cette méthode, il faut :

1) définir un modèle qui puisse supporter une approximation par des facettes,
2) définir une méthode pour trouver le lieu de l'intersection afin de calculer ces facettes,
3) trouver quelles sont les implications d'une approximation non systématique sur l'algorithme des faces planes puisque les objets opérandes peuvent comporter des portions de surfaces non approchées par des facettes,

4) trouver une méthode qui permette la suppression des facettes créées pour l'application de l'algorithme des faces planes afin que l'objet ne comporte plus que des faces et/ou des portions de surfaces.

Nous ne traiterons pas ici le point 1. Nous le développerons plus amplement dans un paragraphe suivant. Disons simplement que le modèle devra permettre la création d'objets composés de faces planes et de faces à surfaces quelconque, et plus particulièrement, l'ajout d'une facette à un objet et assurer également la destruction d'une facette. Ainsi, puisque le modèle supporte l'ajout de facettes, il aurait été possible d'arrêter la méthode à l'application de l'algorithme des faces planes sans considérer une suppression des facettes de l'objet résultat (donc éliminer le point 4).

Cependant cette suppression est triviale et est la même quel que soit le type d'approximation choisie. Le traitement du point 4 ne nous intéresse donc pas actuellement dans le choix de la méthode de calcul de facettes.

Dans la suite de ce paragraphe, nous allons détailler les points 2 et 3. Pour ce faire, montrons en pseudo-code quel serait le schéma global de l'algorithme :

Début {Algorithme général}

O_1, O_2, O_3 : Objet B-Rep exact ;

$Ensface_1$, $Ensface_2$: Ensemble de faces ;

Facettiser ($\underline{O_1}$, $\underline{O_2}$, Englobant (O1) , Englobant (O2)) ;

Traiter (O_1 , O_2 , opérateur , $\underline{Ensface_1}$) ;

Traiter (O_2 , O_1 , opérateur , $\underline{Ensface_2}$) ;

Reconstruction ($Ensface_1$, $Ensface_2$, $\underline{O_3}$) ;

Fin.

Pour trouver le lieu de l'intersection et approcher l'objet par des facettes (ce qui correspond à la procédure *Facettiser* de l'algorithme ci-dessus), il est possible de transformer l'objet considéré en un arbre octal. Rappelons qu'un arbre octal divise l'englobant cubique d'un objet en octants appelés aussi boîtes (voir section 3.2 du chapitre 1 "État de l'art").

Les boîtes qui nous intéressent pour trouver le lieu de l'intersection sont les boîtes grises de l'$objet_1$ qui ont une intersection avec les boîtes grises de l'$objet_2$. On doit ensuite approcher par des facettes les portions de surfaces de l'objet contenues dans ces boîtes. Cette partie est la plus ardue du problème car il ne suffit pas de calculer ces facettes n'importe comment sur la portion de surface de la boîte; l'approximation de la surface contenue dans une boîte doit être fonction de l'approximation des six boites voisines afin de garder une topologie cohérente. Cet aspect de l'approximation a été réalisée lors de la construction d'un maillage [PES 89]. Les auteurs montrent que c'est un travail complexe qui inclut de nombreux cas particuliers.

L'algorithme suivant résume la méthode pour trouver le lieu de l'approximation :

Procédure *Facettiser* ($\underline{O_1}$, $\underline{O_2}$, B_1 , B_2) ;

Début
Comparaison (B_1 , B_2 , Etat , $\underline{Facette_1}$, $\underline{Facette_2}$)
Cas suivant Etat
Arrêt: {Rien à faire}
Poly :**Début**
Ajouter ($Facette_1$, $\underline{O_1}$) ;
Ajouter ($Facette_2$, $\underline{O_2}$) ;
Fin
Div : **Début**
Division (B_1, $\underline{ListeBoite_1}$)
Division (B_2, $\underline{ListeBoite_2}$)
Pour toutes les boîtes BO1 de $ListeBoite_1$ **Faire**
Pour toutes les boîtes BO2 de $ListeBoite_2$ **Faire**
Facettiser (O1 , O2 , BO1 , BO2)
Fin Pour
Fin Pour
Fin
Fin Cas
Fin

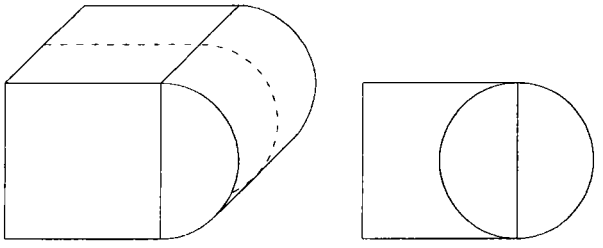
Procédure *Comparaison* (B_1 , B_2 , O_1 , O_2 , Etat , $\underline{Facette_1}$, $\underline{Facette_2}$)

Début
Si (Intersection (B_1, B_2) **et** Frontière (B_1 , Equation (O_1))
et Frontière (B_2 , Equation (O_2))
) **Alors**
Si Niveau (B_1) = Dernier **Alors**
{à polygonaliser}
Etat ← Poly
Sinon {on continue à diviser}
Etat ← Div
Fin Si
Sinon {on arrête}
Etat ← Arrêt
Fin Si
Fin

Fonction *Intersection* (B_1 , B_2)
{Renvoie VRAI si les boîtes B1 et B2 ont une intersection}

Fonction *Frontière* (B , Eq)
{Renvoie VRAI si la boite B contient une partie de la surface représentée par Eq}

Considérons maintenant la procédure *Traiter* citée plus haut. On rappelle que l'on est en train de traiter une facette d'un objet et que l'on cherche à trouver la section du second objet par rapport au plan de cette facette. Pour ce faire, on calcule l'intersection du plan avec toutes les facettes du second objet. Si l'on se borne à ces intersections, on n'obtiendra jamais (ou exceptionnellement) des segments constituant une ligne brisée fermée formant la section puisque l'on n'a approché par des facettes que le lieu de l'intersection. Pour fermer la ligne brisée il faut donc, soit utiliser un moyen quelconque permettant de relier les segments, soit intégrer à ces segments les intersections plan-surface. Ce qui soulève le problème suivant (voir figure 41) : l'intersection plan-surface va nous donner l'ensemble de l'intersection plan-surface même si l'on n'a besoin que d'une portion de cette intersection.



----- Plan de coupe

Figure 41 : La section de l'objet par rapport au plan de coupe va donner un cercle entier qui correspond à l'intersection entre le cylindre et le plan de coupe. Cependant une partie du cercle appartient à la section.

L'algorithme de traitement des faces et facettes d'un objet partiellement approché par des facettes est le suivant :

Procédure *Traiter* (O1 , O2 , Opérateur , EnsFace)

Début
    EnsFace ← ∅ ;
    Pour toutes les facettes F de l'objet O_1 **Faire**
        Intersection (*Plan* (F) , O2 , EnsDroite , EnsPoint) ;
        ConstruireSection (EnsDroite ,EnsPoint , section) ;
        Bool2D (F , section , Face) ;
        EnsFace ← EnsFace ∪ {Face} ;
    Fin Pour
Fin

Procédure *Intersection* (P , O , EnsDroite , EnsPoint)

Début
EnsDroite ← ∅ ;
EnsPoint ← ∅ ;
Pour toutes les faces F de l'objet O **Faire**
Si *plane* (F) **Alors**
InterPlanPlan (P , Plan (F) , Droite) ;
EnsDroite ← EnsDroite ∪ {Droite} ;
Sinon InterPlanGauche (P , Gauche (F) , EnsPoint) ;
Fin Si
Fin Pour
Fin

2.4. Conclusion.

Le temps de calcul important nécessaire à la réalisation de l'intersection exacte de deux surfaces, l'approximation sous-jacente à un calcul d'intersection direct de surfaces et les cas d'avortement des différentes méthodes existantes nous incitent à choisir une approximation des surfaces par des facettes pour traiter les opérations booléennes par la méthode des sections sur des objets quelconques.

L'approximation pouvait être systématique (toutes les surfaces sont approchées par des facettes) ou bien plus "intelligente" (prenant en compte les données du problème). Cependant, les avantages que peut apporter une approximation "intelligente" sont remis en cause par la complexité de programmation des problèmes qu'elle engendre (localisation de l'approximation avec une cohérence topologique et modification de l'algorithme des opérations booléennes sur des faces planes). De plus, le traitement des facettes inutiles est trivial (élimination par les englobants) et se révèle sans doute moins gourmand en temps de calcul que la mise en œuvre d'une approximation "intelligente". En outre, le fait que toutes les surfaces d'un objet opérande d'une opération booléenne doivent être approchées par des facettes, (l'utilisation d'un filtre correspondant à un test d'intersection d'englobants parallélépipédiques étant impossible pour éliminer les surfaces n'ayant pas d'intersection avec le second objet) est compensé par les problèmes qui se poseraient pour la recherche de section (identiques à ceux posés par la recherche d'une section avec une approximation partielle et localisée à l'intersection des deux objets (voir section 2.3 de ce chapitre)) si toutes les surfaces n'étaient pas soumises à une approximation. C'est pourquoi l'approximation systématique par des facettes a été choisie.

Cette approximation demande néanmoins :

- la réalisation d'un modèle capable de supporter cette modification de l'objet et sa visualisation,
- l'étude sur la façon de calculer les facettes sur la surface à partir d'un contour donné,
- la suppression des facettes devenues inutiles lors de la reconstruction de l'objet résultat après application de l'opération booléenne.

Dans la section suivante, nous allons présenter une méthode résolvant ces différents points.

3. MÉTHODE PROPOSÉE.

Dans la section précédente, nous avons montré que la méthode la plus simple pour effectuer des opérations booléennes sur des objets à surfaces quelconques est une approximation systématique par des facettes des deux objets opérandes afin d'utiliser, sans changement, la méthode des opérations booléennes sur des objets à facettes [GAP 95b].

La méthode doit donc passer par les étapes suivantes :
- approximation complète des deux objets;
- opérations booléennes sur les facettes planes;
- suppression des facettes pour retrouver les surfaces réelles.

L'algorithme sur les facettes planes est décrit dans le chapitre 2 "Opérations booléennes sur des polyèdres : la méthode des sections". Pour décrire complètement la méthode, il faut disposer d'un modèle supportant les faces non planes et interprétant les résultats de la méthode. Il est également nécessaire de proposer des algorithmes pour calculer et supprimer les facettes des faces. Nous allons détailler l'ensemble de ces points dans les sections suivantes.

3.1. Le modèle.

3.1.1. Représentation.

Afin de gérer les opérations booléennes sur des objets quelconques, il est indispensable de disposer d'un B-Rep amélioré et tenant compte des surfaces quelconques d'un objet. Rappelons que pour exécuter des opérations booléennes sur ce type d'objets, nous avons décidé d'approcher les surfaces par des facettes. Le modèle doit être capable de représenter des objets comportant des surfaces quelconques ainsi que supporter une approximation de ces surfaces. Il assure l'ajout d'une facette sur une surface donnée. Comme nous ne nous intéressons qu'à la peau de l'objet, l'idée est de garder la représentation d'un B-Rep, c'est-à-dire un graphe FAS (Face Arête Sommet), mais en l'adaptant pour qu'il garantisse les services nécessaires à la gestion et à l'approximation de surfaces d'un objet par des facettes.

Le modèle que nous présentons s'appuie donc sur un graphe FAS avec des changements par rapport à sa définition classique :

- le premier changement est la généralisation du support de la face. Le support de la face peut être, soit un plan (dans ce cas la face est appelée facette), soit une surface quelconque. Une facette est identique à une face d'un objet B-Rep classique. En revanche, pour une face (le support est alors une surface quelconque), plusieurs contours sur la surface sont possibles (notion de face maximale) ainsi le support d'une face n'est représenté qu'une seule fois. Nous appelons contour l'ensemble formé d'une boucle extérieure et de zéro, un ou plusieurs boucles intérieures. Chaque boucle est fermée et bornée par des arêtes. Pour chaque face ou facette, une normale sortante indique l'extérieur de l'objet,
- le second changement concerne l'approximation d'une face. Un ensemble de facettes peut provenir de l'approximation de la surface de la face. Ces facettes ne font donc pas partie intégrante de la composition de l'objet et sont éliminées au cours de la phase de reconstruction de l'objet dans le même modèle que les objets opérandes.

Cependant, malgré ces changements, une arête reste rectiligne même si elle représente une partie d'intersection entre deux surfaces. Une arête sépare donc soit deux facettes, soit deux faces, soit une face et une facette. Une arête est parcourue deux fois et deux fois uniquement mais n'est représentée qu'une seule fois dans le modèle. Enfin, une arête est constituée de deux sommets.

L'implication que toute arête est rectiligne, même si elle représente une portion de courbe, est que lorsque le résultat de l'opération booléenne est topologiquement correct, il est cependant géométriquement faux (c'est le cas de tous les modeleurs aujourd'hui).

L'implémentation du modèle a été faite en C++ (voir section 3.1 du chapitre 4). Le modèle que nous avons réalisé pour notre maquette ne contient que deux quadriques (sphère et cylindre) mais il permet une extension aux autres quadriques et aux surfaces quelconques.

3.1.2. Classification d'un point.

L'une des qualités essentielles demandées à un modeleur est de pouvoir répondre à la question : étant donné un point P de l'espace, peut-on dire s'il est sur la surface d'un objet O, extérieur ou intérieur à O ? La réponse à cette question est indispensable pour la mise en œuvre de nombreux algorithmes.

L'algorithme le plus classique sur des modèles B-Rep est celui de la demi-droite. On "lance" une demi-droite partant du point P et d'une direction choisie pour simplifier les calculs (par exemple parallèle à l'axe des x). Si le nombre d'intersections entre la droite lancée et les faces de l'objet est pair alors le point est extérieur. Si le nombre est impair alors le point est intérieur. Cet algorithme nécessite le traitement d'un certain nombre de cas particuliers. Un simple test d'appartenance aux faces (demi-droite sur un polygone ou angles capables) permet de dire si le point se situe sur la frontière de l'objet. Cet algorithme ne concerne que les objets Eulériens.

Une autre méthode consiste à trouver la face la plus proche du point que l'on traite (plus proche pouvant signifier zéro) et dire suivant la normale sortante à la face si le point est extérieur, intérieur ou sur la frontière.

Nous avons combiné ces deux algorithmes pour résoudre le problème de classification d'un point avec notre modèle. S'il est aisé de connaître le point le plus proche d'une face par rapport à un point P (c'est le projeté orthogonal de P sur la face), ceci l'est moins lorsqu'il s'agit d'une surface quelconque (non unicité du point le plus proche). Par contre nous pouvons dire quel est le point le plus proche d'une surface suivant une direction donnée D puisque le point cherché sera à l'intersection de la droite passant par P et de direction D avec la surface.

D'où l'algorithme :

```
Début
    Prendre une droite D passant par P et d'une direction donnée.
    PlusProche = ∞
    Tant qu'    il existe une surface S de O et que P n'est pas SurO Faire
                Si      P ∈ S Alors
                        P est SurO
                Sinon   I = IntersectionDroiteSurface (D , S)
                        Si      (I < PlusProche) Alors
                                PlusProche = I
                        Fin Si
                Fin Si
    Fin Tant Que
    Si      P n'est pas SurO Alors
            Si      PlusProche est dans le même sens que la normale sortante en S Alors
                    P est ExterieurO
            Sinon   P est InterieurO
            Fin Si
     Fin Si
Fin
```

```
Fonction IntersectionDroiteSurface (D , S)

Début
    Si      Plane (S) Alors
            I = IntersectionDroiteFace (D , S)
    Sinon   Trouver toutes les intersections avec D
            Les ordonner sur D
            trouve ← Faux
            Tant    qu'     il existe une intersection I sur D
                    et
            que     non (trouve) Faire
                    Si      Intérieur (I , S) Alors
                            trouve ← Vrai
                            Stocker la normale à S en I dans I
                    Sinon   Passer à l'intersection suivante
                    Fin Si
            Fin Tant Que
    Fin Si
Fin

Fonction Intérieur (I , S)

Début
    Si      S n'a pas de contour défini Alors
            I est intérieur
    Sinon   Prendre l'arête la plus proche de I
            Tester le point I par rapport à l'arête trouvée
            (Matière à gauche)
    Fin Si
Fin
```

Pour savoir quel est le point le plus proche d'une surface d'un objet O par rapport au point P dont on veut connaître la classification par rapport à O, il faut, dans un premier temps, calculer toutes les intersections entre cette surface et la droite passant par P et d'une direction donnée. Ensuite, pour chaque point d'intersection trouvé et ordonné suivant la direction de la droite, il faut tester s'il appartient à la portion de surface intervenant dans la composition de l'objet.

Pour savoir si un point de la surface appartient réellement à la portion intégrée à l'objet, il faut trouver l'arête la plus proche de ce point sur la surface. L'orientation de l'arête permet de dire si le point est extérieur ou intérieur (voir figure 42).

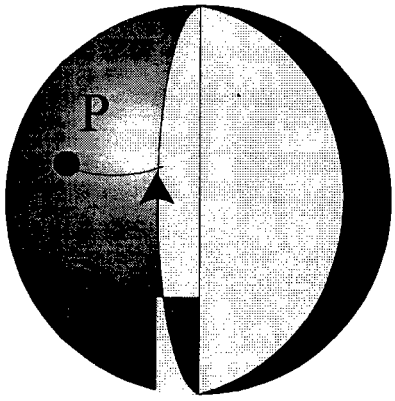


Figure 42 : L'orientation de l'arête (matière à gauche) permet de savoir que le point P appartient à la portion de sphère.

La normale à la surface en l'intersection la plus proche du point P permet de savoir si le point est extérieur ou intérieur (voir figure 43).

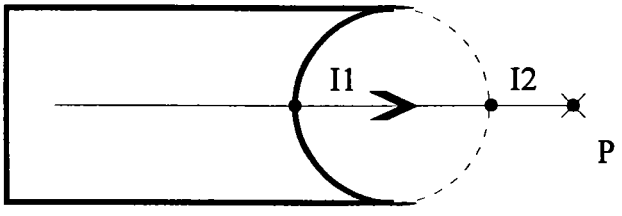


Figure a

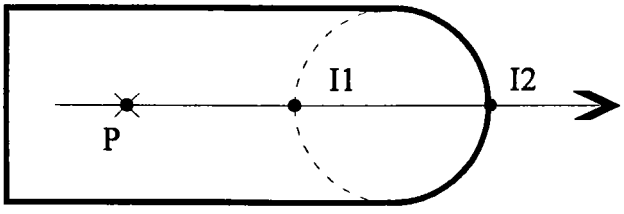


Figure b

Normale en un point de la surface

Coupe de l'objet

Portion de surface n'intervenant pas dans la composition de l'objet

Figure 43 : La figure *a* représente la coupe d'une différence entre une boîte et un cylindre. Les points I_1 et I_2 sont les points d'intersection entre la droite lancée et le cylindre. Le point I_2 plus proche de P, n'appartient pas au solide. C'est le point I_1 qui permet de déterminer que le point est extérieur. La figure *b* représente l'union d'une boîte et d'un cylindre. Le point I_1 plus proche de P, n'appartient pas au solide. C'est le point I_2 qui permet de déterminer que le point est intérieur.

3.1.3. Conclusion.

Le modèle que nous avons créé présente donc les avantages suivants :

- il permet d'effectuer les opérations booléennes sans calculer explicitement une intersection comprenant une surface,
- il permet de déterminer si un point est extérieur, intérieur ou sur l'enveloppe d'un objet.

Cependant il a l'inconvénient suivant :

lorsque l'objet résultat donne un résultat topologiquement correct, le modèle est néanmoins géométriquement faux. Les nouveaux points créés par l'application de l'opération booléenne n'appartiennent pas exactement à la courbe d'intersection des deux objets.

3.2. Approximation d'une surface par des facettes et suppression des facettes.

3.2.1. Approximation.

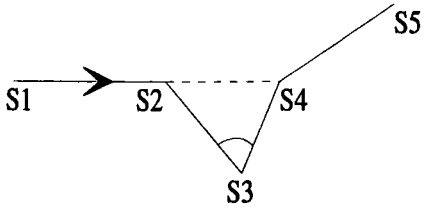
Nous avons expliqué pourquoi nous avons choisi une approximation de l'ensemble de la (ou des) surface(s) traitée(s) (voir section 2). Nous allons maintenant expliciter l'approximation d'une surface par des facettes.

La réalisation de cette approximation que nous proposons est basée sur une triangulation plane qui correspond à celle du mailleur TRIGEO [GEO]. Cette dernière concerne des domaines de contour a priori quelconques. Cependant la partie de contour traitée doit garder une géométrie raisonnable. Si tel n'était pas le cas, l'adjonction d'entités supplémentaires permet de conserver une bonne géométrie (Exemple triangulation d'une face trouée : on ajoute un lien entre le contour extérieur et le trou ce qui permet de se ramener à un polygone simple).

Le contour est défini par une suite de points quelconques. L'algorithme permet de générer des éléments internes (ici des triangles) qui réduisent la complexité du contour. Le contour est en fait assimilé à un polygone. A partir des côtés du polygone, des triangles sont générés et les nouveaux côtés du triangle constituent un nouveau polygone interne.

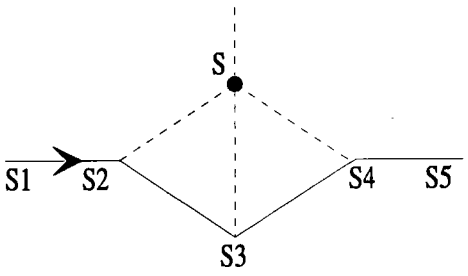
A chaque stade de l'évolution du polygone, on recherche le plus petit angle déterminé par deux côtés consécutifs du polygone. Le triangle n'est généré que si le (ou les) nouveau(x) segment(s) qui le compose(nt) ne coupe(nt) pas le contour du polygone, ce qui permet de traiter les cas de polygones non convexes. Les triangles sont formés suivant la valeur du plus petit angle. Notons α ce plus petit angle et indiquons quels sont les trois cas associés à la valeur de l'angle :

- $0<\alpha<\pi/2$



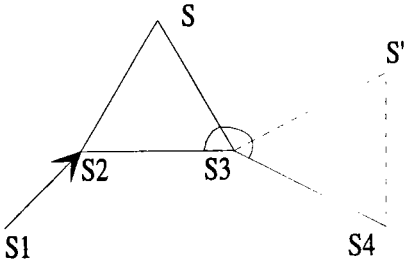
Le triangle (S2 , S3 , S4) est généré. Les arêtes (S2 , S3) et (S3 , S4) sont supprimées du contour. L'arête (S2 , S4) est ajoutée.

• $\pi/2 < \alpha < 2\pi/3$



Le sommet S est généré sur la bissectrice de l'angle α à une distance calculée à partir de la longueur des arêtes (S2 , S3) et (S3 , S4). Nous avons pris la moyenne des deux longueurs pour simplifier. Les triangles (S2 , S3 , S) et (S , S3 , S4) sont générés. Les arêtes (S2 , S3) et (S3 , S4) sont supprimées du contour. Par contre, les arêtes (S2 , S) et (S , S4) y sont ajoutées.

• $\alpha > 2\pi/3$



Le point S est généré tel que le triangle (S2 , S3 , S) soit équilatéral si la longueur de l'arête (S2 , S3) est inférieure à la longueur de l'arête (S3 , S4). Sinon le point S' est généré sous les mêmes critères.

Ces trois cas sont répétés jusqu'à fermeture de la surface à trianguler.

Cette triangulation est intéressante puisqu'elle permet de construire des triangles à partir d'un contour donné. Pour transformer cette triangulation plane en une approximation de surface, il faut être capable d'élever le point sur la surface. Dans la maquette, nous n'avons réalisé que l'approximation d'une sphère et d'un cylindre. Le problème des sphères et des cylindres a été facilement résolu puisque ce sont des surfaces convexes. Les points S sont calculés dans le plan défini par les trois points successifs des deux arêtes du contour. Comme les sphères et les cylindres sont des surfaces convexes, les points S se situent en dessous de la surface. Ils sont donc ajustés en les élevant de telle façon qu'ils appartiennent à la surface. Pour la sphère, il suffit de trouver le point d'intersection sur la sphère appartenant à la droite passant par le centre de la sphère et le point S. Pour le cylindre, on prend le point sur le cylindre appartenant à la droite passant par le point de l'axe du cylindre à même hauteur que le point S et le point S.

Dans le cas où une des deux quadriques de la maquette n'a pas de contour défini, comme ce sont des surfaces fermées définissant un volume, ceci indique que l'objet est la quadrique. Dans ce cas, une polygonalisation classique de la sphère ou du cylindre est mise en œuvre.

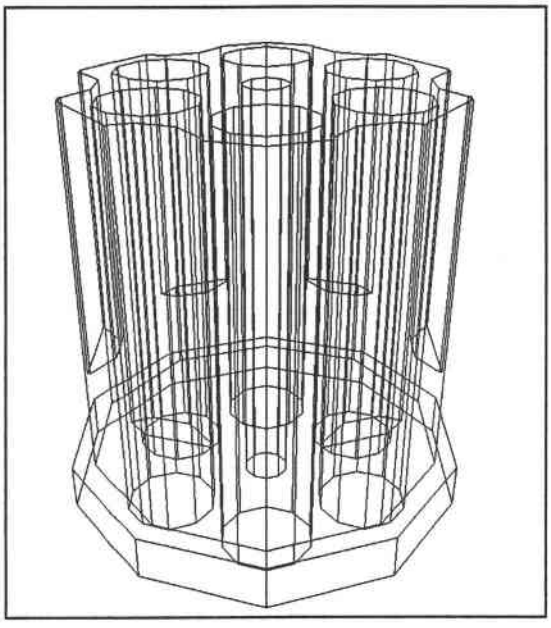
Cependant, les phases de création et de suppression des facettes à chaque opération booléenne peuvent être réduites à une seule opération de création et une seule opération de suppression lorsque l'algorithme des opérations booléennes est utilisé pour évaluer un arbre de construction CSG. Dans ce cas, il suffit de polygonaliser les primitives quadriques de l'arbre et d'appliquer les opérations booléennes successives sur ces objets sans supprimer les facettes à chaque opération. La phase de suppression s'effectue alors à la racine de l'arbre de construction.

3.2.2. Suppression des facettes.

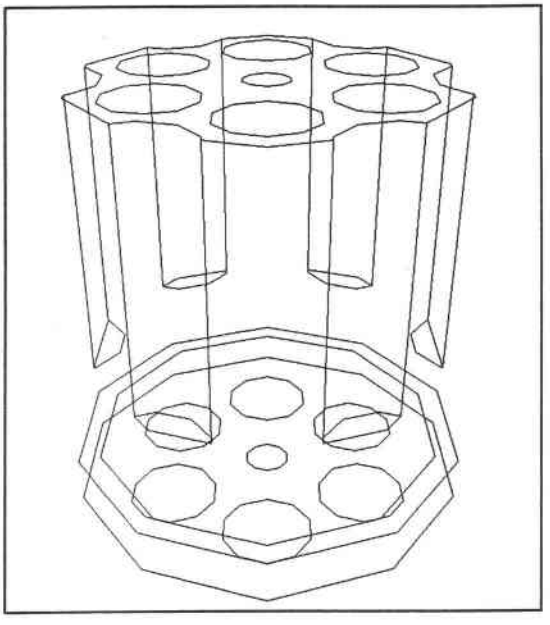
L'implémentation de la suppression des facettes est triviale : soient deux objets sur lesquels une opération booléenne est appliquée. Les deux objets sont approchés par des facettes et l'objet résultat est obtenu en exécutant l'algorithme des opérations booléennes sur des faces planes. L'objet résultat contient donc des facettes qu'il faut supprimer afin d'obtenir l'objet résultat dans le même modèle que les deux objets opérandes. Il suffit pour chaque facette d'une surface donnée de l'objet résultat de ne garder que les arêtes qui ne partagent pas la même surface. L'ensemble de ces arêtes forme un ou plusieurs contours sur la surface de l'objet (idem recherche de section pour les faces planes) et définit la portion de surface à garder dans la composition de l'objet.

3.3. Exemples.

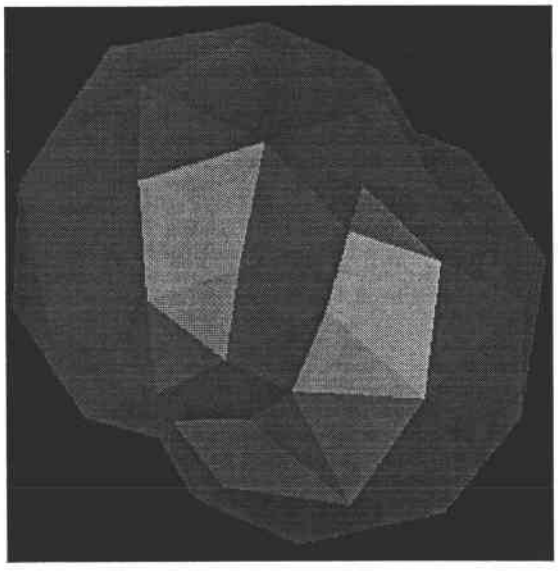
La méthode de création et de suppression de facettes que nous venons de décrire a été implémentée en C++ version 2.0 sur des stations SUN sous le système SUN OS 4.1.1 unix BSD4.3. L'ensemble des algorithmes nécessaires à ces traitements est décrit dans le chapitre 4. La maquette ne comporte comme surfaces que des sphères et des cylindres. Nous donnons ci-après quelques résultats.



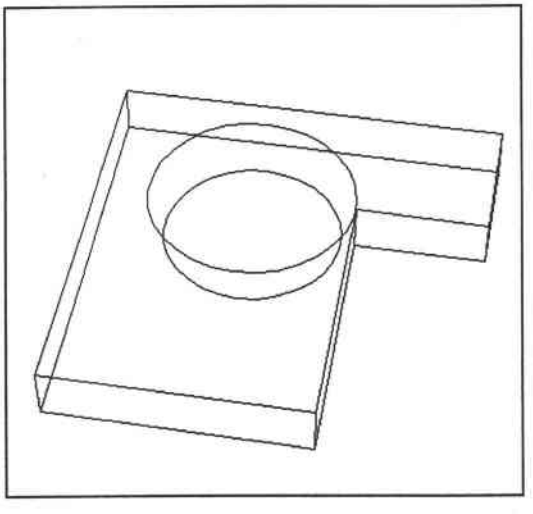
Exemple 5 : Création d'un barillet
Opérations booléennes successives entre plusieurs cylindres sans suppression de l'approximation entre les opérations booléennes.



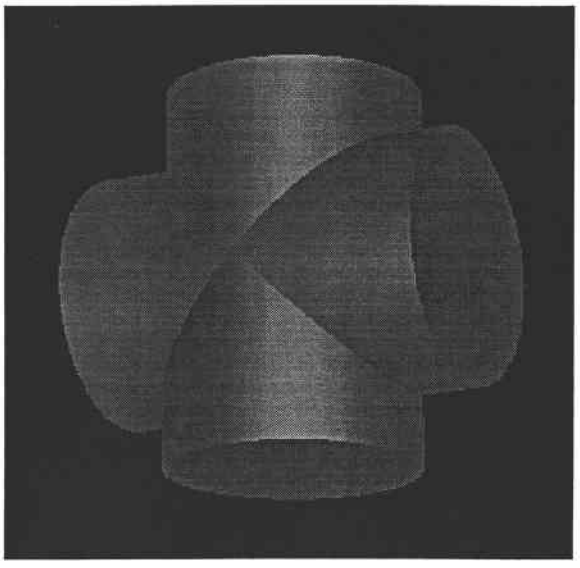
Exemple 6 : Suppression des facettes des surfaces du barillet.



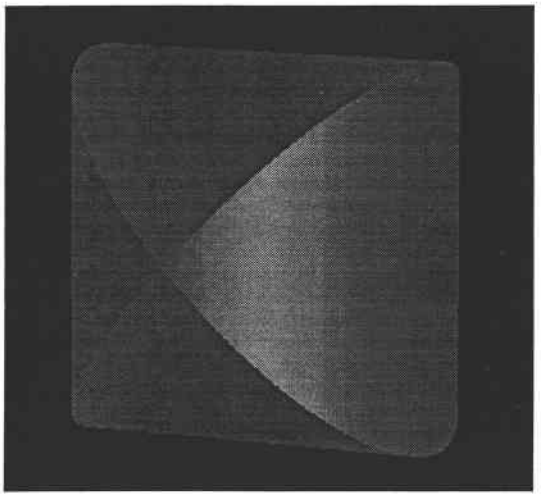
Exemple 7 : Approximation de l'union de deux sphères.



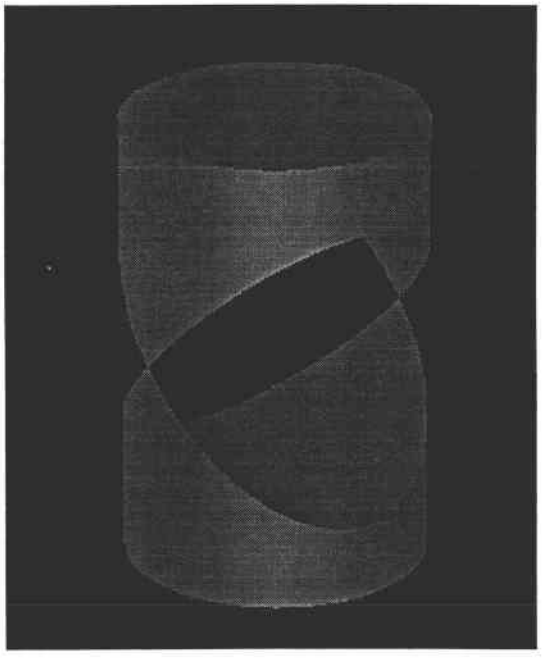
Exemple 8 : Union de deux parallélépipèdes et d'une sphère. La sphère est tangente à l'extrémité de l'arête.



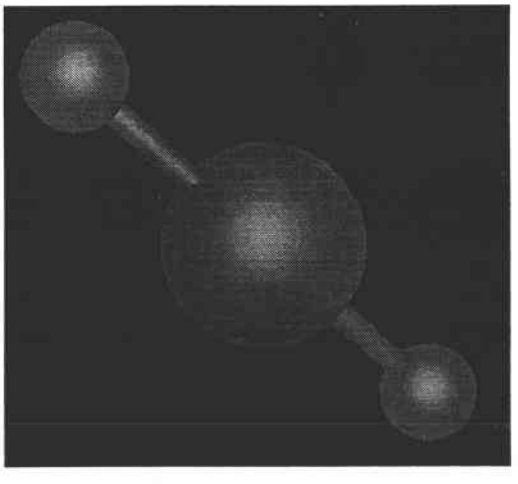
Exemple 9 : Union de deux cylindres perpendiculaires et de même rayon.



Exemple 10 : Intersection de deux cylindres perpendiculaires et de même rayon.



Exemple 11 : Différence de deux cylindres perpendiculaires et de même rayon.



Exemple 12 : Opérations booléennes entre sphères et cylindres.

3.4. Incohérences géométriques.

Nous avons montré qu'il est possible de déterminer si un point de l'espace appartient ou non à un objet représenté par notre modèle. Avec cette méthode, si l'on veut trouver la classification d'un point P par rapport à un objet O, il faut chercher l'intersection la plus proche du point P sur l'objet O suivant une direction donnée. La classification du point P se fait alors avec la normale sortante au point d'intersection trouvé.

Le lancer de rayons est l'un des algorithmes les plus utilisés pour la visualisation d'un objet. Ses traitements principaux sont la recherche de l'intersection la plus proche entre un rayon (ayant pour origine l'oeil et passant par un pixel de l'écran) et un objet de la scène en connaissant la normale en ce point d'intersection. Le lancer de rayons, de par ses propriétés, est, en particulier, approprié pour valider notre méthode de classification d'un point par rapport à un objet puisqu'il demande exactement les mêmes traitements.

Comme on peut s'y attendre (voir la conclusion de la section 3.1), puisque le modèle est géométriquement faux, les résultats montrent qu'il existe des erreurs de visualisation dans le voisinage proche de l'arête. Il existe des raisons à cela :

- le produit vectoriel effectué pour détecter si le point est extérieur ou intérieur suivant l'orientation de l'arête n'est plus fiable dès que le point se rapproche très près de l'arête; le vecteur devient trop petit (norme du vecteur quasiment nulle) pour être caractéristique dans le produit vectoriel. Nous avons résolu le problème en considérant le point intérieur dès qu'il se trouve dans le voisinage de l'arête,
- des points dans le voisinage de l'arête sont omis dès que les extrémités de l'arête s'éloignent de trop de la courbe d'intersection entre les surfaces. Plus les facettes approchent grossièrement l'objet, plus les extrémités des arêtes s'éloignent de la courbe d'intersection. Effectivement seuls les points appartenant au contour de la facette appartiennent exactement à la surface qu'elle représente (puisque la création des facettes est ainsi traitée).

  Supposons qu'une extrémité de l'arête soit un point d'intersection entre une facette d'une surface S_1 avec un segment de la section d'un objet approché. Le segment de la section participant à l'intersection provient d'une facette d'une surface du second objet et le point d'intersection se situe au milieu de la facette de la surface S_1. Moins les facettes seront nombreuses, plus l'écart avec la courbe d'intersection exacte sera grand. Pour pallier ce désagrément, nous proposons d'une part un nombre important de facettes (mais cela augmente le temps de traitement des opérations booléennes), d'autre part de lisser les points afin de les ajuster sur la courbe d'intersection. Nous réduisons ainsi les erreurs de visualisation au voisinage des arêtes concernées (voir notamment les exemples de la section 3.4 du chapitre 4).

4. CONCLUSION.

Afin de pouvoir implémenter les opérations booléennes sur des surfaces gauches, notre choix est d'étendre la méthode proposée pour les polyèdres. La caractéristique principale de la méthode des polyèdres est de traiter les objets face par face. Pour garder cet avantage, deux solutions étaient envisageables :

- utiliser la définition mathématique des surfaces en cherchant un algorithme ayant la même trame que pour les objets polyédriques,
- approcher les surfaces de l'objet par des facettes afin qu'elles puissent être utilisées par la méthode des sections.

Les principaux problèmes qui nous ont décidé à approcher les surfaces non planes de l'objet sont :

- les temps de calcul des méthodes d'intersection pouvant être importants,
- l'approximation sous-jacente à un calcul d'intersection de surfaces,
- les cas d'avortement des méthodes d'intersection de surfaces.

Le type d'approximation pouvait être systématique en approchant toutes les surfaces d'un objet par des facettes ou plus intelligent en prenant en compte les données du problème (seul le lieu d'intersection des surfaces des deux objets nécessite obligatoirement une approximation). Cependant, le gain en temps de calcul du traitement des facettes localisées au lieu de l'intersection se révèle sans doute moins important que le coût du traitement nécessaire à une localisation de l'approximation tout en conservant une cohérence topologique de l'objet. De plus, les facettes inutiles d'une approximation systématique sont rapidement éliminées par les englobants lors du traitement de l'opération booléenne. C'est pourquoi l'approximation porte sur toutes les surfaces des deux objets opérandes de l'opération booléenne.

La réalisation des opérations booléennes sur des objets à surfaces quelconques et la mise en œuvre d'une approximation de toute surface par des facettes demande la définition d'un modèle pouvant représenter les objets composés de surfaces non planes approchées par des facettes. Le modèle que nous avons implémenté s'appuie sur un modèle de type B-Rep. Avec ce type de modèle, un problème courant en C.A.O. qui est l'appartenance d'un point quelconque à un objet a été résolu. Grâce à un algorithme de visualisation, nous avons montré quelles étaient les limites de ce modèle et enfin nous avons donné des solutions qui permettent d'améliorer les imprécisions de l'appartenance d'un point au modèle révélées par l'algorithme de visualisation. La maquette du modèle que nous avons conçue ne traite que deux quadriques simples (sphère et cylindre) mais l'approche de la méthode, non spécifique à ces deux

quadriques, permet une extension possible à d'autres surfaces plus complexes (surfaces paramétriques).

CHAPITRE 4.

RÉALISATION, MISE EN ŒUVRE ET COMPLEXITÉ.

1. INTRODUCTION.

Afin de valider la méthode que nous avons présentée dans les chapitres 2 et 3, nous avons mis en œuvre et programmé les différents algorithmes qui correspondent aux concepts que nous avons formalisés dans les deux chapitres précédents. L'ensemble de cette programmation a permis la réalisation d'une maquette permettant la construction d'objets B-Rep par opérations booléennes. Le terme de maquette est un choix délibéré dans le sens où ce que nous avons mis en œuvre se devait de valider les approches proposées en respectant totalement la formalisation et, donc, sans rechercher à tout prix les meilleures performances.

Dans ce chapitre, nous allons présenter les différents algorithmes que nous avons développés pour valider notre théorie. Ce chapitre va se décomposer en deux sections principales. La première section correspond à la méthode des opérations booléennes sur deux objets B-Rep présentée au premier chapitre. Elle présente le modeleur sur lequel s'appuient les différents algorithmes, explique les différents algorithmes implémentés pour la réalisation de la méthode et se termine par une série de résultats obtenus et par les différents problèmes rencontrés. La seconde section qui correspond au deuxième chapitre reprend avec plus de détails les différents algorithmes mis en œuvre pour la réalisation des opérations booléennes sur des quadriques. Elle a un schéma identique à la première section à savoir présentation du modeleur, des différents algorithmes et des résultats obtenus.

L'ensemble des algorithmes de la première section s'appuie sur le modeleur SACADO qui est le projet fédérateur de notre laboratoire. Ces algorithmes ont été écrits en langage C. Par contre, comme SACADO est actuellement entièrement réécrit en langage C++ , les algorithmes de la seconde section ainsi qu'un modèle spécifique ont été implémentés en langage C++ pour une intégration ultérieure à SACADO plus facile. L'ensemble de ces programmes ont été testés sur des stations SUN sous le système SUN OS 4.1.1 unix BSD4.3.

2. OPÉRATIONS BOOLÉENNES SUR DES POLYÈDRES.

2.1. Le modèle de SACADO.

L'ensemble des travaux de recherches présentés dans cette thèse s'intègre dans une mise en œuvre globale de systèmes de CAO développés dans notre laboratoire et réunis sous le sigle SACADO qui correspond aux termes Système Adaptatif de Conception et d'Aide au Développement assisté par Ordinateur. Le système SACADO est un projet fédérateur en perpétuelle évolution. Son but est de rassembler dans un même outil l'ensemble des travaux réalisés par les différentes équipes du laboratoire (dialogue, modélisation et visualisation).

En fait, SACADO peut être considéré à la fois comme un outil de CAO ou comme une base de développement pour systèmes de CAO. Ce point de vue est générique et ainsi, toute implantation SACADO est construite avec les mêmes outils et la même méthodologie. Une grande partie de son développement est consacrée au dialogue homme-machine : SACADO est basé sur un certain nombre de concepts de dialogue et est articulé autour d'une hiérarchie de menus. Cependant, le sujet que nous traitons n'a besoin que des procédures et fonctions du modèle géométrique 3D implanté dans SACADO. Le modèle géométrique 3D de la version de SACADO que nous avons utilisée est un modèle de type B-Rep polyédrique de style "arête ailée". En tant que tel, il correspond à un graphe Faces-Arêtes-Sommets et il est constitué de sept tables :

1. une table des objets 3D ou polyèdres : un objet est entre autres caractérisé par un ensemble de numéros de contours 3D de la sixième table, par le numéro de la première et de la dernière faces du volume ainsi que des premiers et derniers sommets et arêtes pour un rapide parcours des faces, arêtes et sommets du volume,
2. une table de faces et de contours : chaque contour peut être un contour extérieur ou un contour intérieur. Si le contour est un contour extérieur, on connaît le numéro du premier contour intérieur s'il existe. Si le contour est un contour intérieur on connaît le numéro du contour intérieur suivant,
3. une table d'arêtes : chaque arête est caractérisée par ses numéros de sommets origine et extrémité. Ces deux sommets définissent un sens de parcours intrinsèque de l'arête. On connaît le contour empruntant l'arête dans ce sens (nommé contour gauche) et le contour empruntant l'arête en sens inverse (nommé contour droit). Sont connus éalement les numéros des arêtes suivant l'arête courante sur les contours gauches et droits et sur le volume,
4. une table des sommets : elle comprend les coordonnées de chaque sommet et le numéro de l'objet auquel il appartient,
5. une table des vecteurs,

6. une table des contours 3D avec un lien vers le modèle géométrique 2D,
7. une table des transformations géométriques.

Les procédures et fonctions de SACADO que nous avons utilisées concernent la création et la consultation de différentes entités composant un objet. Dans la suite de cette section sur les opérations booléennes sur des polyèdres, les procédures et fonctions qui feront référence au modèle commenceront par MD.

2.2. Structures de données avec justification du choix.

L'idée principale de la méthode des opérations booléennes sur des polyèdres (voir chapitre 2 "Opérations booléennes sur des polyèdres : la méthode des sections") est de traiter les objets face par face en leur appliquant une opération booléenne en deux dimensions adéquate par rapport à celle en trois dimensions. L'algorithme des opérations booléennes en deux dimensions demande une structure de données qui permet d'effectuer des opérations booléennes sur des polygones. Cependant, comme la reconstruction de l'objet, une fois toutes les faces traitées, s'effectue en considérant les informations topologiques de chaque point, des informations ont été ajoutées à chaque point en plus des informations nécessaires à la réalisation des opérations booléennes sur des polygones. La structure de polygone est la seule structure utilisée pour effectuer les opérations booléennes en trois dimensions. Le résultat de l'opération booléenne en deux dimensions sur la face peut donner plusieurs polygones résultats. Comme le nombre de ces polygones n'est pas connu a priori, une structure en listes a été choisie.

```
ListePolygone =   Enregistrement
                      ContourExterieur : ListePoints ;
                      ContoursInterieurs : ListeTrous ;
                      Suivant : ListePolygone ;
                  FinEnregistrement ;

ListeTrous =      Enregistrement
                      Trou : ListePoints ;
                      Suivant : ListeTrous ;
                  FinEnregistrement ;

ListePoints =     Enregistrement
                      Point : TypePoint ;
                      Précédent : ListePoints ;
                      Suivant : ListePoints ;
                      Saut : ListePoints ;
                  FinEnregistrement ;
```

```
TypePoint =        Enregistrement
                       Numero : Entier ;
                       Solide : Entier ;
                       (X,Y,Z) : Coordonnees ;
                       ListeTopo : ListeTopologie ;
                       Marque : Booléen ;
                   FinEnregistrement ;
```

A un point est associée une liste de topologie. On associe une liste de topologie plutôt qu'une seule topologie donnée car le même point peut être soumis plusieurs fois à une opération booléenne en deux dimensions (par exemple lorsque la section n'est pas composée que d'un seul polygone). Cependant le parcours des opérations booléennes en deux dimensions tient compte du type du point (il faut savoir si c'est un sommet ou un point d'intersection entre les deux polygones opérandes). Dans ce cas, lorsque l'opération booléenne est réentrante (par exemple une série de différences sur un polygone de départ), un point d'intersection entre la face traitée et la section devient un sommet de la face.

```
ListeTopologie =   Enregistrement
                       Topologie : TypeTopologie ;
                       Suivant : ListeTopologie ;
                   FinEnregistrement ;
```

```
TypeTopologie =    Enregistrement
                       Type : Type_P ;          {Type du point}
                       TypeRelie : Type_P ;     {Type du point relié}
                       Face1 : Entier ;         {numéro de la face traitée}
                       Trou1 : Entier ;         {numéro si la face est un trou}
                       Face2 : Entier ;         {numéro de la face coplanaire}
                       Trou2 : Entier ;         {numéro si la face coplanaire est un trou}
                       Arete1 : Entier ;        {arête dont le point est l'extrémité}
                       Arete2 : Entier ;        {arête dont le point est l'origine}
                       Arete3 : Entier ;        {arête dont le point relié est l'extrémité}
                       Arete4 : Entier ;        {arête dont le point relié est l'origine}
                       Coplanaire : Booléen ;   {vrai si traitement de 2 faces coplanaires}
                       Droites : ListeDroites ; {ensemble des droites donnant le point de la
                                                 section}
                   FinEnregistrement ;
```

```
Type_P = (NonRelie , T_Sommet , T_Section , T_Inter , T_Sommet_Inter , T_Section_Inter , T_Termine) ;
```

Le type T_Sommet correspond à un sommet de la face. Le type T_Section correspond à un point de la section. Le type T_Inter correspond à un point d'intersection franc c'est à dire "au milieu" d'une arête et "au milieu" d'un segment de la section. Le type T_Sommet_Inter correspond à un sommet de la face qui est aussi un point d'intersection. Le type T_Section_Inter correspond à un point de la section qui est aussi un point d'intersection. Le type T_Termine correspond à un point des faces créé dans le modèle comme sommet de l'objet final (c'est-à-dire après construction).

2.3. Algorithmes de la méthode.

2.3.1. Recherche de la section.

Nous avons vu dans le deuxième chapitre que l'implémentation de la théorie donnait des temps d'exécution, en pratique, trop importants. Une étude précise a montré que ceci était dû à la recherche de section qui n'était pas optimisée par rapport à la définition des objets B-Rep.

Voici l'algorithme de recherche de section correspondant à la théorie :

```
Procédure RechercheSection (Entrée : face , solide , courbes, matrice ; Sortie : section)
{ face correspond au numéro de la face traitée }
{ solide correspond au numéro de solide sur lequel est recherchée la section }
{courbes correspond aux intersections entre le plan de face et les plans des faces de solide}
{ matrice correspond à la matrice de passage du repère du monde au repère de la face }
{ section correspond à la liste de polygones associée à la section }

Variables locales : ListePointsIntersection ;

Début
   ListePointsIntersection ← Vide ;
      { intersection entre les courbes d'intersections }
   IntersectionEntreLesCourbes (face , courbes , matrice , ListePointsIntersection) ;
      { On relie les différents points d'intersection trouvés }
   RelierPoints (ListePointsIntersection , section) ;
      {On cherche le bon sens pour les contours de la section }
   section ← TrouverSens (section) ;
Fin

Procédure IntersectionEntreLesCourbes (Entrée : face , courbes , matrice ; Sortie : ListePointsIntersection)
{ face correspond au numéro de la face traitée }
{ courbes correspond aux intersections entre le plan de face et les plans des faces de solide }
{ matrice correspond à la matrice de passage du repère du monde au repère de la face }
{ ListePointsIntersection correspond aux points d'intersection entre les courbes }

Début
     Tant que (Non (Vide (Suivant (courbes))))
     Faire   IntersectionUnecourbe (Tete (courbes) , Suivant (Courbes) , matrice , ListePointsIntersection)
             courbes ← Suivant (courbes)
     FinTantQue
Fin
```

```
Procédure IntersectionUneCourbe (Entrée : unecourbe , courbes , matrice ; Sortie : ListePointsIntersection)
{ unecourbe correspond à une courbe d'intersection entre la face traitée et les faces du solide }
{ courbes correspond aux intersections entre le plan de face et les plans des faces de solide }
{ matrice correspond à la matrice de passage du repère du monde au repère de la face }
{ ListePointsIntersection correspond aux points d'intersection entre les courbes }

Début
    Tant que (Non (Vide (courbes)))
    Faire   InterCourbeCourbe (unecourbe , Tete (courbes) , res , point)
            Si (res) Alors
                    Si  AreteCommune (Face (unecourbe) , Face (Tete (courbes)) , matrice , point) Alors
                            ListePointsIntersection ← Cons (point , ListePointsIntersection)
                    FinSi
            FinSi
            courbes ← Suivant (courbes)
    FinTantQue
Fin
```

Voici l'algorithme de recherche de section améliorée :

```
Procédure RechercheSection (Entrée : face , solide , matrice ; Sortie : section)
{ face correspond au numéro de la face traitée }
{ solide correspond au numéro de solide sur lequel est recherchée la section }
{ matrice correspond à la matrice de passage du repère du monde au repère de la face }
{ section correspond à la liste de polygones associée à la section }

Variables locales : ListePointsIntersection ;

Début
   ListePointsIntersection ← Vide ;
      { intersection entre les arêtes de solide et le plan de face }
   IntersectionPlanArêtes (face , solide , matrice , ListePointsIntersection) ;
      { On relie les différents points d'intersection trouvés }
   RelierPoints (ListePointsIntersection , section) ;
      {On cherche le bon sens pour les contours de la section }
   section ← TrouverSens (section) ;
Fin
```

L'ensemble des algorithmes des procédures et fonctions (IntersectionPlanArêtes, RelierPoints et TrouverSens) présentes dans le corps de la procédure RechercheSection est décrit dans l'annexe A.

2.3.2. Opérations booléennes en deux dimensions.

L'algorithme des opérations booléennes en deux dimensions sur deux polygones P_1 et P_2, quelle que soit l'opération booléenne considérée, se divise en trois parties :

- la première partie consiste à calculer tous les points d'intersection entre les deux polygones,

- la deuxième partie consiste à chercher un point de départ sur P_1 et/ou sur P_2 (suivant l'opération booléenne considérée) pour trouver le premier point du polygone résultat. Le cas échéant, cette partie teste tous les cas d'inclusion possibles entre les deux polygones et donne le résultat de l'opération booléenne,
- la troisième partie consiste à former le polygone résultat en parcourant tour à tour et de façon adéquate les deux polygones P_1 et P_2, à partir du point de départ trouvé par la deuxième partie. Pour ce faire, il faut considérer tous les types d'intersection possibles et tous les cas de tangence possibles pour déterminer s'il y a un changement de parcours de polygone ou non.

L'ensemble de ces algorithmes a été décrit dans [BMP 91] [WEA 77]. La difficulté majeure de l'algorithme des opérations booléennes en deux dimensions est le traitement de tous les cas particuliers pouvant se présenter lors du parcours des deux polygones P_1 et P_2 (troisième partie). Suivant les cas d'intersection que l'on rencontre entre les deux polygones, il faut décider sur chaque point d'intersection entre les deux polygones rencontré lors du parcours, si l'on change de polygone ou non. Cette décision tient compte du type d'intersection rencontrée (intersection franche entre les deux segments, sur un sommet des deux segments, ...) et des cas de tangence passés ou à venir (le point d'intersection considéré appartient-il ou non au segment venant d'être parcouru ?).

L'en-tête de la procédure permettant d'effecuter une opération entre deux polygones est donné dans l'annexe A.

2.3.3. Cas particulier des faces coplanaires.

Nous présentons ici les algorithmes nécessaires à la résolution des faces coplanaires. Ces algorithmes ne concernent que deux faces coplanaires (une face du premier objet et une face du second objet). Si une face du premier objet est coplanaire avec plusieurs faces du second objet alors le traitement est réentrant avec le résultat de l'opération booléenne en deux dimensions et la face coplanaire suivante du second objet.

Procédure TraitementFacesCoplanaires (**Entrée** : face1 , face2 , section1 , section2 , ope3d ; **Sortie** : polygone)
{face1 correspond au polygone de la première face coplanaire}
{face2 correspond au polygone de la seconde face coplanaire}
{section1 correspond à la section associée à la première face, donc section sur le solide de face2}
{section2 correspond à la section associée à la seconde face, donc section sur le solide de face1}
{ope3d correspond à l'opération booléenne appliquée en trois dimensions}
{polygone est la liste des polygones résultats du traitement entre les deux face}

Variables locales : /

Début
    Cas ope3d **Suivant**
    Union3D : FaceCoplanaireUnion (face1 , face2 , section1 , section2 , polygone) ;
    Inter3D : FaceCoplanaireIntersection (face1 , face2 , section1 , section2 , polygone) ;
    Diffe3D : FaceCoplanaireDifference (face1 , face2 , section1 , section2 , polygone) ;
    FinCas
Fin

Nous n'allons pas décrire les trois procédures FaceCoplanaireUnion, FaceCoplanaireIntersection et FaceCoplanaireDifference puisqu'elles ont le même schéma d'algorithme. Chacune de ces trois procédures se divise en deux procédures suivant le sens des deux normales des deux faces. Dans ces six procédures, seules les opérations booléennes en deux dimensions appliquées aux faces et aux sections sont modifiées (voir section 2.5 du chapitre 2). Nous n'allons détailler que la procédure FaceCoplanaireUnionMemeSens qui traite l'union sur deux objets pour le cas de deux faces coplanaires ayant leurs normales dans la même direction.

```
Procédure FaceCoplanaireUnionMemeSens (Entrée : face1 , face2 , section1 , section2 ; Sortie : polygone)
{face1 correspond au polygone de la première face coplanaire}
{face2 correspond au polygone de la seconde face coplanaire}
{section1 correspond à la section associée à la première face, donc section sur le solide de face2}
{section2 correspond à la section associée à la seconde face, donc section sur le solide de face1}
{polygone est la liste des polygones résultats du traitement entre les deux face}

Variables locales : /

Début
    PU = Vide ;
    Traitement2D (face1 , face2 , Union2D , PU) ;
    Si      (Egal (face1 , section2) et Egal (face2 , section1)) Alors
            polygone ← PU ;
    SinonSi (Egal (face2 , section1)) Alors
                {On met la topologie de PU en sommets pour effectuer le traitement 2D}
            TransformeEnSommets (PU) ;
            Traitement2D (PU , section2 , Diffe2D , polygone) ;
    SinonSi (Egal (face1 , section2)) Alors
                {On met la topologie de PU en sommets pour effectuer le traitement 2D}
            TransformeEnSommets (PU) ;
            Traitement2D (PU , section1 , Diffe2D , polygone) ;
    Sinon   LR = Vide ;
            TransformeEnSommets (PU) ;
            Traitement2D (PU , section1 , Diffe2D , LR) ;
            TransformeEnSommets (LR) ;
            Traitement2D (LR , section2 , Diffe2D , polygone) ;
    FinSi
            {On garde la topologie adéquate pour chaque point}
    RemetBonneTopologie (polygone) ;
Fin
```

2.3.4. Construction de l'objet résultat.

L'analyse formelle pour la construction de l'objet suivant la représentation du modèle B-Rep a été décrite dans la section 2.6 du chapitre 2. Nous rappelons que les points identiques du solide sont retrouvés par rapport à la topologie dont ils sont issus. Les différents cas proviennent des cas d'intersection possibles entre la face et la section et ils sont au nombre de quatre :

- intersection entre une arête de la face et un segment de la section,
- un sommet de la face appartient à un segment de la section,
- un point de la section appartient à une arête,
- un sommet de la face correspond à un point de la section.

Ces quatre cas ont été programmés. Cependant nous n'allons développer que le premier cas, car les trois autres reprennent un schéma d'algorithme identique. Toutes les parties de faces trouvées par opération booléenne sont parcourues point par point. Si le point a déjà été traité par une recherche de point précédente, le point suivant est traité.

Nous sommes donc dans le cas où l'on traite la recherche de point identique pour un point d'intersection entre une arête d'une face et un segment de la section. L'algorithme correspondant à cette topologie est le suivant :

```
Procédure PointInterRelieAvecInter (Entrée : liste ; Entrée-Sortie : T)
{liste correspond à une liste de points arrêtée sur le point à traiter}
{T est le tableau des polygones de faces indexé par le numéro des faces}

Variables locales : Topo , F1 , F2 , Fa , A1 ;

Début
            {On recherche la topologie du point}
    Topo ← Tete (TopologiePoint (Tete (liste))) ;
            {On cherche le numéro de la face traitée}
    F1 ← FaceTraitee (Topo) ;
            {On cherche le numéro de la face ayant donnée le segment de l'intersection}
    Fa ← Tete (Droites (Topo)) ;
            {On cherche l'arête de l'intersection}
    A1 ← PremiereArete (Topo) ;
            {On crée le point dans le modèle}
    i ← MDCreerPoint (Coordonnees (Tete (liste))) ;
    liste → Point.Numero = i ;
    liste → Point.ListeTopo → Topologie.Type ← T_Termine ;
            {On cherche la face adjacente en l'arête}
    F2 ← MDFaceAdjcente (A1 , F1) ;
            {Recherche du point dans les autres faces}
            {dans la face F2}
    RecherchePointInter (T [F2] , -A1 , Fa , i) ;
            {dans la face Fa}
    RecherchePointSection (T [Fa] , Fa , F1 , F2 , i) ;
Fin
```

L'ensemble des algorithmes correspondant aux procédures RecherchePointInter et RecherchePointSection présentes dans le corps de l'algorithme de la procédure PointInterRelieAvecInter est donné dans l'annexe A.

2.4. Complexité et résultats.

Dans cette section, nous allons étudier la complexité de l'algorithme des opérations booléennes sur les polyèdres en proposant :

- la complexité dans le pire des cas,
- la complexité dans le meilleur des cas,
- la complexité de certains exemples simples.

Nous terminons cette section par quelques temps d'exécution illustrés par des graphiques.

Pour calculer la complexité dans le pire des cas, nous allons chercher la complexité des différentes parties de l'algorithme principal, à savoir :

1. test des englobants parallélépipédiques des deux solides opérandes afin de détecter si les solides sont disjoints,
2. pour chaque face des deux solides :
   a. test de l'englobant parallélépipédique de la face avec l'englobant parallélépipédique du second solide pour savoir si la face traitée peut traverser ou non le solide,
   b. recherche de la section du second solide par rapport au plan de la face,
   c. opération booléenne en deux dimensions entre la face et la section,
3. construction du solide résultat suivant le modèle de représentation B-Rep à partir des parties de faces des deux objets opérandes intervenant dans la frontière de l'objet résultat.

Reprenons ces différents points (1 à 3) afin de déterminer la complexité dans le pire des cas de l'algorithme global. Pour ce faire, donnons les notations que nous avons utilisées pour les différentes entités intervenant dans la complexité :

- notons nf_1 et nf_2 le nombre des faces du premier et du second solide,
- notons nar_1 et nar_2 le nombre d'arêtes du premier et du second solide ayant une intersection avec le plan de la face traitée,
- notons na le nombre d'arêtes de la face et nss le nombre de segments de la section.

1. Le test sur les englobants parallélépipédiques des deux solides est composé de trois comparaisons de coordonnées. Dans le pire des cas, les trois comparaisons sont effectuées. Quelle que soit la disposition des englobants, ce test prendra toujours le même temps d'exécution. Notons ce temps par une constante K.

2a. Le test entre l'englobant d'une face et l'englobant d'un solide consiste comme pour le pas 1, à tester les six coordonnées des englobants. Ce test correspond à une constante qui est identique à celle des englobants des solides. Cette constante est K. Comme ce test est effectué pour toutes les faces des deux solides, nous obtenons :

$(nf_1 + nf_2).K$

2b. La recherche de section consiste à trouver l'ensemble des points formant la section à partir des intersections entre le plan de la face traitée et les arêtes du second solide puis à parcourir l'ensemble de ces points. Le nombre d'arêtes ayant une intersection avec le plan de la face traitée dépend de la position de ce plan. Le traitement pour parcourir les points dépend du nombre d'arêtes ayant une intersection avec le plan de la face. Notons T_1 le temps nécessaire au traitement d'un point de la section. L'algorithme de recherche de section est donc fonction de $T_1.nar_2$ (f).

2c. L'algorithme des opérations booléennes est un algorithme dont la complexité est connue. Les opérations booléennes en deux dimensions que nous développons s'effectuent sur la face et la section trouvée en 2b. L'algorithme des opérations booléennes en deux dimensions est fonction du nombre d'arêtes de la face par le nombre de segments de la section. Il est donc fonction de T_2.na(f).nss(f).

3. La construction du solide résultat consiste à parcourir, point par point, toutes les parties de faces trouvées par opérations booléennes en deux dimensions. Pour chaque point on recherche en considérant leurs données topologiques les points identiques dans les autres faces. Le parcours pour chaque point prend au plus un temps T_3 qui dépent du nombre de faces où l'on doit rechercher le point. Pour une face où la recherche doit s'effectuer on parcourt tous les points de cette face. L'algorithme de construction du solide est donc fonction de :

$$\sum_{\text{P= Points du solide résultat}} \left(T_3 \times \sum_{\text{F=1}}^{\text{Nombre de faces dans lesquelles on recherche P}} \text{NombrePoints}(F)\right)$$

La complexité de l'algorithme général est donc :

$$\begin{aligned}
&K \\
&+ (nf_1 + nf_2) \times K \\
&+ \sum_{f=1}^{nf_1} \left(T_1 \times nar_2(f) + T_2 \times \left(na(f) \times nss(f)\right)\right) \\
&+ \sum_{f=1}^{nf_2} \left(T_1 \times nar_1(f) + T_2 \times \left(na(f) \times nss(f)\right)\right) \\
&+ \sum_{\text{P= Points du solide résultat}} \left(T_3 \times \sum_{\text{F=1}}^{\text{Nombre de faces dans lesquelles on recherche P}} \text{NombrePoints}(F)\right)
\end{aligned}$$

Dans le meilleur des cas, la complexité est K/3. En effet, ce cas arrive lorsque les deux objets sont distincts et lorsque le test des englobants est validé à la première comparaison de coordonnées.

Donnons maintenant la complexité d'une opération booléenne (union, intersection ou différence) sur des objets opérandes simples calculée en reprenant tous les pas cités plus haut (du pas 1 au pas 3). Ces objets opérandes simples sont :

- deux cubes,
- deux sphères de polygonalisation d (c'est-à-dire approchée par d^2 facettes),
- deux cylindres de polygonalisation d,
- deux cônes de polygonalisation d.

Pour deux cubes la complexité est fonction de :

$$13K+(8T_1+16T_2)\times 6\times 2+16\times T_3\times \text{Nombre de points du solide résultat}$$

Pour deux sphères la complexité est fonction de :

$$(1+2\times d^2)\times K+(3\times d\times T_1+4\times d\times T_2)\times 2\times d^2+4\times d\times T_3$$

$\times$Nombre de points du solide résultat.

Pour deux cylindres le complexité est fonction de :

$$(5+2\times d^2)\times K+(2\times d\times T_1+d^2\times T_2)\times(2\times d+4)+d^2\times T_3$$

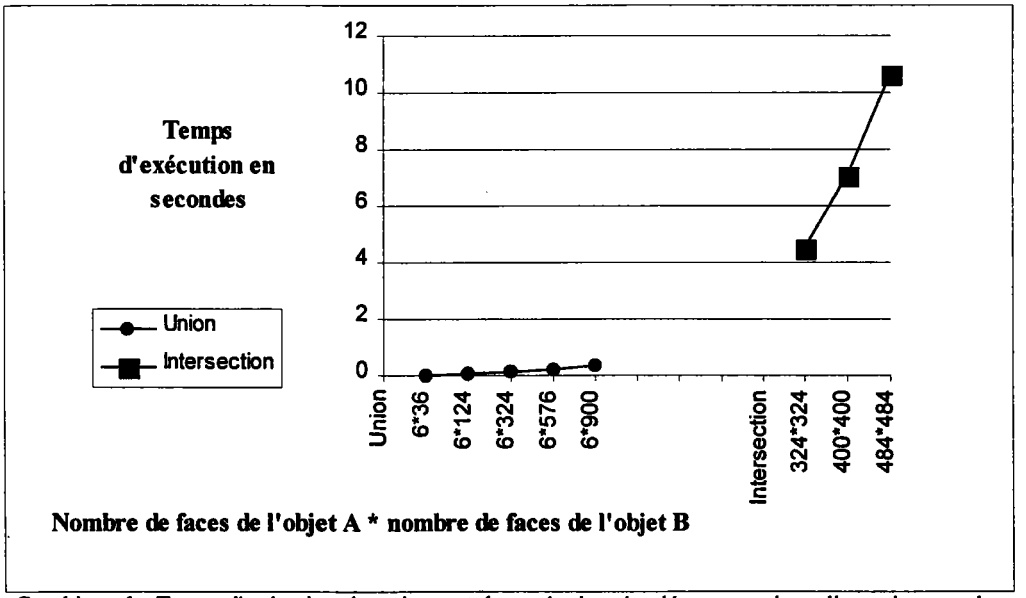
$\times$Nombre de points du solide résultat.

Pour deux cônes la complexité est fonction de :

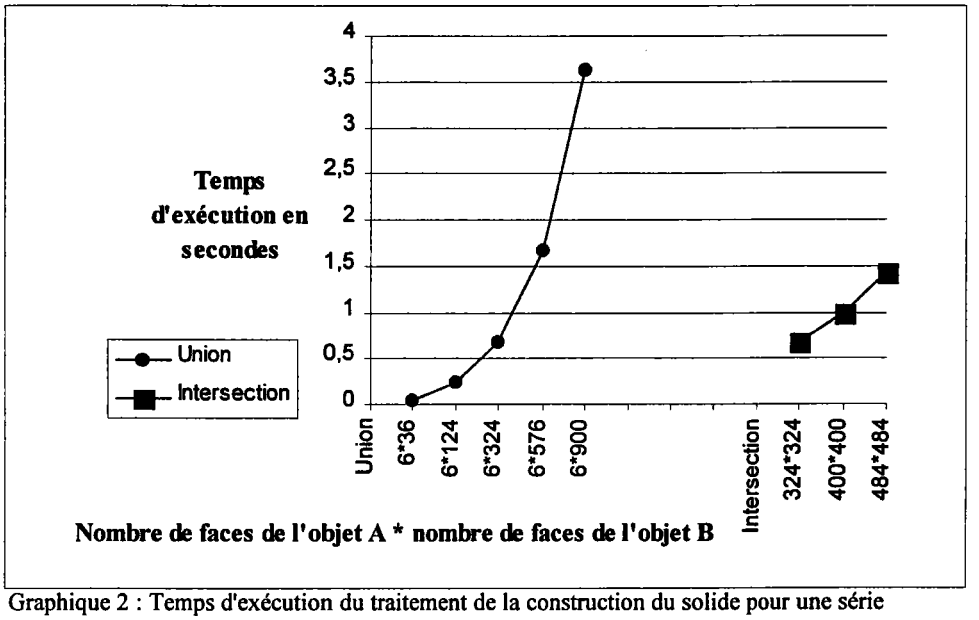
$$(3+2\times d^2)\times K+(2\times d\times T_1+d^2\times T_2)\times(2\times d+2)+d^2\times T_3$$

$\times$Nombre de points du solide résultat.

Pour illustrer la complexité de l'algorithme des opérations booléennes sur des polyèdres, nous proposons des temps d'exécution pour le traitement des opérations booléennes en deux dimensions sur les faces des deux objets (graphique 1) et des temps d'exécution pour le traitement de la construction du solide résultat (graphique 2) pour les mêmes séries d'opérations booléennes sur les mêmes objets.



Graphique 1 : Temps d'exécution du traitement des opérations booléennes en deux dimensions sur les faces des deux objets pour une série d'unions et une série d'intersections.



Graphique 2 : Temps d'exécution du traitement de la construction du solide pour une série d'unions et une série d'intersections.

Nous pouvons constater, comme le laissait prévoir l'étude de la complexité théorique, que le temps d'exécution (qui est égal au temps d'exécution des opérations booléennes en deux dimensions sur les faces + le temps d'exécution de la construction solide) d'une opération booléenne sur deux objets polyédriques dépend, d'une part, du nombre de faces traitées (graphique 1) et, d'autre part, du nombre de points à prendre en considération pour la construction solide (graphique 2).

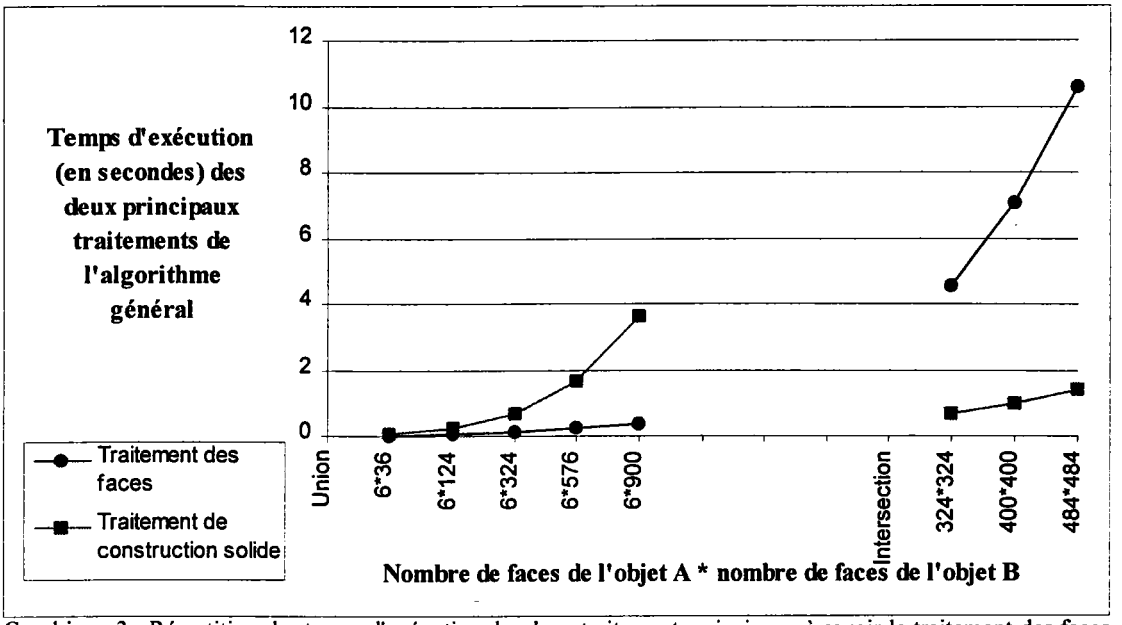
En effet, pour une intersection entre deux objets le nombre de points de l'objet final peut être moindre que pour une union entre les deux mêmes objets, ce qui donne des temps d'exécution moins importants pour la construction solide (graphique 2).

Nous pouvons également noter (graphique 1) que pour une face donnée en cours de traitement, plus le second solide a d'arêtes, plus le temps d'exécution de traitement de cette face sera important (exemple : le temps d'exécution du traitement des faces de 324*324 est supérieur à celui de 900*6 même si le nombre de faces des deux objets au départ est inférieur à celui du second traitement (648 < 906)).

A partir du graphique 3, nous observons que le rapport :

$$\frac{\text{Temps d'exécution du traitement des faces}}{\text{Temps d'exécution du traitement de la construction solide}}$$

n'est pas constant. Nous ne pouvons donc pas supposer, a priori, l'importance d'une partie (soit traitement des faces, soit traitement de la construction solide) quelle que soit l'opération booléenne effectuée et quels que soient les objets opérandes.



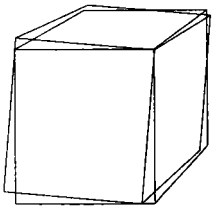
Graphique 3 : Répartition des temps d'exécution des deux traitements principaux à savoir le traitement des faces et le traitement de construction du solide par rapport au temps total d'exécution pour deux séries d'opérations booléennes différentes : union et intersection

2.5. Problèmes rencontrés.

Les problèmes rencontrés pour le traitement des opérations booléennes sur des objets polyédriques sont essentiellement des problèmes de réalisation. Nous avons expliqué ces différents problèmes dans la section 2.8 du chapitre 2. Nous allons maintenant détailler ces différents problèmes par des exemples et envisager quelques solutions permettant de les minimiser. Les problèmes que nous avons eus sont de deux types :

- problème de gestion d'erreurs dans les différents tests d'appartenance d'une entité à une autre (exemple : appartenance d'un point à une droite),
- problème de temps d'exécution trop importants pour la réalisation des opérations booléennes.

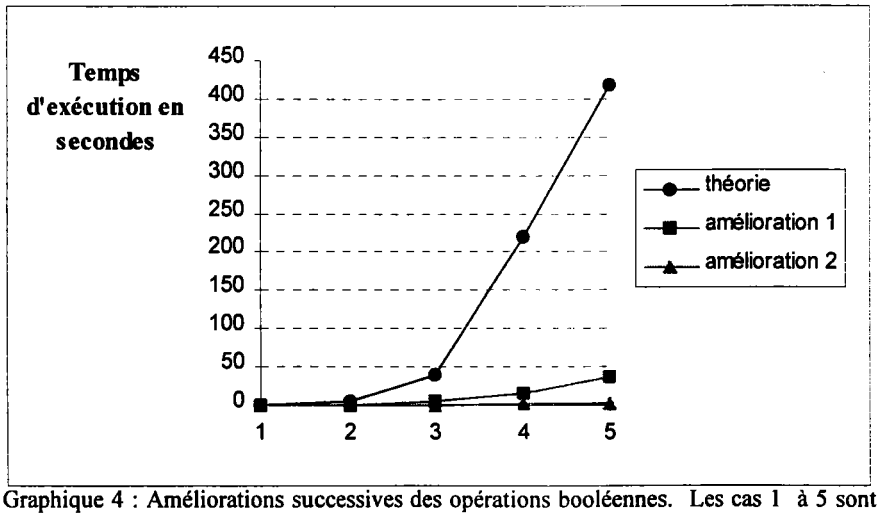
Un exemple classique pour montrer les problèmes de gestion d'erreurs est le cas d'une opération booléenne entre deux cubes tel que le second cube soit le résultat d'une transformation très faible sur le premier :



Supposons que l'écart entre un point du premier cube et son transformé sur le second cube soit de l'ordre de l'epsilon théorique que nous avons choisi pour effectuer des tests entre réels, il se peut que pour le traitement d'une face *f*, une arête *a* de cette face ait deux intersections alors que pour le traitement de la face adjacente à *f* (en l'arête *a*), l'arête *a* n'ait qu'une seule intersection. Une des solutions possibles dans ce cas de configuration d'erreur est de compter le nombre d'intersections sur une arête donnée afin de mettre en œuvre un algorithme de décision pour les intersections de la même arête lors du traitement de la face adjacente.

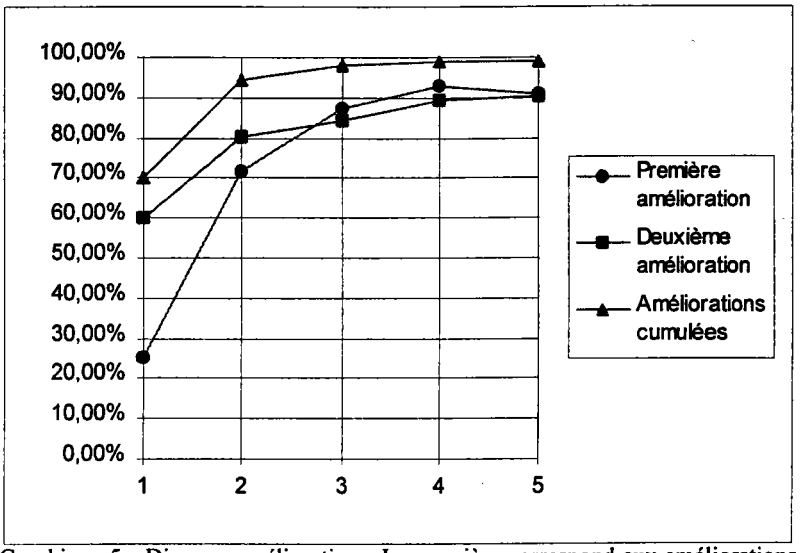
L'implémentation de la théorie a montré des temps de calcul trop importants (voir les temps représentés par les histogrammes "théorie" dans le graphique 4). Une étude appropriée a permis de réaliser que ces temps de calcul trop élevés étaient dus à une recherche de section trop générale lorsque la méthode n'est employée que sur des objets B-Rep (voir section 2.8 du chapitre 2). L'optimisation adéquate a été réalisée (voir section 2.8 du chapitre 2), ce qui a donné des améliorations de temps conséquents surtout pour les cas les plus défavorables (voir les temps de calcul représentés par les histogrammes "pratique" dans le graphique 4).

Comme les temps de calcul n'étaient pas encore suffisamment satisfaisants, nous avons cherché à améliorer encore les performances. Pour ce faire, nous avons modifié la méthode de construction du solide qui utilisait des listes de polygones. Nous avons remplacé cette liste par un tableau indexé par le numéro de la face traitée afin d'obtenir un accès direct sur la liste de polygones associée à un numéro de face (voir les temps de calcul représentés par les histogrammes "pratique améliorée" dans le graphique 4).



Graphique 4 : Améliorations successives des opérations booléennes. Les cas 1 à 5 sont des unions entre une sphère et un cube en augmentant au fur et à mesure le degré de polygonalisation pour la sphère. Degré 6 pour le cas 1, 12 pour le cas 2, 18 pour le cas 3, 24 pour le cas 4 et 30 pour le cas 5.

Afin d'apporter un point de vue différent sur les améliorations effectuées, nous donnons ci-après le graphique des pourcentages de gain de temps en fonction des différents cas (1 à 5).



Graphique 5 : Diverses améliorations. La première correspond aux améliorations engendrées par la nouvelle recherche de section. La seconde correspond aux améliorations de la construction de solide.

3. OPÉRATIONS BOOLÉENNES SUR LES QUADRIQUES.

Après avoir formalisé de façon rigoureuse les opérations booléennes sur les objets polyédriques afin que la méthode puisse s'étendre à des objets plus complexes, nous avons montré que la meilleure approche pour traiter les surfaces gauches est d'enrober la méthode sur les objets polyédriques par un pré-traitement et un post-traitement (voir chapitre 3). Le pré-traitement consiste à approcher les surfaces gauches des deux objets opérandes par des facettes et le post-traitement consiste à supprimer les facettes des surfaces gauches de l'objet résultat afin qu'il ait les mêmes caractéristiques que les deux objets opérandes.

La réalisation des opérations booléennes sur des objets à surfaces gauches demande la réalisation d'un modèle capable de supporter la représentation de tels objets. Le modèle doit également permettre une facettisation des surfaces d'un objet ainsi que la suppression de ces facettes une fois l'opération booléenne effectuée. Dans cette section, nous allons décrire les algorithmes nécessaires à l'implémentation du modèle, de l'approximation et de la suppression des facettes.

3.1. Modèle pour objets à surfaces quelconques.

Pour la réalisation des opérations booléennes sur des objets à surfaces quelconques, nous avons implémenté un modèle capable de représenter de tels objets. Nous n'avons pas pu utiliser le modèle SACADO que nous avons présenté dans la section 2.1 de ce chapitre, puisque SACADO ne s'applique qu'aux objets polyédriques. Les diverses possibilités offertes par un langage objet (classes, généricité, héritage, surcharge des opérateurs, ...) nous ont amenés à écrire les différents algorithmes en langage C++. De plus, SACADO étant actuellement en cours de traduction en langage C++, le développement du modèle pour des objets à surfaces quelconques en langage C++ permet ainsi une intégration future plus simple.

Nous allons présenter la structure du modèle. Les objets du modèle ne peuvent avoir, actuellement, que deux quadriques : la sphère et le cylindre. Cependant comme le modèle a été implémenté en C++ et parce que l'approche de la méthode le permet, l'ajout de nouvelles surfaces est aisé (héritage).

Le schéma suivant donne l'ensemble des classes du modèle. Le formalisme utilisé est celui de Grady Booch [BOO 91].

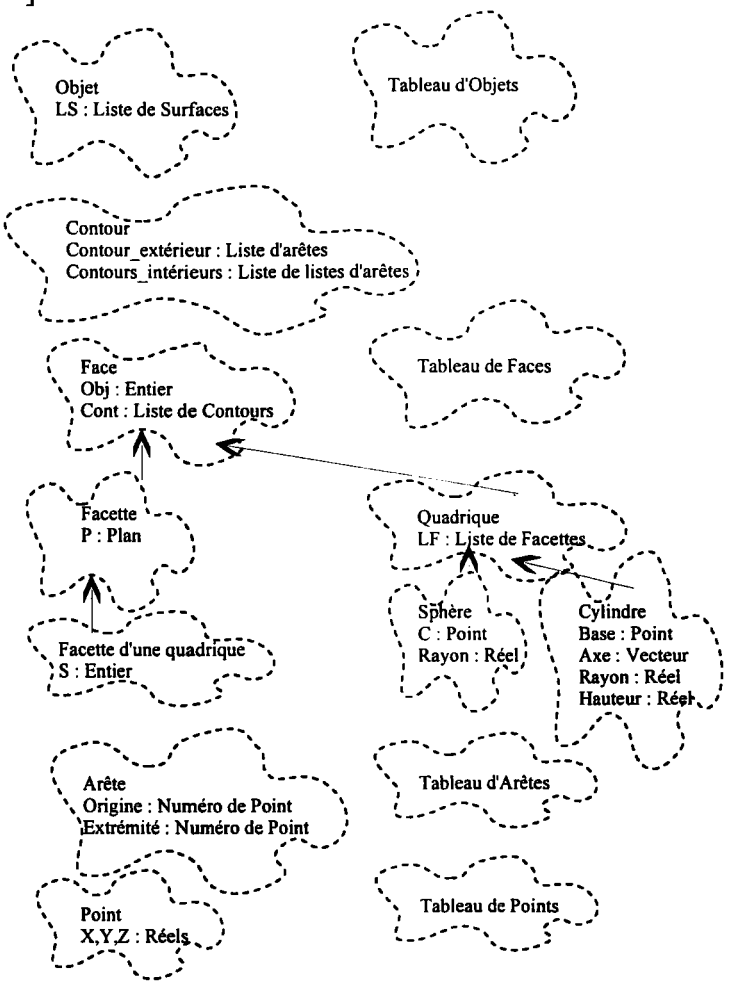


Schéma 1 : Les différentes classes du modèle.

Nous n'allons pas détailler l'ensemble des classes définies pour la création du modèle, cependant la classe "Face" est la plus importante car la plus complexe. Sa définition est donnée dans l'annexe B ainsi que toutes les procédures et fonctions disponibles pour accéder au modèle.

3.2. Algorithmes de la méthode.

Pour réaliser les opérations booléennes sur des objets à surfaces quelconques, nous avons choisi d'encapsuler les opérations booléennes sur les polyèdres en une phase de pré-traitement consistant à transformer les objets en polyèdres et une phase de post-traitement qui permet de supprimer les faces servant à approcher une surface par polygonalisation (voir sections 3 et 4 du chapitre 3). Donnons dans les deux prochaines sections les algorithmes nécessaires à la réalisation de ces deux phases.

3.2.1. Phase de pré-traitement : approximation par des facettes.

L'approximation (voir section 4.2.1 du chapitre 3) consiste à approcher des surfaces quelconques par une collection de facettes. Dans la maquette que nous avons réalisée, seules deux quadriques simples sont considérées : la sphère et le cylindre. Cette maquette permet simplement de valider les différents concepts d'une approche générale qui permet une extension à des surfaces plus complexes. Lorsqu'il n'existe pas de contour défini sur la surface, la surface représente la frontière d'un objet. Dans ce cas pour notre maquette, une approximation classique a été implémentée. Cependant dès qu'il existe un ou plusieurs contours définissant des parties de surfaces intervenant dans la surface, une triangulation est mise en oeuvre pour chaque contour. Dans cette section, nous allons donner l'algorithme global consistant à trianguler une partie de surface à partir d'un contour d'arêtes donné.

```
Procédure Triangulation (Entrée : Lpt , Num ; Entrée-Sortie : S)
{Triangulation du contour concave représenté par la liste d'arêtes Lpt sur la surface S}
{Lpt : liste des arêtes du contour}
{Num : numéro de la surface dans le modèle S : Objet de type surface}

Début
    Tant que (Longueur (Lpt)) Faire
                {On recherche le plus petit angle et suivant la valeur de ce dernier, on construit le
                triangle}
        PlusPetitAngle (Lpt , S , lieu , angle , a1 , a2 , pnouveau)
        Cas angle Suivant
        < π/2 :   PremierCas (a1 , a2 , Num , Lpt , lieu) ;
        <2π/3:    DeuxiemeCas (a1 , a2 , pnouveau , Num , Lpt , lieu) ;
        sinon:    TroisiemeCas (a1 , a2 , pnouveau , Num , Lpt , lieu) ;
        FinCas
    MDAjouterFacette (MDConsOrigine (Tete (Lpt)) , MDConsExtremite (Tete (Lpt)) , MDConsExtremite
    (Tete (Suivant (Lpt))) , Contour (Lpt) , Num) ;
    FinTantQue
Fin
```

L'ensemble des procédures et fonctions présentes dans le corps de la procédure Triangulation sont décrites dans l'annexe B.

3.2.2. Phase de post-traitement : l'annulation de la facettisation.

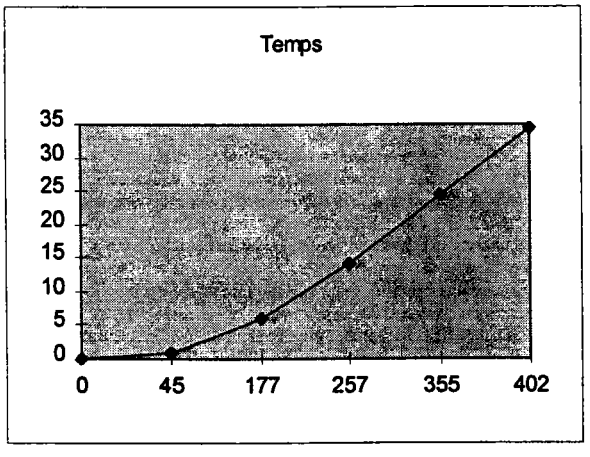
Après avoir effectué les opérations booléennes sur des objets à surfaces quelconques devenus polyédriques grâce à l'algorithme décrit dans la section 3.2.1, l'objet obtenu est un polyèdre même s'il est composé de surfaces quelconques. L'ensemble des facettes qui ont servi à rendre les objets opérandes polyédriques et dont certaines parties forment la frontière de l'objet résultat doivent être éliminées (voir section 4.2.2 du chapitre 3). Ainsi, l'objet résultat est du même modèle que les objets opérandes. Pour chaque surface de l'objet résultat provenant d'une surface approchée des objets opérandes, il suffit d'éliminer toutes les arêtes dont les faces adjacentes représentent une même surface. Ensuite, avec les arêtes restantes on reforme les contours sur la surface de la même façon que pour la recherche de section (voir section 2.3.1 de ce chapitre).

```
Procédure Quadrique::SupprimerFacettes (Entrée : NS)
{Supprime toutes les facettes de la quadrique}
{NS est le numéro de la surface dans le modèle}

Début
        {Recherche des arêtes à garder et à supprimer}
    agarder ← ConsVide ; asupprimer ← ConsVide ;
        {parcours de toutes les facettes de la surface et construction des listes d'arêtes}
    Tant que (Non (Vide (LF))) Faire
        contour ← MDConsContoursSurface (Tete (LF)) ;
        TeteContour (contour).SupprimerAretes (agarder , NS , asupprimer) ;
            {On supprime la facette}
        MDDetruireSurface (Tete (LF)) ;
        LF ← Suivant (LF) ;
    FinTantQue
        {On supprime les points et les arêtes inutiles dans le modèle}
    pointsagarder ← ConsVide ;
    MDListeAretesListePoints (agarder , pointsagarder) ;
    Tant que (Non (Vide (asupprimer))) Faire
        po ← MDConsOrigine (Tete (asupprimer)) ; pe ← MDConsExtremite (Tete (asupprimer)) ;
        Si      (Non (Appartient (po , pointsagarder))) Alors
                MDDetruirePoint (po) ;
        FinSi
        Si      (Non (Appartient (pe , pointsagarder))) Alors
                MDDetruirePoint (pe) ;
        FinSi
        MDDetruireArete (Tete (asupprimer)) ; asupprimer ← Suivant (asupprimer) ;
    FinTantQue
        {Mise à jour des contours sur la surface}
    LC ← ReconstruitContour (agarder , this) ;
Fin
```

3.3. Résultats.

Nous illustrons maintenant les temps de calcul pour l'approximation d'une surface en fonction du nombre de triangles créés :



Graphique 6 : Courbe du temps d'exécution de l'approximation en fonction du nombre de triangles créés.

En supposant que la relation plausible entre le temps et le nombre de triangles créés soit linéaire et d'ordre deux, alors le modèle s'écrit :

$$\text{Temps} = \alpha + \beta x + \gamma x^2 + \varepsilon$$

où x représente le nombre de triangles créés.

Par une régression linéaire [SAP 90], on trouve l'équation qui ajuste au mieux ce modèle et qui s'écrit :

$$\text{Temps} = 2.07 \times 10^{-4} \times x^2 \qquad (1).$$

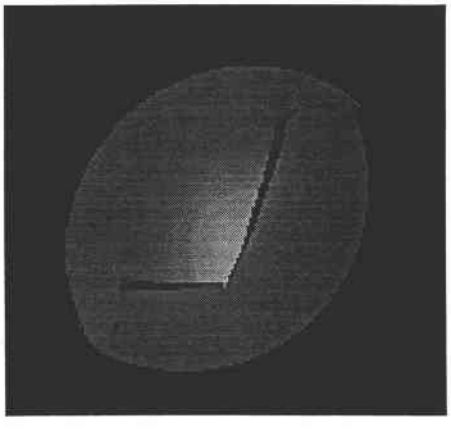
Le coefficient de détermination ajusté R^2 égal à 0.99 reflète la qualité de l'ajustement.

L'équation (1) permet d'estimer le temps de calcul pour un nombre de triangles donné.

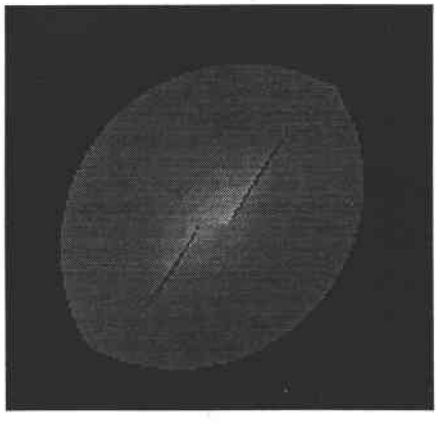
3.4. Problèmes de visualisation du modèle.

Dans la section 4.4 du chapitre 3, nous avons vu, comme nous nous y attendions, qu'il existe des erreurs de visualisation d'un objet au niveau de l'arête. Ceci est dû au fait que les extrémités des arêtes n'appartiennent pas à la surface. Un moindre mal est d'augmenter la polygonalisation des surfaces avant opération booléenne afin de réduire cet écart entre les extrémités des arêtes du solide résultat et la surface.

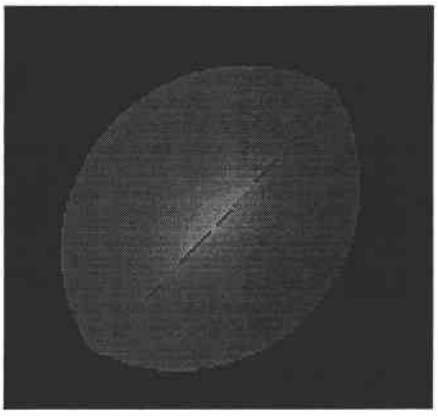
Nous donnons ci-après des exemples qui permettent de valider notre solution.



Exemple 13 : Visualisation de l'intersection de deux sphères de degré de polygonalisation 6.



Exemple 14 : Visualisation de l'intersection de deux sphères de degré de polygonalisation 12.



Exemple 15 : Visualisation de l'intersection de deux sphères de degré de polygonalisation 18.

3.5. Apports de C++ par rapport à C.

Les algorithmes que nous venons de décrire dans cette section ont été écrits en langage C++. Le modeleur SACADO de notre laboratoire étant en train d'être entièrement réécrit dans ce langage, l'intégration ultérieure des différents algorithmes pour les opérations booléennes sera plus simple.

L'utilisation de C++ montre que ce langage permet de développer des algorithmes plus clairs et plus concis. En effet, C++ permet l'héritage ce qui évite notamment d'avoir des séries de "Si ... Alors ... Sinon" imbriqués puisque c'est suivant le type de l'objet manipulé que se fait le routage vers l'algorithme correspondant.

Par exemple au lieu d'avoir l'algorithme :

Si (Surface = Sphère)
Alors TraitementSurSphère ;
SinonSi (Surface = Cylindre)
Alors TraitementSurSphère ;
SinonSi (Surface = Cône)
Alors TraitementSurCône ;
FinSi

on a directement :

S → Traitement ; où la procédure Traitement a été développée pour chaque classe associée à un type d'objet (sphère, cylindre et cône).

De plus, la surchage des opérateurs permet également des algorithmes plus clairs. Il est intuitivement plus évident de lire $V = V_1 + V_2$ que $V = Plus(V_1,V_2)$ si V, V_1 et V_2 sont des vecteurs. C++ est plus fortement typé que C. Ceci implique une compilation moins permissive ce qui réduit considérablement le nombre des erreurs de programmation et d'exécution.

Enfin, la généricité semblait a priori un outil intéressant (ce qui autorise plusieurs types de listes avec exactement les mêmes fonctions d'accès) cependant son implémentation demande un outil spécifique qui n'était pas disponible lors du début de l'implémentation du modèle.

4. CONCLUSION.

Dans ce chapitre, nous venons de présenter une partie du développement des opérations booléennes sur des objets polyédriques et sur des objets à surfaces quelconques. L'implémentation des algorithmes théoriques que nous avons présentés dans les chapitres 2 et 3 a permis de valider notre méthode.

L'implantation des opérations booléennes sur les polyèdres s'est appuyée sur le modeleur qui a été implémenté dans notre laboratoire : SACADO. Sur ce modeleur, se greffent toutes les maquettes des algorithmes et méthodes développées dans notre laboratoire. Les programmes ont été écrits en langage C contrairement aux algorithmes correspondant aux opérations booléennes sur les objets à surfaces quelconques qui ont été développés en langage C++, et ce pour la raison suivante :

> SACADO est en train d'être réadapté à nos besoins en langage C++, l'intégration des opérations booléennes sera donc plus aisée si son implémentation est en C++.

Développer des programmes en langage C++ semble plus intéressant que de développer en langage C. La lecture des algorithmes est plus claire parce que ces derniers sont plus concis (héritage et surchage des opérateurs). De plus, la compilation de programmes C++ est moins permissive ce qui enlève certaines erreurs de programmation.

CONCLUSION.

Notre objectif a été tout au long de ce mémoire de proposer une méthode d'opérations booléennes qui permette de résoudre des problèmes courants liés à ce type d'algorithme (chapitre un) : des temps d'exécution trop importants pour de plus arriver à un résultat erronné, une gestion de cas particuliers trop nombreux et des difficultés à s'adapter à d'autres familles d'objets que les polyèdres. Le but de l'amélioration de ces problèmes est de pouvoir, à terme, obtenir un modèle hybride basé sur un arbre CSG.

La méthode que nous avons proposée, dans le chapitre deux, pour résoudre ces problèmes sur les objets polyédriques, a exigé un effort de formalisation pour arriver à un algorithme robuste et plus fiable (limitant, ainsi, les cas où les opérations booléennes échouent). Cette formalisation nous a aussi permis de réduire considérablement les cas particuliers en ne traitant que le cas de contact entre les frontières des deux objets.

De plus, l'intérêt de la méthode est de résoudre un problème tridimensionnel par un problème bidimensionnel basé sur les faces des deux objets. Les objets non-Eulériens pouvant comporter des parties sans volume (i.e. des faces isolées), leur prise en compte, dans le futur, en sera donc facilitée.

Le troisième chapitre montre que la solution la plus adéquate pour adapter la méthode que nous proposons aux objets à faces quelconques est un pré-traitement et un post-traitement des objets, afin d'utiliser sans changement les opérations booléennes sur les polyèdres. Le pré-traitement consiste à approcher les faces non planes de l'objet par des facettes et la phase de post-traitement consiste à supprimer les facettes d'une même surface.

La maquette que nous avons réalisée comporte seulement deux types de quadriques, à savoir la sphère et le cylindre. Une de nos préoccupations futures sera de permettre la construction d'objets par opérations booléennes avec des surfaces plus complexes que des quadriques (surfaces paramétriques).

Le quatrième chapitre a permis d'illustrer l'important développement algorithmique nécessaire à la mise en œuvre des différents concepts présentés dans les chapitres deux et trois ainsi que les différents problèmes rencontrés. Les difficultés mentionnées dans ce chapitre, sont de deux sortes :

- d'une part, liées à la gestion des erreurs par un "epsilon" théorique. La gestion d'erreurs est un thème actuellement en cours de traitement dans notre laboratoire [GAP 95c] et nous pourrons modifier notre algorithme pour intégrer les nouveaux concepts issus de cette recherche,

- d'autre part, liées aux temps de calcul. Nous avons montré, dans ce chapitre, qu'il était possible de modifier l'algorithme général pour en accélérer le traitement. De plus, une étude précise sur l'utilisation du parallélisme pourrait peut-être permettre d'améliorer considérablement les temps d'exécution. En particulier, la structure de notre programme s'adapte aux architectures massivement parallèles orientées S.I.M.D. (Single Instruction Multiple Data). En effet, le traitement des faces du premier objet opérande se fait indépendamment du second objet opérande (et réciproquement). En outre, chaque traitement d'une face d'un des deux objets ne dépend d'aucune autre face. Ainsi, chaque processus gère les opérations d'une face d'un des objets en consultant les faces de l'autre objet se trouvant en mémoire principale.

Enfin, l'ensemble des procédures et fonctions de cette implémentation vont nous servir à la conception et la réalisation d'un modèle hybride. En effet, l'évaluation des opérations booléennes permet le passage du modèle CSG (dans lequel les opérations booléennes sont implicites) au modèle B-Rep (qui représente la peau de l'objet). Cependant, il est préférable d'avoir un modèle hybride plutôt que deux représentations pour un même objet. C'est pourquoi, le nouveau modèle géométrique tridimensionnel REGAIN [GAR 94] de notre laboratoire pourra être un modèle CSG dans lequel des informations supplémentaires sur les faces de l'objet seront ajoutées en chaque nœud. Ces informations pourront être :

- les faces des deux sous-objets fils éliminées par l'application de l'opération booléenne,
- les faces des deux sous-objets fils inchangées après l'opération booléenne,
- les parties des faces restantes (i.e. ni inchangées, ni éliminées par l'opération booléenne) qui appartiennent à la frontière de l'objet et dont le contour doit alors être explicitement spécifié.

En parcourant l'arbre de construction, nous obtiendrons ainsi la frontière de l'objet représenté par la racine de l'arbre.

RÉFÉRENCES BIBLIOGRAPHIQUES.

RÉFÉRENCES BIBLIOGRAPHIQUES.

[AYA 88] D. Ayala
Boolean operations between solids and surfaces by octrees : models and algorithm.
Computer aided design - volume 20 - n° 8 - p. 452/465 - 1988

[AZB 90] N.M. Aziz, S. Bhat
Bezier surface/surface intersection.
IEEE computer graphics and applications - p.50/58 - Janvier 1990

[BAH 88] C.L. Bajaj, C.M. Hoffmann, R.E. Lynch, J.E.H. Hopcroft
Tracing surface intersections.
Computer aided geometric design - volume 5 - p.285/307 - 1988

[BEN 93] M. O. Benouamer
Opérations booléennes sur les polyèdres représentés par leurs frontières et imprécisions numériques.
Thèse de doctorat en informatique - Ecole Nationale Supérieure des Mines de Saint Etienne - Juillet 1993

[BEM 93] M. Benouamer, D. Michelucci, B. Peroche
Error-free boundary evaluation using lazy rational arithmetic : a detailed implementation.
ACM solid modeling - volume 5 - p. 115/126 - 1993

[BEG 95] M. Berroug, Y. Gardan, M. Sahnoune
Intersection de deux surfaces paramétriques : méthode de subdivision récursive.
Rapport de recherches - Laboratoire de Recherches en Informatique de Metz - n°95-06 - Juin 1995

[BMP 91] E. Boudinet, B. Martin, E. Perrin
Rapport de maîtrise informatique
Projet de C.A.O. : opérations booléennes en deux dimensions.
Mars 1991

[BOO 91] G. Booch
Object oriented design with applications.
The Benjamin / Cummings publishing company Inc.

[BRA 75] I.C. Braid
The synthesis of solid bounded by many faces.
Communications of ACM - volume 18 - n° 4 - 1975

[BRO 88] W.F. Bronsvoort
Boundary evaluation and direct display of CSG models.
Computer aided design - volume 20 - n° 7 - Septembre 1988

[CAB 89] M.S. Casale, J.E. Bobrow
A set operation algorithm for sculptured solids modeled with trimmed patches.
Computer aided geometric design - volume 6 - p. 235/247 - 1989

[CAS 87] M.S. Casale
Free form solid modeling with trimmed patches.
IEEE computer graphics and applications - p. 33/43 - Janvier 1987

[DES 92] H. Desaulniers, N.F. Stewart
An extension of manifold boundary representations to the r-sets.
ACM transactions on graphics - volume 11 - n° 1 - p. 40/60 - 1992

[ELV 94] W.H. Elmaraghy, S.R. Valluri, B.M. Skubnik, P.D. Surry
Intersection volumes and surface areas of cylinders for geometrical modelling and tolerancing.
Computer aided design - volume 26 - n° 1 - p. 29/45 - 1994

[FAR 87] R.T. Farouki
Trimmed-surface algorithms for the evaluation and interrogation of solid boundary representations.
IBM J. research and developpement - volume 31 - n° 3 - 1987

[FAR 87b] R.T. Farouki
Direct surface section evaluation.
Geometric modeling : algorithms and new trends by G.E. Farin
1987 SIAM p. 319-334

[GAJ 88] Y. Gardan, J.P. Jung, J.N. Kolopp, C. Minich, W.Totino
Une approche nouvelle de la convivialité dans un système de CAO : les principes de dialogue dans SACADO.
Actes de la 7ème conférence européenne sur la CFAO et l'infographie - MICAD 1988

[GAP 93] Y. Gardan, E. Perrin
Une nouvelle approche pour la mise en œuvre d'opérations booléennes sur des modèles par les frontières.
Rapport de recherches - Laboratoire de recherches en informatique de Metz - Juin 1993

[GAP 94] Y. Gardan, E. Perrin
Un nouvel algorithme pour les opérations booléennes sur des solides par les frontières.
Revue Internationale de CFAO et d'infographie - Actes de Micad - volume 9 - n° 1/2 - 1994

[GAP 95] Y. Gardan, E.Perrin
An algorithm reducing 3D boolean operations to 2D problem : concepts and results.
Accepté et à paraître dans Computer Aided Design.

[GAP 95b] Y. Gardan, E. Perrin
Une nouvelle approche pour la mise en œuvre des opérations booléennes sur des objets à surfaces quelconques.
Rapport de recherches - Laboratoire de recherches en informatique de Metz - n° 95-04 - Juin 1995

[GAP 95c] Y. Gardan, D. Pirus
Imprécisions numériques en CAO : étude de la propagation des erreurs.
Rapport de recherches - Laboratoire de recherches en informatique de Metz - n° 95-09 - Septembre 1995

[GAR 83] Y. GARDAN
Systèmes de C.F.A.O.
Hermès - 1983

[GAR 94] Y. Gardan
REGAIN : an adaptable CAD system, using a novel man-machine interface and generic applications models.
Deuxième congrès Franco-Japonais de mécatronique - TAKAMATSU - Japon - octobre 1994

[GCP 91] E.L. Gursoz, Y. Choi, F.B. Prinz
Boolean set operations on non-manifold boundary representation objects.
Computer aided design - volume 23 - n° 1 - p. 33/39 - 1991

[GEO] P.L. George
Module F : génération automatique de maillage.
INRIA collection didactique

[HIG 93] M. Higashi
High-quality solid-modelling system with free-form surfaces.
Computer aided design - volume 25 - n° 3 - p. 172/183 - 1993

[HOE 85] E.G. Houghton, R.F. Emnett, J.D. Factor, C.L. Sabharval
Implementation of a divide and conquer method for intersection of parametric surfaces.
Computer aided geometric design - volume 2 - 1985 - p.173/183

[HOF 89] C.M. Hoffmann, J.E. Hopcroft, M.S. Karasick
Robust set operations on polyhedral solids.
IEEE computer graphics and appications - volume 9 - Novembre 1989

[JAI 93] P. Jaillon
Proposition d'une arithmétique rationnelle paresseuse et d'un outil d'aide à la saisie d'objets en synthèse d'images.
Thèse de doctorat en informatique - Ecole Nationale Supérieure des Mines de Saint Etienne - Septembre 1993

[JOH 93] J.K. Johnstone
A new intersection algorithm for cyclides and swept surfaces unsing circle decomposition.
Computer aided geometric design - volume 10 - p. 1/24 - 1993

[KIY 92] K. Kitajima, M. Yamaguchi
A shell oriented boolean set operations algorithm suited for the B-Reps based on boundary edge loops.
Systems and computers in Japan - volume 23 - n° 6 - 1992

[LTH 86] D.H. Laidlaw, W.B. Trumbore, J.F. Hugues
Constructive solid geometry for polyhedral objects.
Proc. SIGGRAPH'86 - ACM computer graphics - volume 20 - n° 4 - 1986

[LUK 89] G. Lukacs
The generalised inverse matrix and the surface-surface intersection problem.
dans "Theory and practice of geometric modelig Springer Verlag" USA 1989 - W. Strasser et Seidel - p. 167/185

[MAN 82] M. Mantyla
An inversion algorithm for geometric models
Computer graphics - volume 16 - n° 3 - 1982

[MAT 83] M. Mantyla, M. Tamminen
Localized set operations for solid modelling
Computer graphics - volume 17 - n° 3 - 1983

[MAT 88] D. Ma, R. Tang
Realizing the boolean operations in solid modeling technique via directed loops.
Computer and graphics - volume 12 - n° 3/4 - 1988

[MAS 93] H. Masuda
Topological operators and boolean operations for complex-based nonmanifold geometric models.
Computer aided design - volume 25 - n° 2 - p. 119/129 - 1993

[MIC 87] D. Michelucci
Les représentations par les frontières : quelques constructions; difficultés rencontrées.
Thèse de doctorat en informatique - Ecole Nationale Supérieure des Mines de Saint Etienne - Novembre 1987

[MIC 92] D. Michel
Contribution à la conception, la mise en œuvre et l'amélioration des algorithmes de calcul des intersections de carreaux NURBS.
Thèse de doctorat en informatique - Université de Metz - Mars 1992

[MIL 87] J.R. Miller
Geometric approaches to nonplanar quadric surface intersection curves.
ACM transactions on graphics - volume 6 - n° 4 - p. 274/307 - 1987

[MIL 93] J.R. Miller
Incremental boundary evaluation using inference of edge classifications.
IEEE computer graphics and applications - p. 71/78 - Janvier 1993

[MMP 87a] P. Martin, D. Martin
Les algorithmes de calcul de l'intersection de solides définis par leur bord.
Revue de CFAO et d'infographie - volume 2 - n° 2 - 1987

[MMP 87b] P. Martin, D. Martin
Les algorithmes de calcul de l'intersection de solides définis par leur bord.
Revue de CFAO et d'infographie - volume 3 - n° 2 - 1987

[NAB 86] I. Navazo, D. Ayala, P. Brunet
A geometric modeler based on the exact octtree representation of polyhedra.
Computer graphics Forum - volume 5 - p. 91/104 - 1986

[NFB 87] I. Navazo, J. Fontdecaba, P. Brunet
Extended octree, between CSG trees and Boundary representations.
Eurographics'87 - p. 239/247 - 1987

[PAT 93] N.M. Patrikalakis
Surface-to-surface intersections.
IEEE computer graphics and applications - volume 13 - n°1 - p. 89-95 - Janvier 1993

[PER 92] E. Perrin
Une nouvelle approche pour les opérations booléennes sur les solides.
Rapport de stage en vue de l'obtention du D.E.A. d'informatique de l'université de Nancy 1 - Septembre 1992

[PES 89] R. Perruchio, M. Saxena, A. Kela
Automatic mesh generation from solid models based on recursive spatial decompositions.
International journal for numerical methods in Engineering - volume 28 - 1989

[PIL 89] M. Pilz, H.A. Kamel
Creation and boundary evaluation of CSG-models.
Engineering with computers - volume 5 - p. 105/118 - 1989

[PLA 93] N. Pla-Garcia
Boolean operations and spatial complexity of face octrees.
Eurographics'93 - volume 12 - n° 3 - p. 153/164 - 1993

[RET 78] A.A.G. Requicha, R.B. Tilove
Mathematical foundations of constructive solid geometry : general topology of regular closed sets.
Tech. Memo. 27, Production Automation Project, Univ. Rochester, Rochester, N.Y., Mars 1978

[REV 85] A.A.G. Requicha, H.B. Voelcker
Boolean operations in solid modeling : boundary evaluation and merging algorithms.
Proceeding of the IEEE - volume 13 - n°1 - p. 30/44 - Janvier 1985

[ROR 91] J.R. Rossignac, A.A.G. Requicha
Constructive non-regularized geometry
Computer aided design - volume 23 - n° 1 - p. 21/32 - 1991

[SAP 90] G. Saporta
Probabilités, analyse des données et statistiques.
Editions Technip - 1990

[SHV 91] V. Shapiro, D.L. Vossler
Construction and optimieation of CSG representations.
Computer aided design - volume 23 - n° 1 - p. 1/20 - 1991

[SHV 93] V. Shapiro, D.L. Vossler
Separation for boundary to CSG conversion.
ACM transactions on graphics - volume 12 - n° 1 - p. 35/55 - 1993

[TIL 80] R.B. Tilove
Set membership classification : a unified approach to geometric intersection problems.
IEEE transactions on computers - volume 29 - n°10 - p. 874/883 - Octobre 1980

[TOT 91] H. Toriya, T. Takamura, T. Satoh, H. Chiyokura
Boolean operations for solids with free-form surfaces through polyhedral approximation.
The visual computer - volume 7 - n° 2/3 - 1991

[WAL 89] M. Walker
Boolean operations with enriched octtree structures.
Computer and graphics - volume 13 - n° 4 - p. 487/495 - 1989

[WEA 77] K. Weiler, P. Atherton
Hidden surface removal using polygon area sorting.
Computer and graphics - volume 11 - n°2 - 1977

[ZHA 92] Y. Zhao
Opérations booléennes sur les solides eulériens définis par NURBS utilisant les informations de proximité topologique.
Thèse de doctorat en informatique - Ecole Centrale de Paris - Janvier 1992

INDEX

INDEX.

ANNEXE A.

ANNEXE A.

Les algorithmes ci-dessous sont les algorithmes des procédures et fonctions utilisés dans l'algorithme de la procédure de recherche de section :

```
Procédure AjouterPoint (Entrée : P , face1 , face2 , face , solide ; Entrée-Sortie : liste)
{ P correspond aux coordonnées du point à ajouter dans la liste }
{ face1 , face2 et face correspondent à la topologie du point }
{ liste correspond à la liste de points d'intersection construite par la procédure, elle est de type ListePoints }

Variables locales : Topologie ;
Début
    Topologie ← ConsTypeTopologie (T_Courbe , NonRelié , face , 0 , 0 , 0 , 0 , 0 , 0 , 0 , FALSE , Cons
    (face1 , Cons (face2 , Vide))) ;
    NumeroPoint ← NumeroPoint + 1 ;
    Point ← ConsTypePoint (NumeroPoint , solide , P , Cons (Topologie , Vide) , FALSE) ;
    liste ← Cons (Point , liste) ;
Fin

Procédure IntersectionPlanArêtes (Entrée : face , solide , matrice ; Sortie : liste)
{ face correspond au numéro de la face traitée }
{ solide correspond au numéro de solide sur lequel est recherchée la section }
{ matrice correspond à la matrice de passage du repère du monde au repère de la face }
{ liste correspond à la liste de points d'intersection construite par la procédure, elle est }
{ de type ListePoints }

Variables locales : Plan , ListeArêtes , Origine , Extremite , PointI , Res , Arête , face1 , face2 ;

Début
    Plan ← MDPlan (face) ;
    ListeArêtes ← MDArêtesObjet (solide) ;
    Tant que (Non (Vide (ListeArêtes))
    Faire   Arête ← Tete (ListeArêtes) ;
            MDConsArête (Arête , Origine , Extremite) ;
            IntersectionSegmentPlan (Origine , Extremite , Plan , Res , PointI) ;
            { On conserve la topologie du point d'intersection }
            MDConsFacesArête (Arête , face1 , face2) ;
            Cas (Res) Suivant
            0 :   {L'arête est dans le plan d'intersection}
                  AjouterPoint (Extremite*matrice , face1 , face2 , face , solide , liste) ;
                  AjouterPoint (Origine*matrice , face1 , face2 , face , solide , liste) ;
            1 :   {L'origine de l'arête est le point d'intersection}
                  AjouterPoint (Origine*matrice , face1 , face2 , face , solide , liste) ;
            2 :   {L'extrémité de l'arête est le point d'intersection}
                  AjouterPoint (Extremite*matrice , face1 , face2 , face , solide , liste) ;
            3 :   {PointI est le point d'intersection}
                  AjouterPoint (PointI*matrice , face1 , face2 , face , solide , liste) ;
            FinCas
            ListeArêtes ← Suivant (ListeArêtes) ;
    FinTantQue
Fin
```

```
Procédure RelierPoints (Entrée : liste ; Sortie : section)
{ liste correspond à la liste de points d'intersection }
{ section correspond à la liste de polygones associée à la section }

Variables locales : Changement , Nouveau ;
Début
    section = Vide ;
    Si      (Non (Vide (liste)))
    Alors   { On construit l'ensemble des contours à partir des points d'intersection }
            ConstruitCycle (liste , section) ;
            { On construit l'ensemble des contours extérieurs et leurs contours intérieurs }
            Changement = Non (Vide (section)) ;
            Nouveau = Vide ;
            Tant que (Changement)
            Faire   ImbriquationPolygone (section , Nouveau , Changement) ;
                    section = Nouveau ;
                    Nouveau = Vide ;
            FinTantQue
    FinSi
Fin

Procédure ImbriquationPolygone (Entrée : section ; Sortie : polygone , changement)
{ Cette procédure prend tous les polygones de section un à un et les compare }
{ suivant l'algorithme décrit dans la section 2.3 du chapitre 2 }
```

```
Procédure ConstruitCycle (Entrée : liste ; Sortie : section)
{ liste correspond à la liste de points d'intersection }
{ section correspond à la liste de polygones associée à la section }

Variables locales : Cycle , Contour , Numero , NumeroSuivant , Endroit ;
Début
    Cycle ← Vide ;
    {On recherche les mêmes points d'intersection et on rectifie pour chaque point sa }
    {topologie}
    liste ← TrouveMêmesPoints (liste) ;
    Si      (Longueur (liste) <= 2)
    Alors   section ← vide ;
    Sinon   {On aligne et ordonne les points qui proviennent d'une même face}
            Alignement (liste , Cycle) ;
            {On segmente chaque droite en segments intérieurs à la face}
            Cycle ← Segmenter (Cycle) ;
            {On construit chaque contour en suivant les segments sur chaque face}
            Endroit ← Vide ;
            Tant que PossibleConstruireCycle (Cycle , Endroit)
            Faire   Contour ← Vide ;
                    {Numéros de la face courante et de la suivante à parcourir}
                    Numero ← 0 ; NumeroSuivant ← 0 ;
                    {On insere le premier segment dans le contour et on initialise les}
                    {numeros de face}
                    DébutCycle (Endroit , Numero , NumeroSuivant , Contour ) ;
                    {On construit le contour jusqu'à arriver au même point de départ}
                    Tant que (NumeroSuivant <> Numero)
                    Faire   ContinueCycle (NumeroSuivant , Contour , Cycle) ;
                    FinTantQue
                    section ← Cons (Contour , section) ;
            FinTantQue
    FinSi
Fin
```

Nous donnons ci-après l'en-tête de la procédure permettant d'effectuer une opération booléenne en deux dimensions sur deux polygones.

```
Procédure Traitement2D (Entrée : polygone1 , polygone2 , ope2D ; Sortie : polygone)
{polygone1 correspond au premier polygone opérande}
{polygone2 correspond au second polygone opérande}
{ope2D correspond à l'opération booléenne en deux dimensions choisie}
{polygone est la liste des polygones résultats}
```

Nous donnons ci-après les algorithmes des procédures présentes dans le corps de la procédure PointInterRelieAvecInter de la section 2.3.4 du chapitre 4.

```
Procédure RecherchePointInter (Entrée : polygone , arete , face , numero)
{polygone correspond à la liste des polygones résultats de la face traitée}
{arete est le numero de l'arete sur lequel se situe l'intersection recherchée}
{face est le numero de la face ayant donnée le segment de l'intersection recherchée}
{numero est le numero de point que l'on doit attribuer au point recherché}

Variables locales : Trouve , Contour , Trous ;

Début
    Trouve = Faux ;
    Tant que (Non (Trouve) et Non (Vide (polygone)))
    Faire   Contour = ContourExterieur (polygone) ;
            Trous = ContoursInterieurs (polygone) ;
            AttribueNumeroPointInter (Contour , arete , face , numero , Trouve) ;
            Tant que (Non (Trouve) et Non (Vide (Trous)))
            Faire   AttribueNumeroPointInter (Tete (Trous) , arete , face , numero , Trouve) ;
                    Trous = Suivant (Trous) ;
            FinTantQue
            polygone = Suivant (polygone) ;
    FinTantQue
Fin


Procédure AttribueNuméroPointInter (Entrée : liste , arete , face , numero ; Entrée-Sortie : trouve)
{liste correspond à la liste de points sur lesquels s'effectue la recherche}
{arete est le numero de l'arete sur lequel se situe l'intersection recherchée}
{face est le numero de la face ayant donnée le segment de l'intersection recherchée}
{numero est le numero de point que l'on doit attribuer au point recherché}
{trouve devient vrai si le point recherché est trouvé}

Variables locales : Topo , A1 , Fa ;

Début
    Tant que (Non (trouve) et Non (Vide (liste)))
    Faire   Topo ← Tete (TopologiePoint (Tete (liste))) ;
            Si      (Type (Topo) = T_Inter)
            Alors   {On cherche l'arête de l'intersection}
                    A1 ← PremiereArete (Topo) ;
                    {On cherche le numéro de la face ayant donnée le segment de l'intersection}
                    Fa ← Tete (Droites (Topo)) ;
                    Si      ((A1 = arete) et (Fa = face))
                    Alors   trouve ← vrai
                            liste → Point.Numero ← i ;
                            liste → Point.ListeTopo → Topologie.Type ← T_Termine ;
                    FinSi
            FinSi
            liste ← Suivant (liste) ;
    FinTantQue
Fin
```

La procédure RecherchePointSection correspond à la procédure RecherchePointInter sauf que la recherche porte sur les différentes faces ayant donné le point de la section.

ANNEXE B.

ANNEXE B.

Nous donnons ci-dessous l'ensemble des définitions de la classe (et de ses sous-classes) la plus importante du modèle pour les objets à surfaces quelconques.

```
/*-------------------------------------------------------*/
/* Définitions de la classe face                         */
/*-------------------------------------------------------*/
class Face
{
    protected :
        int             obj ;       /* référence au numero de l'objet          */
        ListeContour    LC ;        /* liste des arêtes des contours troués de la surface */
    public :
            /* Fonctions de topologie pour la consultation et la création d'une face dans le
            modèle*/
            ListeContour    DonneAretes ();
            void            AjouterNumeroObjet (int);
            int             DonneObjet ();
        virtual void        AjouterSurfaceFacePremier (int , Arete*) = 0;
        virtual void        AjouterSurfaceFaceDeuxieme (int , Arete*) = 0;
            /* Affichage des caractéristiques d'une surface */
        virtual void        Affiche () = 0;
            /* Sauvegarde des caractéristiques d'une surface dans un fichier */
        virtual void        Sauvegarde (ostream&) = 0;
            /* Initialisation d'une surface à partir d'un fichier */
        virtual void        Charge (istream&) = 0;
            /* Donne le point d'intersection entre la surface et la droite */
        virtual int         IntersectionRayon (Droite , Intersection&) = 0 ;
            /* Vrai si le point appartient à la surface */
        virtual int         PointAppartient (Point) = 0         ;
        virtual int         PointAppartient (Point , Contour) = 0  ;
            /* Calcule le coin bas gauche et haut droit de l'englobant parallélèpipèdique de la
            surface */
        virtual void        Englobant (Point & , Point &) = 0  ;
            /* Calcule la normale en un point de la surface */
        virtual Vecteur     Normale (Point) = 0;
            /* Translate la surface d'un vecteur */
        virtual void        Translater (Vecteur) = 0;
} ;
```

```
class Sphere : public Quadrique
{
    Point    Centre ;
    double   Rayon ;
    int      IntersectionRayonMonde (Droite , double & , double &);

    public :        Sphere ();          /* Centre, Rayon, Sens , Liste Aretes , Liste Facettes */
                    Sphere (Point , double , int , ListeContour , ListeEntier);
             Vecteur Normale (Point);
             void   Affiche ();
             void   Sauvegarde (ostream&) ;
             void   Charge (istream&);
             void   TriangulerEntierement (int , const int , int) ;
             void   Englobant (Point & , Point &);
             Point  Surelever (Point , Plan);
             void   Translater (Vecteur);
             Point  Aleatoire (double , double);
             Point  CentreSurface ();
} ;

class Cylindre : public Quadrique
{
    Point    CentreBase;
    Vecteur  Directeur;
    double   Rayon;
    double   Hauteur;
    int      IntersectionRayonMonde (Droite , double & , double &);
    int      IntersectionRayonUnitaire (Droite , double & , double &);
    void     MiseAJourMatrices ();

    public : Cylindre ()                ;
             /* Centre de la base , Vecteur Directeur , Rayon , Hauteur , Sens , Liste Aretes , Liste
             Facettes */
             Cylindre (Point , Vecteur , double , double , int , ListeContour , ListeEntier);
    Vecteur  Normale (Point);
    void     Affiche ();
    void     Sauvegarde (ostream&);
    void     Charge (istream&);
    void     TriangulerEntierement (int , const int , int);
    void     Englobant (Point & , Point &);
    Point    Surelever (Point , Plan);
    void     Translater (Vecteur);
    Point    CentreSurface ();
} ;
```

A partir de l'ensemble des classes définies pour le modèle (voir section 3.1 du chapitre 4), voici l'ensemble des fonctions disponibles dans le modèle :

```
/*-------------------------------------------------------------------------*/
/* Fonctions de CREATION                                                   */
/*-------------------------------------------------------------------------*/

/*********************************/
/* Creation d'un point dans le modele */
/*********************************/
int MDCreerPoint (    double x , double y , double z ) ;
```

```
class Facette : public Face
{
    Plan P ;          /* Plan de la face                              */
    Matrice M ;       /* Matrice de projection sur le plan de la face  */
    Matrice Inv ;     /* Matrice inverse de la projection sur le plan de la face */

    public :                  Facette ();
                              Facette (Contour);
                              Facette (Plan , Contour);
              Plan            DonnePlan ();
              Matrice         Projection ();
              Matrice         ProjectionInverse ();
              /* fonctions de topologie */
              void            AjouterNumeroSurface (int);
              void            AjouterSurfaceFacePremier (int , Arete*);
              void            AjouterSurfaceFaceDeuxieme (int , Arete*);
              /* autres fonctions */
              void            Affiche ();
              void            Sauvegarde (ostream&);
              void            Charge (istream&);
              int             IntersectionRayon (Droite , Intersection&);
              int             PointAppartient (Point);
              int             PointAppartient (Point , Contour)  ;
              void            Englobant (Point & , Point &);
              Vecteur         Normale (Point);
              void            Translater (Vecteur);
} ;

class Quadrique : public Face
{
    protected :
    ListeEntier     LF ;          /* Liste des numeros des facettes de la surface gauche  */
    Matrice         LocalMonde;   /* Matrice de passage du repere local au repere du monde */
    Matrice         MondeLocal;   /* Matrice de passage du repere du monde au repere local */
    int             Positif;      /* Vrai si la surface est prise en positif  */
    int             IntersectionRayonEntier (Droite , Intersection&)    ;
    virtual int     IntersectionRayonMonde (Droite , double & , double &) = 0   ;

    public :        /* Liste des facettes de la surface si elle est facettisée */
                    ListeEntier     DonneFacettes () ;
                    /* Sens de l'utilisation de la quadrique, en différence ou pas */
                    int             Sens ();
                    /* Matrices de passages des deux repères Monde et Local */
                    Matrice         PassageLocalMonde ();
                    Matrice         PassageMondeLocal ();
                    /* Fonctions de topologie */
                    void            AjouterSurfaceFacePremier (int , Arete*);
                    void            AjouterSurfaceFaceDeuxieme (int , Arete*);
                    void            AjouterNumeroFacette (int);
                    /* Autres fonctions */
                    void            SupprimerFacettes (int);
                    int             IntersectionRayon (Droite , Intersection&);
                    int             PointAppartient (Point);
                    int             PointAppartient (Point , Contour);
                    void            Trianguler (int , int , int);
} ;
```

```
/*************************************************************************/
/* Creation d'une arete dans le modele connaissant son point origine et son point extremite */
/*************************************************************************/
int MDCreerArete (int P1 , int P2 );

/*************************************************/
/* Creation d'une face dans le modele en spécifiant trois points */
/*************************************************/
int MDCreerFacette (int P1, int P2 , int P3 , Contour C);

/*****************************/
/* Creation d'une face dans le modele */
/*****************************/
int MDCreerFacette (Contour C) ;

/*****************************************************************/
/* Creation d'une portion de sphere, la portion pouvant etre la sphere elle-meme */
/* LA : Liste des Aretes LF : Liste des Facettes */
/*****************************************************************/
int MDCreerPortionSphere (Point P ,double R , int SensPlus , ListeContour LA, ListeEntier LF );

/*****************************************************************/
/* Creation d'une portion de cylindre , la portion pouvant etre le cylindre entier */
/* LA : Liste des Aretes LF : Liste des Facettes */
/*****************************************************************/
int MDCreerPortionCylindre ( Point Base ,Vecteur V ,double Ray ,double H ,int SensPlus ,
                             ListeContour LA , ListeEntier LF ) ;

/*************************************************/
/* Creation d'un objet connaissant la liste des surfaces le composant */
/*************************************************/
int MDCreerObjet (ListeEntier LS);

/*************************************************************************************/
/* Cree un objet qui contient la liste de vraies faces planes LF, comme liste de facettes LFS s'appuyant sur */
/* les surfaces deja existantes dans le modele LS */
/*************************************************************************************/
int MDCreerObjetFacettise ( ListeEntier LF, ListeDeListeEntier LS, ListeDeListeEntier LFS ) ;

/*********************************/
/* Ajoute la facette NF sur la surface NS */
/*********************************/
int MDAjouterFacette (int P1 ,int P2 ,Contour L ,int NS ) ;

/*********************************/
/* Ajoute une nouvelle face au solide NO */
/*********************************/
int MDAjouterFacette (Contour C , int NO ) ;

/*-----------------------------------------------------------------------*/
/* Fonctions de CONSULTATION */
/*-----------------------------------------------------------------------*/

/*********************************************************************/
/* Procedure donnant les coordonnees du point connaissant son numero dans le modele */
/*********************************************************************/
Point MDConsPoint ( int num );
```

```
/*********************************/
/* Origine de l'arete num dans le modele */
/*********************************/
int MDConsOrigine ( int num) ;

/*********************************/
/* Extremite de l'arete num dans le modele */
/*********************************/
int MDConsExtremite ( int num) ;

/*************************************/
/* Renvoie l'arete suivante de na sur la face nf  */
/*************************************/
int MDAreteSuivante ( int na , int nf ) ;

/*************************************/
/* Renvoie l'arete precedente de na sur la face nf */
/*************************************/
int MDAretePrecedente (int na , int nf ) ;

/*******************/
/* Vrai si l'arete est vive */
/*******************/
int MDAreteVive (int NA) ;

/*****************************************/
/* Donne la liste de points associee a la liste d'aretes */
/*****************************************/
void MDListeAretesListePoints (ListeEntier LA   , ListeEntier & LP ) ;

/*********************************/
/* Renvoie les faces encadrant l'arete*/
/*********************************/
void MDConsFacesArete (int na , int & ns1 , int & ns2 ) ;

/*************************************/
/* Renvoie la liste des contours sur la face */
/*************************************/
ListeContour MDConsContoursFace (int num) ;

/*************************/
/* Renvoie le plan de la facette num */
/*************************/
int MDConsPlanFacette (int num , Plan & P) ;

/*************************************************/
/* Renvoie vrai si la facette presumee NF appartient a une quadrique */
/*************************************************/
int MDFacetteDansSurface (int NF) ;

/***********************/
/* Donne la surface de la facette */
/***********************/
int MDFaceFacette (int NF) ;

/*********************************************************/
/* Renvoie la liste des numeros de faces de l'objet num dans le modele */
```

```
/*********************************************************/
ListeEntier MDConsFacesObjet (int num) ;

/*****************************************/
/* Donne la liste des numeros des aretes du solide num */
/*****************************************/
ListeEntier MDConsAretesObjet (int num) ;

/*********************************************************/
/* Renvoie la liste des numeros de facettes de l'objet num dans le modele */
/*********************************************************/
ListeEntier MDConsFacettesObjet (int num) ;

/************************************/
/* Donne la liste de tous les objets du modele */
/************************************/
ListeEntier MDConsTousLesObjets () ;

/************************************/
/* Donne la liste de toutes les aretes du modele */
/************************************/
ListeEntier MDConsToutesLesAretes () ;

/*-------------------------------------------------------------------------*/
/* Fonctions de DESTRUCTION                                                */
/*-------------------------------------------------------------------------*/

/*********************************/
/* Destruction d'un point dans le modele */
/*********************************/
void MDDetruirePoint (int Num) ;

/*********************************/
/* Destruction d'une arete dans le modele */
/*********************************/
void MDDetruireArete (int Num) ;

/*********************************/
/* Destruction d'une face dans le modele */
/*********************************/
void MDDetruireFace (int Num) ;

/*********************************/
/* Destruction d'un objet dans le modele */
/*********************************/
void MDDetruireObjet (int Num) ;

/*-------------------------------------------------------------------------*/
/* SAUVEGARDE                                                              */
/*-------------------------------------------------------------------------*/

/********************/
/* Sauvegarde du modele */
/********************/
void MDSauvegarde (char * Nom);

/*-------------------------------------------------------------------------*/
/* CHARGEMENT                                                              */
```

```
/*-------------------------------------------------------------------------*/

/**********************/
/* Chargement d'un modele */
/**********************/
void MDCharge (char * Nom);

/*-------------------------------------------------------------------------*/
/* PROCEDURES AYANT ACCES AU MODELE                */
/*-------------------------------------------------------------------------*/

/*****************************************************/
/* Cacule la premiere intersection visible entre le modele et un rayon */
/*****************************************************/
int MDIntersectionRayon (Droite Rayon , Intersection & I );

/*****************************************************************/
/* Renvoie vrai si le point P appartient a la face de numero num dans le modele */
/*****************************************************************/
int MDPointAppartientFace (int num , Point P ) ;

/*****************************/
/* Renvoie l'arete la plus proche de P  */
/*****************************/
void MDAreteLaPlusProche (   ListeEntier L , Point P , double & Dist , int & arete);

/*************************************************************************/
/* Renvoie 0 si le point est exterieur , 1 si le point est interieur , 2 s'il est sur la frontière de l'objet */
/*************************************************************************/
int MDExterieurInterieurObjet (int num , Point P) ;

/************************/
/* Englobant parallepipedique */
/************************/
void MDEnglobantObjet ( int num , Point & BG , Point & HD ) ;

/*****************************************************/
/* Renvoie la liste des aretes communes entre les faces NS1 et NS2 */
/*****************************************************/
void MDFacesAdjacentes (      int NS1 , int NS2 , ListeEntier & L ) ;

/***************************************/
/* Fait la triangulation de l'objet num dans le modele */
/***************************************/
void MDTrianguler (int num) ;

/*********************************************/
/* Donne la face adjacente a la surface ns en l'arete na */
/*********************************************/
void MDFaceAdjacenteEnUneArete (int na  , int ns , int & adj ) ;

/*******************/
/* Union de deux objets */
/*******************/
void MDUnion3D (int NO1 , int NO2 , ListeEntier & LO3 ) ;

/************************/
/* Intersection de deux objets */
```

```
/************************/
void MDIntersection3D (int NO1 , int NO2 , ListeEntier & LO3 ) ;

/***************************************/
/* Difference de deux objets           */
/***************************************/
void MDDifference3D (int NO1 , int NO2 , ListeEntier & NO3 ) ;

/***************************************/
/* Translation de vecteur V de l'objet NO du modele */
/***************************************/
void MDTranslater (    int NO , Vecteur V ) ;

/***************************************/
/* Translation de vecteur V du point du modele */
/***************************************/
void MDTranslaterPoint (int NOP , Vecteur V ) ;

/*****************************************/
/* Supprime les facettes de la triangulation sur l'objet NO */
/*****************************************/
void MDSupprimerTriangulation (int NO) ;
```

L'ensemble des procédures et fonctions complètent la procédure de triangulation de la section 3.2 du chapitre 4.

```
Procédure PlusPetitAngle (Entrée : Lpt , S ; Sortie : Lieu , Angle , A1 , A2 , Pnouveau)
{Recherche le plus petit angle formé par deux arêtes consécutives de Lpt}
{Lpt : Liste des arêtes du contour}
{Lieu : pointeur sur la premiere arête où se forme le triangle}
{Angle : valeur numérique du plus petit angle trouvé}
{A1 : première arête sur laquelle va porter le triangle}
{A2 : deuxième arête sur laquelle va porter le triangle}
{Pnouveau : troisième point du triangle s'il est nécessaire de la calculer, cas 2 et 3}

Variables locales : petitangle , lsauv , debut , lprec , ar1 , ar2 , P1 , P2 , P3 , possible , pnouv ;

Début
    Angle ← 2π ; petitangle ← 0 ; lsauv ← Lpt ; debut ← Vrai ; lprec ← Dernier (Lpt) ;
    Tant que (Non (Vide (Lpt)))
    Faire   {On prend les deux arêtes consécutives à partir de Lpt}
            ar1 ← PremiereArete (Lpt) ; ar2 ← DeuxiemeArete (Lpt , lsauv) ;
            {On prend les trois points consécutifs des deux arêtes}
            Point P1 ← MDConsPoint (MDConsOrigine (ar1)) ;
            Point P2 ← MDConsPoint (MDConsExtremite (ar1)) ;
            Point P3 ← MDConsPoint (MDConsExtremite (ar2)) ;
            {On calcule l'angle (P1,P2,P3)}
            petitangle ← CalculeAngle (P1 , P2 , P3) ;
            Si      (petitangle < Angle)
            Alors   possible ← Faux ;
                    {On teste s'il est possible de former le triangle, notamment test pour que les arêtes du
                    triangle ne coupent pas le contour actuel soumis à la triangulation}
                    Cas petitangle Suivant
                    <π/2:   possible ← TestPremierCas (P1 , P2 , P3 , lsauv , S) ;
                    <2π/3:  possible ← TestDeuxiemeCas (P1, P2 , P3 , lsauv , pnouv , S) ;
                    sinon:  possible ← TestTroisiemeCas (P1 , P2 , P3 , lsauv , pnouv , S) ;
                    FinCas
                    {Sauvegarde des données si le triangle est candidat}
                    Si      (possible)
                    Alors   Angle ← petitangle ; A1 ← ar1 ; A2 ← ar2 ;
                            Pnouveau ← pnouv ;
                            Lieu ← lprec ;
                    FinSi
            FinSi
            Si      (debut)
            Alors   debut ← Faux ;
                    lprec ← lsauv ;
            FinSi
            Lpt ← Suivant (Lpt) ;
    FinTantQue
Fin

Fonction CalculeAngle (Entrée : P1 , P2 , P3) : Réel ;
{Calcule l'angle formé par les trois points}
```

Fonction TestPremierCas (**Entrée** : P1 , P2 , P3 , lsauv , S) : **Booléen**
{Cherche si le triangle formé par les points P1, P2 , P3 ne coupe pas le contour lsauv sur S}

Les fonctions TestDeuxiemeCas et TestTroisiemeCas ont un rôle identique à celui de la fonction TestPremiereCas en y ajoutant un calcul de nouveau point sur la quadrique S.

Fonction TestDeuxiemeCas (**Entrée** : P1 , P2 , P3 , lsauv , S ; **Sortie** : Pnouv) : **Booléen**
{Calcule le point Pnouv sur la surface S puis teste si les triangles (P1 , P2 , Pnouv) et (Pnouv, P2 , P3) ne recoupent pas le contour lsauv sur S}

Fonction TestTroisiemeCas (**Entrée** : P1 , P2 , P3 , lsauv , S ; **Sortie** : Pnouv) : **Booléen**
{Calcule le point Pnouv sur la quadrique S puis teste si le triangle (P1 , P2 , Pnouv) ou si le triangle (P2 , P3 , Pnouv) doit être ajouté à la quadrique}

Procédure PremierCas (**Entrée** A1 , A2 , Num ; **Entrée-Sortie** : Lpt , Lieu)
{Création de la facette (A1 , A2 , -As) et suppression des arêtes A1 et A2 dans Lpt et Lieu}

Les procédures DeuxiemeCas et Troisieme cas ajoutent et suppriment les arêtes adéquates dans Lpt et Lieu afin de former les triangles correspondants à ces deux cas.

Résumé :

Notre objectif, tout au long de ce mémoire, est de proposer une méthode consacrée à l'évaluation des opérations booléennes sur deux objets B-Rep (Boundary Representation).

Le premier chapitre s'intéresse aux différentes méthodes existantes suivant les types d'objets auxquels les méthodes s'appliquent. Ce chapitre permet de souligner les problèmes inhérents aux algorithmes d'opérations booléennes, à savoir une certaine sensibilité aux erreurs de calcul, un temps d'exécution pouvant être important, un résultat parfois erroné, une gestion de trop nombreux cas particuliers et une difficulté à s'adapter à tous les types d'objets.

Le second chapitre propose une méthode pour traiter les opérations booléennes sur des objets à faces planes. L'intérêt principal de la méthode est de résoudre un problème tridimensionnel par un problème bidimensionnel basé sur les faces des deux objets. Les apports fondamentaux de cette méthode sont la formalisation rigoureuse des différents traitements à effectuer débouchant sur un algorithme robuste et fiable ainsi que le traitement sur les faces facilitant la prise en compte des objets non-Eulériens.

Le troisième chapitre montre que la solution la plus adéquate, pour adapter la méthode du chapitre deux, sur les objets à surfaces quelconques, consiste en un pré-traitement et un post-traitement des objets en vue d'utiliser la méthode sur les polyèdres sans changement.

Le quatrième chapitre permet de mettre en évidence l'important développement algorithmique nécessaire à la mise en œuvre des différents concepts présentés dans les chapitres deux et trois. Ce dernier chapitre met aussi l'accent sur la complexité de la méthode présentée.

Mots clés : **Modélisation, Opérations booléennes, Boundary Representation.**