Opérations booléennes sur les polyèdres représentés par leurs frontières et imprécisions numériques

Mohand Ourabah Benouamer

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

N° d'ordre **96 ID**

THESE

Présentée par

Mohand Ourabah BENOUAMER

pour obtenir le titre de

DOCTEUR
DE L'UNIVERSITE DE SAINT-ETIENNE
ET DE L'ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

(Spécialité : Informatique)

OPERATIONS BOOLEENNES SUR LES POLYEDRES REPRESENTES PAR LEURS FRONTIERES ET IMPRECISIONS NUMERIQUES

soutenue à SAINT-ETIENNE le 8 Juillet 1993

COMPOSITION DU JURY

Monsieur	J. M. BRUN	Rapporteurs
Mademoiselle	A. VERROUST	
Messieurs	J. C. LAFON	Examinateurs
	B. LAGET	
	B. PEROCHE	
	D. MICHELUCCI	

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

N° d'ordre **96 ID**

THESE

Présentée par

Mohand Ourabah BENOUAMER

pour obtenir le titre de

DOCTEUR
DE L'UNIVERSITE DE SAINT-ETIENNE
ET DE L'ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

(Spécialité : Informatique)

OPERATIONS BOOLEENNES SUR LES POLYEDRES REPRESENTES PAR LEURS FRONTIERES ET IMPRECISIONS NUMERIQUES

soutenue à SAINT-ETIENNE le 8 Juillet 1993

COMPOSITION DU JURY

Monsieur	J. M. BRUN	Rapporteurs
Mademoiselle	A. VERROUST	
Messieurs	J. C. LAFON	Examinateurs
	B. LAGET	
	B. PEROCHE	
	D. MICHELUCCI	

Remerciements

J'exprime mes plus vifs remerciements à tous les membres du jury :

Monsieur Jean Marc BRUN, Professeur à l'Université de Lyon I, rapporteur ;

Mademoiselle Anne VERROUST, Chargée de Recherches à l'INRIA de Rocquencourt ;

Monsieur Jean Claude LAFON, Professeur à l'ESSI-CERISI de Sophia Antipolis 1 ;

Monsieur Bernard LAGET, Professeur à l'UER de Sciences de Saint-Etienne ;

Monsieur Bernard PEROCHE, Directeur du Département Informatique Appliquée de l'Ecole Nationale Supérieure des Mines de Saint-Etienne, pour m'avoir accueilli en thèse et pour avoir bien voulu encadrer mes travaux ;

Monsieur Dominique MICHELUCCI, Ingénieur de Recherches à l'Ecole des Mines de Saint-Etienne, pour son aide assidue et ses conseils éclairés.

Je remercie tous les membres passés et présents du département informatique, permanents et thésards, pour leur sympathie et leur soutien moral. En particulier:

Les autres membres du groupe "paresseux": Dominique MICHELUCCI, Philippe JAILLON et Jean Michel MOREAU ;

Madame Marie-Line BARNEOUD, irremplaçable secrétaire du département informatique, pour sa disponibilité et sa gentillesse ;

Mes collègues thésards: Elisabeth ROUDIER, Laurent MALEYSSON, Helmi BEN AMARA, Bernard KADDOUR, Samy AIT-AOUDIA, Marc ROELENS, François JAILLET, Gabriel HANOTEAUX, Véronique BOURGOIN, Nahed ABDELFATTAH, ainsi que les nouveaux thésards: Jean Luc MAILLOT, Gilles MATHIEU et Boukhalfa HADIM.

Michel BEIGBEDER, Annie BOURGEAT, Jori LEONARDON, Jean Jacques GIRARDOT et Roland JEGOU.

Je n'oublie pas mes parents, pour leurs sacrifices, et tous mes amis "Hittistes" d'Alger.

Enfin, toute ma gratitude va à ma femme Titem, pour la patience dont elle a fait preuve au cours de mes années de thèse et toute mon affection va à ma petite Laetitia qui a dû subir mes horaires impossibles.

TABLE DES MATIERES

Chapitre II : *Un algorithme d'intersection de polyèdres rationnels définis par leurs frontières*

Chapitre III : *Programmation de l'algorithme. Difficultés dues à l'imprécision. Résultats et Tests.*

Chapitre IV : *Une optimisation pour les algorithmes d'évaluation d'arbres CSG*

INTRODUCTION

La modélisation solide a enregistré des progrès notables au cours des deux dernières décennies, grâce à une meilleure combinaison des techniques mathématiques et informatiques. Ces progrès ont beaucoup profité aux différents domaines d'application tels que l'analyse d'interférences, l'infographie, la génération automatique de réseaux d'éléments finis, le contrôle numérique de procédés de fabrication, la robotique, la conception de circuits intégrés, ...

Depuis la représentation filaire, ambiguë et rudimentaire, en passant par la représentation surfacique, insuffisante, plusieurs modèles volumiques de solides ont été développés [Requ-80]. Ces modèles ne cessent d'évoluer pour intégrer de plus en plus d'informations allant de la géométrie des solides jusqu'aux informations nécessaires à la fabrication des objets tels que les pièces mécaniques.

Cependant, devant les besoins accrus des applications, les modèles solides restent encore insuffisants voire inadaptés à certaines applications. Aussi, les systèmes de modélisation industriels combinent-ils souvent plusieurs représentations afin de satisfaire chaque besoin spécifique (visualisation, calcul de propriétés géométriques ou mécaniques, ...). Cette combinaison n'est pas sans inconvénients: elle implique une redondance des données qui pose fatalement le problème de *cohérence* entre les représentations.

Parmi les modèles volumiques, la représentation par *arbre de construction* (ou "Constructive Solid Geometry") et la représentation par la *frontière* ("Boundary Representation") ont été particulièrement utilisés par les modeleurs solides existants. Ces deux modèles sont complémentaires, dans un certain sens : la représentation CSG est intuitive et concise mais elle est implicite ; la représentation par frontière est riche en informations mais elle est de validation difficile. Aussi, de nombreux modeleurs les utilisent conjointement, quoique la représentation par frontière est souvent dominante ([RV-83], [Allen-84], [Mille-89]).

En général, tous les systèmes de modélisation solide combinant ces deux représentations comportent un module qui convertit une description CSG en une description par frontière équivalente.

Les algorithmes sous-jacents, dits d'*évaluation dè frontières* (ou "Boundary Evaluation" [RV-85]) sont bien connus dans la littérature, mais ils restent confrontés à un certain nombre de difficultés pour lesquelles on ne connaît que des solutions partielles.

Dans cette thèse, nous nous proposons de traiter trois difficultés essentielles dans le domaine de la modélisation polyédrique:

1) le traitement et la représentation corrects et uniformes des nombreux cas particuliers. En partant des travaux de Michelucci [Mich-87], nous détaillerons un algorithme simple d'intersection de polyèdres réguliers (non nécessairement *eulériens*), qui utilise une structure de données très générale ;

2) la prise en compte des imprécisions numériques et de leurs méfaits sur la cohérence des représentations par frontière. Nous décrirons une solution radicale basée sur l'emploi d'une arithmétique rationnelle originale [BJMM-93-a], dont le coût est très inférieur à celui des arithmétiques rationnelles classiques. Des résultats de tests seront présentés et commentés ;

3) enfin, nous aborderons le problème des calculs géométriques "inutiles" qui sont souvent effectués lors de la résolution des opérations booléennes. Dans le cas général, nous proposerons une méthode d'optimisation qui combine les zones actives de Rossignac et Voelcker [RV-89] et les boîtes de Cameron [Came-91]. Cette méthode permet de réduire raisonnablement le volume de ces calculs "inutiles".

PLAN DE LA THESE

Chapitre I : Etude bibliographique

Le premier chapitre regroupe les résultats de nos recherches bibliographiques dans le domaine de la modélisation solide. Il ne constitue nullement un rapport sur l'état de l'art dans ce domaine (voir[Ross-92]), mais plutôt un survol des principaux concepts qui seront évoqués tout au long de cette thèse. Nous rappelons quelques notions mathématiques de *topologie*, de *régularisation* et de *classification* [Tilo-80].

Nous insistons sur deux schémas particuliers de représentation des solides [Requ-80]: la représentation par arbre de construction et la représentation par la frontière. Leurs avantages et inconvénients sont évalués. Nous rappelons le principe de la résolution des opérations booléennes sur les objets solides décrits par leur frontières, puis nous mettons en évidence les difficultés qui surviennent dans ce domaine. En particulier, nous insistons sur le problème difficile des *incohérences* des représentations par frontière, dues aux imprécisions numériques.

Enfin, nous présentons brièvement quelques algorithmes particuliers, en indiquant leur comportement face aux difficultés exposées.

Chapitre II : Un algorithme d'intersection de polyèdres

Le deuxième chapitre donne la description détaillée d'une version améliorée de l'algorithme de Michelucci [Mich-87] pour l'intersection de polyèdres représentés par leurs frontières. Cet algorithme se caractérise par trois points essentiels: 1) il utilise une structure de données suffisamment simple et générale pour permettre une représentation uniforme des cas particuliers, fort nombreux dans domaine des solides *non eulériens* ; 2) il ne comporte qu'un unique cas particulier dont la prise en compte ne perturbe pas le fonctionnement général de l'algorithme ; 3) il produit toujours des solides cohérents à partir d'autres solides cohérents: la cohérence topologique est garantie par l'emploi d'une arithmétique *rationnelle* optimisée, présentée au chapitre III.

Chapitre III : Implantation robuste, tests et résultats

Le troisième chapitre présente une implantation d'un algorithme qui calcule la frontière d'un objet solide décrit par un *arbre de construction* [BMP-93], basé sur l'algorithme d'intersection de polyèdres présenté au chapitre précédent.

Nous revenons sur la notion de *solides rationnels* et présentons une solution permettant leur construction à partir des solides flottants habituels. De même, nous abordons le problème de l'approximation d'un solide rationnel par un solide flottant cohérent.

Nous présentons trois versions successives de l'algorithme: la première utilise une arithmétique rationnelle classique ; la deuxième inclut une heuristique simple à base d'epsilons ; la troisième utilise une arithmétique rationnelle optimisée, dite "paresseuse" [BJMM-93a].

En arithmétique paresseuse, les valeurs exactes des nombres ne sont pas toujours connues. Nous présentons une méthode qui permet d'associer des *clés* aux nombres "paresseux", sans l'aide des valeurs exactes [BJMM-93c]. Ces clés servent notamment dans les tables à *adressage dispersé* [Knut-81], pour accélérer les recherches dans les listes d'éléments géométriques.

Nous terminons par quelques résultats de tests commentés, qui permettent d'estimer et de comparer les performances des trois versions de l'algorithme.

Chapitre IV : Une méthode d'optimisation

Le quatrième chapitre est davantage prospectif. Il traite un problème très général survenant dans tous les algorithmes d'évaluation d'arbres CSG: il s'agit des calculs géométriques "inutiles" qui sont la conséquence du principe fondamental "générer et tester".

Nous proposons une méthode qui combine judicieusement deux techniques d'optimisation existantes: la première, due à Cameron [Came-91], est basée sur une classe particulière de boîtes englobantes et la deuxième, due à Rossignac et Voelcker [RV-89], exploite les propriétés algébriques des expressions booléennes.

Chapitre I

Modélisation solide: problèmes et solutions

INTRODUCTION

Dans ce chapitre, nous regroupons quelques notions de modélisation géométrique utilisées tout au long de cette thèse. En particulier, nous insistons sur la modélisation des objets *solides* tridimensionnels que l'on rencontre dans les diverses applications de la CAO (Conception Assistée par Ordinateur) ou de la synthèse d'images.

En section **A**, nous rappelons quelques concepts mathématiques sur la *topologie* et la *régularisation* des ensembles et leur application à la *classification* géométrique [Tilo-80].

En section **B**, nous présentons les fondements théoriques de la modélisation des solides, puis nous comparons deux schémas de représentation particuliers: la représentation CSG et la représentation par la frontière. Une introduction à la modélisation solide peut être trouvée dans [RV-83], [Allen-84] ou dans [Mant-88]. Un exposé exhaustif sur l'état de l'art dans ce domaine peut être trouvé dans [Ross-91] et dans [RR-92].

En section **C**, nous exposons le problème de la résolution des opérations booléennes régularisées sur les objets solides décrits par leurs frontières et les difficultés liées à ce problème. En particulier, nous insistons sur les effets de l'imprécision numérique inhérente à l'arithmétique flottante.

Enfin, en section **D**, nous présentons brièvement quelques algorithmes connus, en mettant en évidence leur comportement face aux difficultés exposées.

A. FONDEMENTS DE LA MODELISATION GEOMETRIQUE

La modélisation d'objets géométriques repose sur la théorie mathématique des *ensembles* et sur leurs propriétés *topologiques* (intérieur, fermeture, frontière) et *géométriques*.

A.1. Rappels de topologie élémentaire

Soit S un sous-ensemble de E^n (l'espace euclidien de dimension n). Un point p de E^n appartient à l'*intérieur*, $\mathbf{i}S$, de S s'il existe un voisinage de p (par exemple, une boule ouverte de E^n centrée en p) qui soit inclus dans S ; p appartient à l'*adhérence*, $\mathbf{k}S$, de S si tout voisinage de p contient un point de S ; p est un élément de la *frontière*, ∂S, de S s'il appartient à la fois à $\mathbf{k}S$ et à $\mathbf{k}(\mathrm{c}S)$, où $\mathrm{c}S$ (ou $\bar{S}$) est le *complémentaire* de S dans E^n. Un ensemble S est dit *ouvert* si $S = \mathbf{i}S$; S est dit *fermé* si $S = \mathbf{k}S$; enfin, S est dit *régulier* si $S = \mathbf{r}S$, où $\mathbf{r} = \mathbf{k} \circ \mathbf{i}$ désigne l'opérateur de *régularisation* ("adhérence de l'intérieur").

A.2. Opérations booléennes régularisées

La classe des ensembles réguliers n'est pas *fermée* sous les opérations booléennes usuelles (intersection $\cap$, union $\cup$, différence $-$). En revanche, leurs versions *régularisées* ($\cap^*$, $\cup^*$, $-^*$) préservent la régularité (Fig. 1). Les opérations booléennes régularisées sont définies par :

$$A \cap^* B = \mathbf{r}(A \cap B),$$
$$A \cup^* B = \mathbf{r}(A \cup B),$$
$$A -^* B = \mathbf{r}(A - B),$$
$$\mathrm{c}^* A = \mathbf{r}(\mathrm{c}A).$$

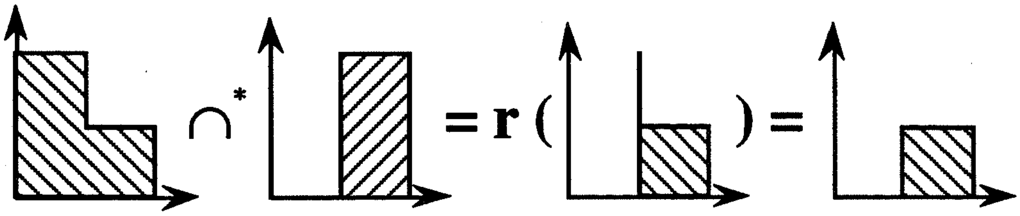


Figure 1. La régularisation élimine de la frontière tout point isolé ou toute partie pendante qui n'adhère pas à l'intérieur.

Ensembles réguliers et opérations booléennes régularisées forment une *algèbre booléenne* dans laquelle s'appliquent les lois bien connues de De Morgan :

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

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

A.3. Notion de classification

Beaucoup de problèmes pratiques de Géométrie Algorithmique (par exemple : test d'appartenance d'un point à un polygone ou à un polyèdre, calcul de l'intersection de deux polygones ou de deux polyèdres, découpage d'un segment ou d'un polygone contre une fenêtre polygonale ou polyédrique, ...) peuvent être reformulés en termes de *classification* ("Membership Classification" [Tilo-80]) d'un ensemble *candidat* X par rapport à un ensemble de *référence* S.

Soient W un sous-espace de E^n (n = 0, 1, 2, 3,...) et W' un sous-espace de W. La *classification* d'un sous-ensemble *régulier* X de W' par rapport à un sous-ensemble *régulier* S de W consiste en une partition, M[X, S], de X en trois sous-ensembles *X-dans-S*, *X-hors-S* et *X-sur-S* qui se trouvent à l'intérieur, à l'extérieur et sur la frontière de S, respectivement. Ces trois sous-ensembles sont des parties de X régulières au sens de la topologie de W' (ils sont soit vides, soit de même dimension que X) (Fig. 2).

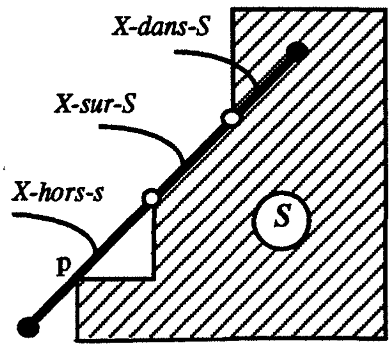


Figure 2. Classification d'un segment X par rapport à un polygone S. Le point p, bien qu'appartenant à ∂S, est exclu de *X-sur-S* par régularisation.

Plus précisément, si l'on désigne par r' l'opérateur de *régularisation* dans l'espace W', on a:

$$X\text{-}dans\text{-}S = \mathbf{r}'(X \cap \mathrm{i}S),$$
$$X\text{-}sur\text{-}S = \mathbf{r}'(X \cap \partial S),$$
$$X\text{-}hors\text{-}S = \mathbf{r}'(X \cap \mathrm{c}S).$$

A.4. Fusion de deux classifications

On peut être amené à classifier un candidat X par rapport à un ensemble de référence S défini implicitement par $S = A \% B$, où A et B sont deux sous-ensembles réguliers de W et % est une opération booléenne régularisée. En général, il n'est pas possible de déduire directement la classification M[X, S] par une simple combinaison (ou *fusion*) de M[X, A] et M[X, B]. Certains points p appartenant à la fois à ∂A et à ∂B peuvent se trouver aussi bien à l'intérieur, à l'extérieur ou sur la frontière de l'objet $A \% B$, selon l'opération booléenne (Fig. 3). Pour lever cette ambiguïté, il est nécessaire d'étudier la géométrie de A et B au *voisinage* immédiat du point p.

A.5. Notion de voisinage

Soit $\beta(p, \varepsilon)$ la boule ouverte de E^n centrée en p et de rayon $\varepsilon > 0$, suffisamment petit. Un *voisinage* régulier de p par rapport à un sous-ensemble régulier S de W est défini par $\vartheta(p, S, \varepsilon) = \mathbf{k}\beta(p, \varepsilon) \cap^* S$. Le point p est intérieur à S si, et seulement si, il possède un voisinage "plein" (i.e. $\vartheta(p, S, \varepsilon) = \mathbf{k}\beta(p, \varepsilon)$) ; p est extérieur à S si, et seulement si, il possède un voisinage "vide" (i.e. $\vartheta(p, S, \varepsilon) = \varnothing$) ; enfin, p est sur la frontière de S si, et seulement si, tout voisinage de p n'est ni "plein" ni "vide" (i.e. il est en partie dans S et en partie hors de S).

La Figure 3 ci-dessous montre que pour $S = A \cap^* B$, le voisinage $\vartheta(p, S, \varepsilon)$ défini par $\vartheta(p, S, \varepsilon) = \vartheta(p, A, \varepsilon) \cap^* \vartheta(p, B, \varepsilon)$ est un demi-disque dans le cas (a), et c'est l'ensemble vide dans le cas (b).

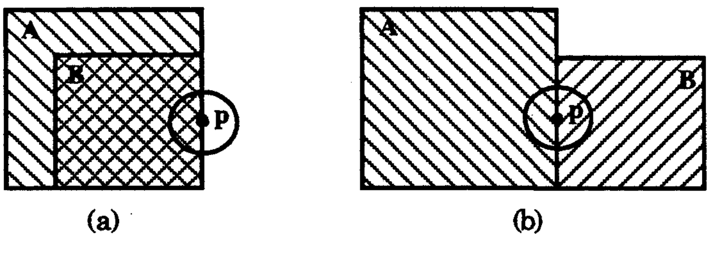


Figure 3. Dans les deux cas de figure, le point p appartient à la fois à ∂A et à ∂B. Dans le cas (**a**), p appartient à $\partial(A \cup^* B)$, à $\partial(A \cap^* B)$, mais il est extérieur à $A -^* B$ et à $B -^* A$ ($= \varnothing$). Dans le cas (**b**), p appartient à $\partial(A -^* B)$, et à $\partial(B -^* A)$, mais il est intérieur à $A \cup^* B$, et extérieur à $A \cap^* B$ ($= \varnothing$).

B. MODELISATION D'OBJETS SOLIDES

La conception d'un objet solide au moyen de l'outil informatique doit reposer sur un modèle suffisamment rigoureux qui permette de traduire mathématiquement, au moins dans le principe, toute propriété que l'on peut attendre d'un objet.

B.1. Modèles abstraits d'objets solides

On appelle *r-ensemble* tout sous-ensemble S de E^3 *fermé*, *borné*, *régulier* et *semi-analytique*. Un ensemble S est *semi-analytique* s'il peut s'exprimer comme une combinaison booléenne finie d'ensembles *analytiques*. Un ensemble *analytique* est défini par $\{(x, y, z) \in E^3 : \psi(x, y, z) \leq 0\}$, où ψ est une fonction qui admet un développement en série de puissances en chaque point de son domaine de définition.

Les *r-ensembles* de E^3 constituent des modèles mathématiques acceptables pour les objets solides tridimensionnels [Requ-80]. Intuitivement, ils décrivent des objets dont le bord est une surface *orientable*, suffisamment lisse, d'aire finie, délimitant un volume fini et ne comportant aucune partie "pendante" qui n'adhère pas à l'intérieur (le matériau). A priori, un solide peut être constitué de plusieurs morceaux ou comporter un nombre arbitraire de trous. On peut toujours envisager des solides *non bornés*, définis comme les *complémentaires* régularisés de solides *bornés*.

B.2. Représentation des solides

La mise en œuvre sur ordinateur d'un modèle de solide nécessite le codage du modèle par des combinaisons de symboles respectant certaines règles syntaxiques: un codage syntaxiquement correct constitue une *représentation* du solide. Divers schémas de représentation sont proposés pour les solides ([Requ-80], [Allen-84]), dont deux semblent prédominer en synthèse d'images [Pero-87] et dans les divers domaines d'application de la conception et de la fabrication assistées par ordinateur (CAO/CFAO). Il s'agit de la *représentation solide constructive* (CSG: "Constructive Solid Geometry") et de la *représentation par frontière* (BRep: "Boundary Representation").

B.2.1. Représentation solide constructive (CSG)

Fondamentalement, cette représentation repose sur un *arbre de construction*, dont les nœuds internes portent des opérateurs booléens régularisés et dont les nœuds terminaux désignent des *demi-espaces algébriques*. Dans E^3, ces demi-espaces sont de la forme $\{(x, y, z) \in E^3 : f(x, y, z) \leq 0\}$, où f est une fonction *algébrique* (Fig. 4).

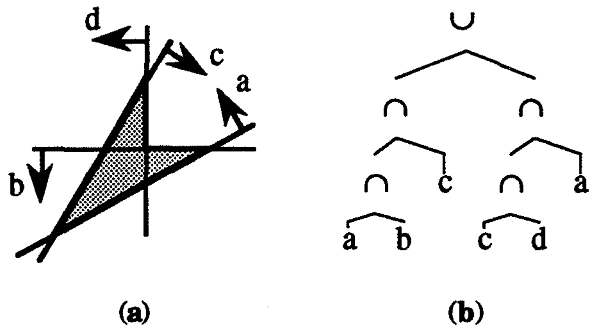


Figure 4. a) un polygone régulier P de E^2. **b)** un arbre CSG possible décrivant P comme une combinaison booléenne (régularisée) de quatre demi-plans orientés.

En pratique, l'arbre de construction est étendu aux opérateurs *unaires* désignant des transformations ponctuelles simples (translation, rotation, homothétie, affinité, symétrie), voire des *déformations* ([Barr-84], [SP-86], [Nie-91]). Très souvent, les nœuds terminaux portent des objets solides bornés (les *primitives*): typiquement, ce sont des instances de solides élémentaires (cubes, sphères, cônes, cylindres, tores, ...) ou encore des objets simples obtenus par *extrusion* ou par *révolution*. Enfin, certains nœuds peuvent être partagés, de sorte que l'on substitue à la structure d'*arbre* la structure, plus générale, de *graphe orienté sans cycles* [Beig-88].

La représentation CSG est "intuitive" car elle permet de décrire, naturellement (i.e. par une *méthode descendante*), un objet de forme complexe au moyen d'objets élémentaires. Elle est "implicite" car elle ne donne pas directement la frontière de l'objet, mais seulement un processus de construction de l'objet. Un arbre de construction correspond toujours à un objet solide valide (éventuellement vide), de sorte que la validité de la représentation peut être vérifiée au niveau syntaxique.

Il existe des algorithmes qui permettent de visualiser un objet solide directement à partir de son arbre de construction ([Ather-83], [Roth-82], [RR-86]).

Cependant, le calcul des propriétés géométriques ou physiques est plutôt laborieux. Notamment, la détection d'un objet CSG *vide* ne peut se faire sans une évaluation, au moins partielle, de l'arbre de construction ([Tilo-84], [RV-89]).

La représentation CSG n'est pas adaptée aux applications interactives dans lesquelles un utilisateur construit un objet par une succession de modifications locales (voir toutefois [CR-91]). Le positionnement spatial des objets primitifs reste une tâche contraignante pour l'utilisateur, qui préfère plutôt spécifier un ensemble de *contraintes* géométriques sur les objets ([RR-86], [Verr-90], [Emme-91], ...).

En définitive, cette représentation est à utiliser comme une description d'*entrée* ou d'*archivage* dans les systèmes de modélisation solide (voir [RV-83], [Mill-89]).

B.2.2. Représentation par frontière

Dans cette représentation, un objet solide est totalement défini par la "peau". La *frontière topologique* d'un solide est décrite par un *graphe* qui précise les relations d'incidence ou d'adjacence entre les sous-ensembles de frontière, de différentes dimension: typiquement des *faces*, des *arêtes* et des *sommets* (Fig. 5). Géométriquement, les faces sont portées par des *surfaces* (planes ou gauches) et les arêtes par des *courbes*.

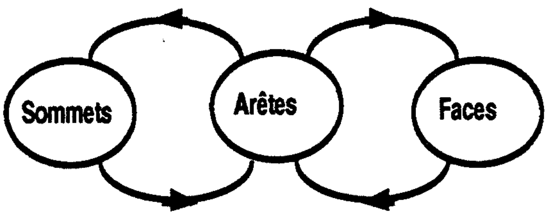


Figure 5. Exemple de graphe d'adjacence Faces/Arêtes/Sommets. Il donne les arêtes qui délimitent une face donnée, les arêtes incidentes à un sommet donné, les faces incidentes à une arête donnée, ou les sommets qui délimitent une arête donnée.

La frontière d'un objet peut être découpée selon une hiérarchie de divers sous-ensembles. Dans le cas général, la frontière comporte plusieurs composantes connexes (ou "Shells") constituées de *faces* plus ou moins complexes. Chaque face est décrite par une liste de *contours fermés* ("Loops"), délimités par les *arêtes*. Une orientation adéquate des contours permet de distinguer les contours *externes* de la face des contours *internes* qui délimitent les trous dans la face.

La représentation par frontière est donc une description explicite de la frontière de l'objet. Cette description est souvent nécessaire aux applications de CAO/CFAO en vue de l'usinage des surfaces de pièces mécaniques sur une machine à commande numérique. Elle permet un calcul efficace des propriétés géométriques (aire et volume) ou physiques (masse, centre de gravité, moment d'inertie, ...) des objets. Les algorithmes de visualisation et de rendu des objets décrits par leur frontière sont plus nombreux qu'en représentation CSG. La représentation par frontière rend possible la construction *incrémentale* des objets, par des modifications locales.

Des travaux récents font apparaître une nouvelle approche de modélisation par frontière fondée sur les *formes fonctionnelles* [1] ("Features" [Ross-90]). Ce concept permet de mieux prendre en compte les intentions de chaque classe d'utilisateurs. Par exemple, un mécanicien peut définir des formes familières telles que des alésages, des épaulements, des congés, des chanfreins , etc...

La vérification de la *validité* d'une représentation par frontière est un problème difficile: il s'agit de garantir non seulement la *cohérence topologique* (pas de faces ou d'arêtes manquantes, pas de faces ou d'arêtes pendantes, ...) mais aussi la *non contradiction* entre les données *topologiques* et les données *géométriques* spécifiées dans les structures de données (voir C.2.3).

L'efficacité d'un modeleur solide fondé sur la représentation par frontière dépend, dans une certaine mesure, des structures de données mises en œuvre pour décrire la frontière des solides. Depuis la structure de données "arête ailée" ("Winged-Edge" [Baum-72], [Baum-75]) conçue pour les besoins de la vision robotique, plusieurs variantes ont été proposées: une étude comparative sur une dizaine de versions est donnée dans [Ala-91] et [Ala-92].

Les divers schémas de représentation par frontière différent par leur *domaine de modélisation* (voir C.3), par le type de relations d'incidence explicitées dans le *graphe d'adjacence* [Weil-85] et par des critères pratiques (et contradictoires) liés à la simplicité, au volume de stockage, à la facilité de recherche et à la prise en compte des cas particuliers.

---

[1] Le terme de "forme fonctionnelle" correspond à notre traduction du terme Anglais "Form Feature" (Ross-90], [Emme-91], [Wu-92]).

B.2.3. Combinaison de représentations

La confrontation des divers modèles de représentation des solides [Requ-80] montre qu'aucun modèle ne peut satisfaire à lui seul tous les besoins des divers applications (synthèse d'images, CAO/FAO, robotique, ...). En particulier, il apparaît que les deux modèles de représentation par arbre de construction et par frontière sont complémentaires sur plusieurs aspects. Plusieurs systèmes de modélisation combinent les deux représentations afin d'allier leurs avantages respectifs. Cependant, une telle combinaison n'est pas sans inconvénients: à l'évidence, le maintien d'une double représentation pose le problème de la *cohérence* entre les représentations.

L'idéal serait de disposer d'un système de modélisation "hybride" dans lequel on puisse passer d'une représentation par arbre de construction (resp. par frontière) à une représentation par frontière (resp. par arbre de construction) strictement équivalente. Cependant, alors que l'on connaît un certain nombre d'algorithmes pour la conversion *constructive* $\rightarrow$ *frontière* , la conversion inverse reste un problème ouvert (voir [Peter-86] et [DGHS-88] pour le cas 2D, [Juan-88], [SV-91] et [SV-93] pour le cas 3D).

B.2.4. Quelques systèmes de modélisation solide

En pratique, les modeleurs utilisent l'une des deux représentations comme représentation *principale* (à priori, l'arbre CSG est plus adapté au stockage ou à la spécification des objets) et l'autre comme représentation *auxiliaire* (la représentation par frontière est plus adaptée à la construction *interactive* des objets et se prête mieux à leur visualisation ou au calcul de leurs propriétés).

D'après les quelques informations dont nous disposons, nous avons tenté d'établir la liste des principaux modeleurs solides connus à ce jour (voir aussi [RV-83], [RV-85], [Allen-84], [Mille-89]).

Parmi les modeleurs issus de diverses universités :

Modeleur	Université	Type
TIPS-1	Hokkaido (Japon)	CSG/Frontière
PADL-1/2 [RV-85]	Rochester (USA)	CSG/Frontière
UNISOLIDS [RV-85]	Rochester (USA)	CSG/Frontière
DMI[2] [NAB-86]	Barcelone (Espagne)	Frontière/CSG
GWB [MS-82]	Helsinki (Finlande)	Frontière/CSG
MINERVA [PRS-89]	Rome (Italie)	Frontière/CSG
PARASOLID[3] [SV-93]	Cornell, Ithaca (USA)	Frontière/CSG
BUILD [BHS-80]	Cambridge (Angleterre)	Frontière/CSG
MODIF [CK-83]	Tokyo (Japon)	Frontière/CSG
NOODLES [GCP-91]	Carnegie Mellon (USA)	Frontière/CSG
FREEDOM-II [YT-84]	Institut de Kyushu (Japon)	Frontière/CSG
MODEL [FR-88]	Navarre (Espagne)	Frontière/CSG
GEMS [LST-90]	Tsinghua (Chine)	Frontière/CSG
GEOMAP III [CK-83]	Tokyo (Japon)	Frontière/CSG

Parmi les modeleurs apparus sur le marche de la CAO/FAO :

Modeleur	Origine	Type
CATIA	IBM & Dassault (France)	Frontière/CSG
EUCLID	MATRADIVISION (France)	Frontière/CSG
PATRAN [Casa-87]	PDA Engineering, Californie (USA)	Frontière/CSG
ROMULUS	Evans & Sutherland (Angleterre)	Frontière/CSG
MEDUSA	PRIME (Angleterre)	Frontière/CSG
DESIGNBASE [TSUC-86]	Ricoh Co. Soft. Division (Japon)	Frontière/CSG
SOLIDESIGN	COMPUTERVISION (USA)	Frontière/CSG
GEOMOD-II	SDRC/General Electric CAE (USA)	Frontière/CSG
GLIDE [EH-77]		Frontière/CSG
SYNTHAVISION [GM-79]	MAGI (USA)	CSG/Frontière
GMSOLID [BR-82]		CSG/Frontière
GDP [WLLLG-80]		CSG/Frontière

Dans tous ces systèmes de modélisation, la principale composante est le module d'*évaluation de frontière*, qui consiste en un algorithme de résolution d'opérations booléennes sur des solides décrits par leurs frontières. Un certain nombre d'algorithmes connus seront présentés en section D.

---

[2] Pour la résolution des opérations booléennes, le modeleur DMI se sert exclusivement de la représentation par arbre octal étendu.

[3] Dans [SV-93], il est fait état d'un système expérimental qui sait convertir les quadriques naturelles du modeleur PARASOLID en arbres CSG du modeleur PADL-2.

B.3. Domaine de modélisation

La puissance d'un schéma de représentation solide dépend de l'étendue de son domaine de modélisation. On distingue deux types de domaines indépendants liés à la *géométrie* et à la *topologie* des objets représentables.

B.3.1. Domaine géométrique

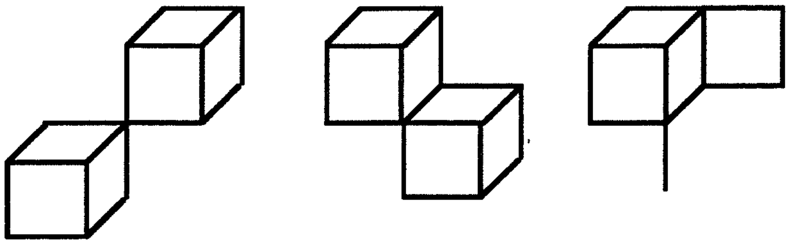
Dans le cas de la représentation solide constructive, le *domaine géométrique* est lié à la richesse du jeu de *primitives* adopté. Plus précisément, il dépend du *degré* des surfaces qui enveloppent les primitives. Dans le cas de la représentation par frontière, le domaine géométrique est déterminé par l'espace de *plongement* géométrique du graphe d'adjacence associé. En d'autres termes, il dépend du degré des surfaces et des courbes qui supportent les faces et les arêtes du solide, respectivement.

Les systèmes de modélisation actuels se limitent aux *quadriques* simples (sphères, cylindres ou cônes) [Mill-88], à certaines *cubiques* (MODIF [CK-83], PATRAN-G [Casa-87]) ou *quartiques* (des tores ou des "patches" de Steiner [SA-85]) ou à un *mélange* ("Blending") de ces surfaces. Le plus souvent, les modeleurs solides se contentent d'approximations *linéaires* raisonnables, obtenues par "facétisation" des courbes et des surfaces.

B.3.2. Domaine topologique

Le *domaine topologique* d'un schéma de représentation solide est lié à l'ensemble des propriétés indépendantes de la *position* et de l'*orientation* spatiales. Plus précisément, la *topologie* d'un solide est invariante par *homéomorphisme* (i.e. une transformation continue qui possède un inverse continu). Par exemple, le nombre d'*anses* d'un solide (trous à travers le solide) constitue une caractéristique topologique liée au *genre* de la surface qui enveloppe le solide (0 pour la sphère, 1 pour le tore, ...).

On dit qu'une surface est 2-*eulérienne* (ou "2-manifold") si, et seulement si, tout point p de la surface admet un voisinage topologiquement équivalent (ou *homéomorphe*) à un disque ouvert de E^2. Intuitivement, la surface peut être "aplatie" localement en un disque centré en p, par déformation élastique. On dit qu'un solide est *eulérien* ("2-manifold") si sa frontière est une surface 2-*eulérienne* [Ross-91].



(a) (b) (c)

Figure 6. L'union régularisée de deux solides eulériens (ici, deux cubes) peut produire un solide dont la frontière comporte un sommet avec deux "cônes" de faces incidentes (**a**) ou une arête avec plus de deux faces incidentes (**b**) : dans les deux cas, le résultat est un solide régulier non eulérien. La figure (**c**) donne un exemple d'objet qui ne peut être modélisé par un r-ensemble car il est hétérogène en dimension, un tel objet est un "non-manifold" plus général qu'un "pseudo non manifold".

La classe des solides *eulériens* n'est pas *fermée* sous les opérations booléennes régularisées. Certaines combinaisons booléennes d'objets *eulériens* peuvent produire des objets réguliers, mais dont la frontière comporte des points qui n'admettent pas de voisinage *homéormorphe* à un disque (Fig. 6a et 6b). De tels solides sont souvent qualifiés de *pseudo non eulériens* (ou "orientable non-manifold" [HHK-89]) afin de les distinguer des objets *hétérogènes* en dimension (Fig. 6c), qui ne sont pas modélisables par des *r-ensembles* (Sect. B.1).

B.3.3. Débat "manifold" versus "non-manifold"

Il existe un intéressant débat sur le choix du modèle topologique le plus adapté à la modélisation des objets solides. Certains auteurs, comme Mantyla [Mant-88] considèrent que seul le modèle *eulérien* est acceptable car il décrit des objets "manufacturables" (usinables). D'autres, comme Requicha et Voelcker [RV-85] ou Weiler [Weil-86] opposent l'argument selon lequel il existe des processus de fabrication qui ne peuvent pas être décrits en termes de solides *eulériens*.

Un autre argument en faveur du modèle *pseudo non eulérien* est avancé par Hoffmann, Hopcroft et Karasick [HHK-89], qui pensent que ce modèle simplifie les opérations booléennes car il permet de "banaliser" les positions relatives particulières pour les objets à combiner (par exemple, les deux cubes des figures 6a ou 6b).

Desaulniers et Stewart [DS-92] tentent d'unifier les deux points de vue en étudiant la relation existant entre la classe des *r-ensembles* et celle, apparemment plus restreinte, des solides *eulériens*. Ils montrent que tout *r-ensemble* peut être vu comme la *limite* d'une certaine suite de solides *eulériens*, au sens d'une certaine métrique (distance de Hausdorff), puis ils proposent un ensemble suffisant d'opérateurs (analogues aux opérateurs d'Euler) qui permettent de construire et de manipuler les *r-ensembles*. Bien avant, Weiler [Weil-86] avait proposé une représentation par frontière ("Radial Edge") pour les objets "non-manifold" généraux (hétérogènes en dimension), ainsi qu'un ensemble complet d'opérateurs topologiques pour manipuler ces objets.

Il semble que la notion même de *régularisation fermée* (au sens de Tilove [Tilo-80]), sur laquelle était fondée jusqu'ici toute la théorie de la modélisation des solides, commence à être perçue comme une *restriction* qui empêche la modélisation d'objets plus généraux tels que les agrégats d'objets hétérogènes en dimension (Fig. 6c), les assemblages dotés de *séparations internes* ([Arbab-90], [Ross-91]) ou encore les objets indépendants de la dimension [RR-91]. Enfin, des algorithmes d'opérations booléennes commencent à émerger pour les objets non régularisés ([GCP-91], [CR-91]).

C. OPERATIONS BOOLEENNES ET PROBLEMES LIES

C.1. Principe général des algorithmes

Fondamentalement, les algorithmes de résolution d'opérations booléennes reposent sur la notion de *classification* [Tilo-80] et sur le principe "Générer et tester": la frontière de l'union, de l'intersection ou de la différence de deux solides A et B est toujours incluse dans l'union des frontières de A et B.

C.1.1. Schéma de résolution d'une opération booléenne

Les algorithmes classiques de résolution d'opérations booléennes sur deux objets A et B représentés par leurs frontières sont articulés soit autour des *faces* ([SS-88], [PRS-89], ...), soit autour des *arêtes* [RV-85], soit autour des sommets [Mant-86]. D'une manière générale, ils procèdent en quatre étapes:

1) Calcul de toutes les intersections entre les frontières de A et B et leur découpage en éléments non intersectants (Fig. 8) ;

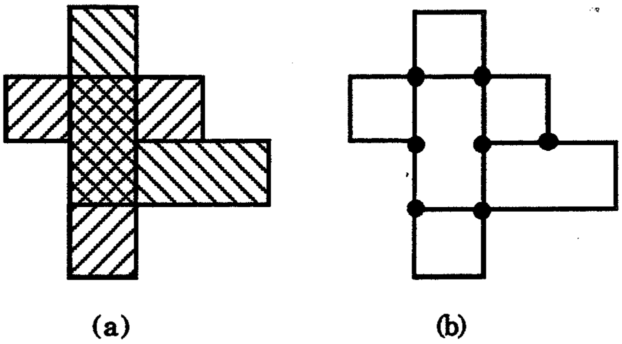


Figure 8. a) Deux objets A et B (formes en 'L' et 'T', respectivement) ;
b) les intersections entre les frontières de A et B.

2) Classification de chaque élément découpé comme appartenant à l'intérieur, à la frontière ou à l'extérieur de l'autre objet (Fig. 9) ;

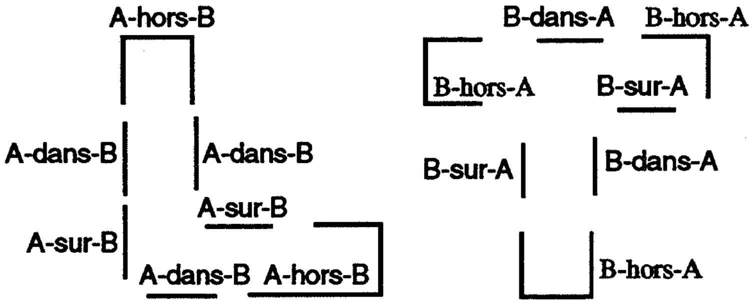


Figure 9. Classification des éléments de frontière de A et B.

3) Sélection, selon l'opération booléenne, des éléments à retenir dans la description de l'objet résultat. La sélection doit tenir compte de l'*orientation* des faces coplanaires, par une étude de *voisinage* (Fig. 10) ;

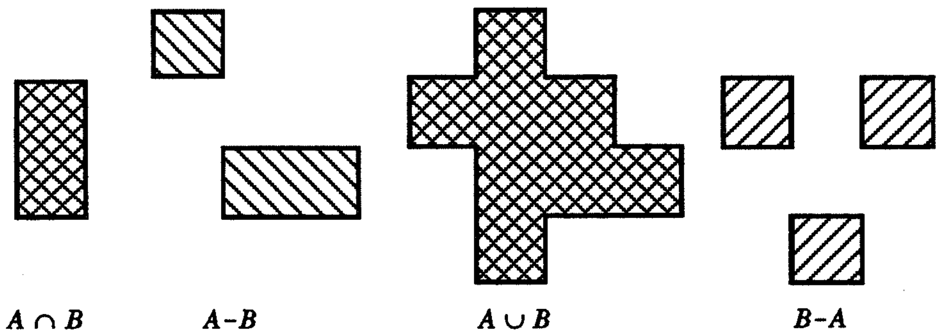


Figure 10. Sélection des éléments classifiés, selon l'opération booléenne.

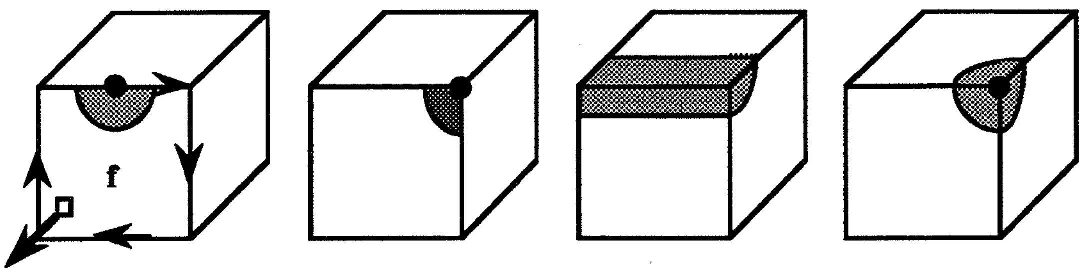
4) Construction d'un *graphe d'adjacence* à partir des éléments retenus lors de la *sélection*. Ce graphe permet de décrire la topologie de l'objet résultat.

C.1.2. Analyse de voisinage

Les éléments communs aux frontières des solides opérandes n'appartiennent pas nécessairement à la frontière du solide résultat. Pour lever les ambiguïtés possibles lors de l'étape de *sélection*, on recourt à une étude de *voisinage*. La frontière d'un solide cohérent est *orientable* (contrairement à la bande de Mœbius ou à la bouteille de Klein), de sorte que le *voisinage* de tout point frontière (sommet d'arête ou point intérieur à une arête) peut être déduit de l'orientation des faces et des arêtes. Chaque face est munie d'un *vecteur normal* dirigé (par exemple) de l'intérieur vers l'extérieur du solide et chaque contour de face est orienté suivant (par exemple) la règle de la "main droite" (Fig. 10a).

On distingue classiquement quatre types de voisinages [RV-85]: les voisinages bidimensionnels *arête/face* (Fig. 10a) ou *sommet/face* (Fig. 10b), qui précisent la géométrie d'une face localement à un point intérieur à une arête ou à un sommet d'arête, respectivement ; et les voisinages tridimensionnels *arête/solide* (Fig. 10c) ou *sommet/solide* (Fig. 10d), qui précisent la géométrie d'un solide localement à un point intérieur à une arête ou à un sommet du solide, respectivement.

L'analyse d'un voisinage *sommet/face* nécessite un tri radial, autour du sommet, de toutes les arêtes incidentes au sommet ; de même, l'analyse d'un voisinage *arête/solide* nécessite un tri radial, autour de l'arête, de toutes les faces incidentes à l'arête. Les divers algorithmes de résolution d'opérations booléennes diffèrent par le type des voisinages utilisés et par la manière (implicite ou explicite) dont ils sont décrits. Au chapitre II nous présenterons une description explicite simple des voisinages *arête/face*.



(a) (b) (c) (d)

Figure 10. Notion de voisinage: **a)** arête/face, **b)** sommet/face, **c)** arête/solide, et **d)** sommet/solide.

C.2. Difficultés liées à la résolution des opérations booléennes

Tous les algorithmes connus pour la résolution d'opérations booléennes entres solides décrits par leur frontière sont confrontés à trois problèmes majeurs : 1) réduire le volume des calculs géométriques nécessaires à la classification ; 2) prendre en compte tous les cas particuliers, fort nombreux ; et 3) assurer la *cohérence* des représentations par frontière des objets résultats.

C.2.1. Réduire le volume des calculs géométriques

Bien souvent, les algorithmes procèdent en intersectant deux à deux toutes les faces des solides opérandes. La méthode d'optimisation classique consiste en l'utilisation de boîtes englobantes autour des solides et des faces ([LTH-86], [SS-88], [Came-91]).

Des techniques plus élaborées de recherche *multidimensionnelle* ([Mehl-84], [PS-85]) permettent de déterminer en $O(K+n*\log^2 n)$ les K boîtes intersectantes parmi n boîtes considérées.

Bien que sans incidence sur la complexité théorique globale des algorithmes, le test des boîtes englobantes améliore sensiblement les performances *pratiques* des algorithmes. L'objectif est d'éviter en pratique les $M*N$ tests systématiques entre les M faces d'un solide et les N faces de l'autre solide.

D'autres techniques sont fondées sur les *répertoires* géométriques [MT-83], sur les diverses variantes des *arbres octaux* ("Octrees" [NAB-86], [BH-88], [Walk-89]), sur la *géométrie projective* de Monge [Szil-84] ou sur les "zones actives" de Rossignac et Voelcker [RV-89], sur lesquelles on reviendra au chapitre IV.

C.2.2. Traiter tous les cas particuliers

La prise en compte exhaustive des cas particuliers rend pénible l'implémentation des algorithmes d'intersection. Chaque cas particulier nécessite un traitement particulier, conduisant à des structures de données lourdes et à des algorithmes laborieux ([LTH-86], [MM-87]).

Certaines méthodes s'attachent à l'élimination systématique de tous les cas dégénérés, en simulant symboliquement une perturbation *infinitésimale* sur les données ([Yap-88], [EM-88]).

Ces méthodes sont génériques (elles ne dépendent pas des algorithmes), mais elles sont de mise en œuvre et d'utilisation délicates. Elles sont surtout utiles dans $\Re^n$ (n > 3), où l'intuition géométrique ne peut plus s'exercer en raison du nombre et de la complexité des cas particuliers: les dégénérescences telles que la *colinéarité* de trois points ou la *coplanarité* de quatre points sont difficiles à imaginer dans des espaces de dimension supérieure à trois: dans ce cas, le développement d'une librairie de "simulation de simplicité" [EM-88] se justifie et son usage est alors plus simple que la prise en compte directe des cas particuliers au niveau des algorithmes eux-mêmes.

C.2.3. Eviter les incohérences

Par définition, les représentations par frontière doivent correspondre à des objets solides cohérents. Le maintien de cette cohérence est un problème difficile qui appelle deux sous-problèmes liés: 1) assurer la *cohérence topologique* en vérifiant qu'une structure de données (décrivant une hiérarchie de faces, d'arêtes et de sommets) définit bien la frontière d'un objet solide (pas de faces manquantes, pas de faces ou d'arêtes pendantes, ...) et 2) assurer la *cohérence numérique* en vérifiant que les données *géométriques* (par exemple les coordonnées des sommets) ne sont pas en contradiction avec les données topologiques (graphe d'adjacence) (Fig. 11).

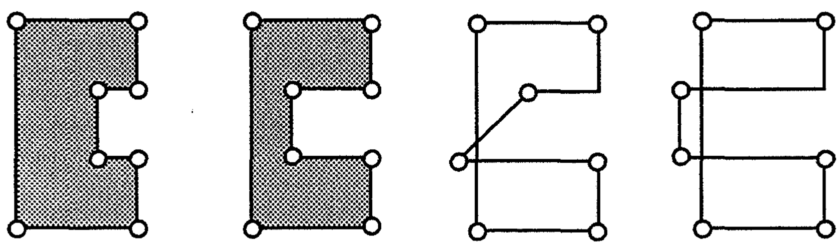


Figure 11. Soit f (a) une face valide décrite dans la représentation par frontière d'un objet cohérent. Entres autres critères, les arêtes de f ne s'intersectent qu'en des points qui sont décrits explicitement comme des sommets et elles délimitent clairement l'intérieur et l'extérieur de f. Pour le même graphe d'adjacence, si l'on perturbe les coordonnées d'un ou de plusieurs sommets de f, la face peut rester cohérente (b) ou devenir incohérente (c et d).

Cohérence topologique

Mantyla et Sulonen [MS-82] assurent la cohérence topologique de leur modeleur solide GWB par le strict emploi des opérateurs d'Euler. Dans le cas des solides *eulériens*, un objet est créé par une séquence d'opérateurs de bas niveau (ajout d'un sommet, d'une arête, d'une face, ...) qui préservent un certain *invariant* topologique. Pour un solide composé de V sommets, E arêtes, F faces connexes, S composantes connexes ("Shells"), H anses topologiques ("Handles": trous à travers le solide) et R contours internes ("Rings": trous dans les faces) cet invariant est donné par la formule d'Euler-Poincaré:

$$V - E + F = 2 * (S - H) + R.$$

Cinq opérateurs (avec leurs inverses) sont suffisants pour décrire la frontière de tout solide *eulérien* ([BHS-80], [MS-82], [Mant-88]). Cependant, les opérateurs d'Euler sont purement topologiques, ils ne garantissent pas la cohérence entre données numériques et données topologiques.

Cohérence numérique

Les incohérences *numériques* sont fondamentalement dues à l'imprécision numérique qui entache tout calcul en précision finie. Les données géométriques sont habituellement représentées par des nombres *flottants* approchés ; or les algorithmes géométriques ont souvent à comparer des éléments (points, droites, plans,...) issus d'objets distincts, qui peuvent être arbitrairement voisins: leur distance peut être inférieure à la précision de représentation de chaque objet. Dans ce cas, il serait hasardeux de conclure (par exemple) à la coïncidence de deux points p et q sur la base d'un simple calcul flottant portant sur les coordonnées.

C.3. Approches de solution au problème de l'imprécision numérique

Michelucci [Mich-87] a mis en évidence les effets néfastes des imprécisions numériques sur les algorithmes géométriques: les programmes peuvent avorter même dans des situations les plus ordinaires. Les algorithmes qui exploitent la cohérence *locale* des objets sont particulièrement sensibles à l'imprécision numérique: c'est le cas de l'algorithme d'intersection de segments de Bentley-Ottman [BO-79], qui utilise l'*y-ordre* local induit par le balayage du plan (la version "naïve" qui consiste à intersecter les segments deux à deux se comporte mieux car elle ne propage pas les ereurs).

A priori, la *robustesse* d'un algorithme dépend du *conditionnement* des données et du type de calculs effectués sur ces données. Nous présentons ci-dessous quelques approches de solution présentées dans la littérature:

C.3.1. Heuristiques simples à base d'epsilons

Laidlaw, Trumbore et Hugues [LTH-86] considèrent égales deux valeurs x et y telles que $|x - y| < \varepsilon$, où ε est un seuil empirique fixé à l'avance. Dans leur algorithme d'intersection, deux points p et q distants de moins de ε sont fusionnés. Lorsqu'un nouveau sommet appartenant à une arête ou à un plan est calculé, il est *projeté* sur l'arête ou le plan afin d'éviter de propager les petites erreurs. Cependant, cette heuristique simple ne garantit pas toujours la cohérence des résultats, comme l'admettent les auteurs eux-mêmes.

C.3.2. Intervalles de confiance

Guibas, Salesin et Stolfi [GSS-89] décrivent un cadre théorique ("epsilon-geomety") qui permet de quantifier les erreurs numériques. Soient O un ensemble d'objets géométriques muni d'une métrique $\|.,.\|$ et P un prédicat géométrique sur O. Pour tout $\varepsilon > 0$ et pour chaque objet $X \in O$, on définit un prédicat $\varepsilon\text{-}P(X)$ tel que : $\varepsilon\text{-}P(X) \Leftrightarrow (\exists X' \in O)$, $(\|X, X'\| \leq \varepsilon)$ tel que $P(X')$. Intuitivement, $\varepsilon\text{-}P(X)$ est vrai si X est à au plus ε de satisfaire $P(X)$.

Les auteurs proposent d'implanter chaque test géométrique $P(X)$ comme une procédure qui retourne, non pas une simple valeur logique (*vrai* ou *faux*), mais plutôt une estimation de l'erreur commise en considérant que X satisfait P (en particulier, $0\text{-}P \equiv P$). Plus précisément, la procédure retourne une partition $(\vartheta, \Im, F)$ de la droite numérique $\Re$ telle que le prédicat $\varepsilon\text{-}P(X)$ est *vrai* pour $\varepsilon \in \vartheta$, *faux* pour $\varepsilon \in F$ et *indéfini* pour $\varepsilon \in \Im$.

Cette approche permet de préciser les intervalles de "confiance" dans lesquels on est assuré que les calculs numériques conduisent à des décisions cohérentes. Mais comment décider hors de ces intervalles ?

C.3.3. Tolérances explicites

Segal et Sequin [SS-88] généralisent l'heuristique des eplisons en associant à chaque élément géométrique (face, arête ou sommet) une *tolérance* explicite qui peut varier au cours de l'algorithme. Cette approche est assez laborieuse mais permet souvent d'éviter les incohérences qui résultent d'un choix "aveugle" des epsilons.

Dans [SS-85] les mêmes auteurs proposent que chaque représentation par frontière soit réaménagée de façon que tous les éléments de frontière soient séparés les uns des autres par au moins une distance μ fixée à l'avance. Les éléments distants de moins de μ doivent être soit "fusionnés", soit nettement séparés de façon à maintenir la distance μ. Dans une telle représentation, toute boule de rayon $\mu/2$ ne peut contenir qu'au plus un sommet. Cependant, ce réaménagement ("consolidation") ne peut être effectué élément par élément, nous reviendrons sur ce point au chapitre III.

C.3.4. Raisonnement symbolique

Cette famille de méthodes est fondée sur un constat simple: on peut se contenter de calculs en précision finie tant que les décisions logiques prises sur la foi de ces calculs ne se contredisent pas mutuellement. Dans le cas général, seul un puissant *démonstrateur de théorèmes* permet d'assurer la cohérence d'une séquence de décisions. Cependant, en comprenant le type d'incohérences qui peuvent survenir dans chaque algorithme particulier, il est possible d'organiser les tests géométriques de façon à respecter un ensemble limité de *règles* qui garantissant la cohérence *globale* de l'algorithme [Verr-90]. Cette approche relève du *raisonnement symbolique*, sa mise en œuvre dépend de chaque algorithme particulier.

• Hoffmann, Hopcroft et Karasick ([HHK-88], [HHK-89]) utilisent systématiquement le raisonnement symbolique pour l'implantation de leur algorithme d'intersection de polyèdres. Leur approche consiste à :

1) contenir la propagation des erreurs en limitant au maximum la *redondance* des calculs et des données: il est bien connu que la redondance est une source possible de contradictions. Dans leur représentation par frontière des solides, chaque sommet (resp. arête) est défini(e) explicitement comme l'intersection de trois (resp. deux) plans, de sorte que tous les tests géométriques se ramènent à la comparaison de plans, pris comme données initiales ;

2) associer à chaque résultat de calcul une *tolérance* qui tiennne compte de la taille des données et du type des calculs. Par exemple, si le point p d'intersection de trois plans est calculé en résolvant un système linéaire par la méthode de Gauss, le *conditionnement* du système donne une bonne estimation de l'erreur

maximale ε_p sur les coordonnées de p, et deux points p et q calculés de cette manière peuvent être considérés distincts lorsque $\|p, q\| \geq (\varepsilon_p + \varepsilon_q)$;

3) veiller à ce que les décisions logiques prises sur la foi d'un calcul numérique ne soient pas conflictuelles: chaque test d'incidence entre deux éléments X et Y issus de deux objets distincts A et B est basé sur la *stratégie d'inférence* [Kara-89] suivante :

a) <u>test de proximité</u> :
si $\| X , Y \| > \varepsilon$ alors X et Y ne sont pas incidents ;

b) <u>recherche des conflits</u> :
autrement, si X et Y ont dans A et B des éléments adjacents X' et Y' distants d'au moins ε, mais dont l'incidence découle logiquement de celle de X et Y, alors X et Y ne sont pas incidents ;

c) <u>conclusion</u> :
si de tels éléments X' et Y' n'existent pas alors X et Y sont considérés incidents.

Notons que X et Y peuvent être déclarés non incidents soit du fait de leur éloignement ($\|X, Y\| > \varepsilon$), soit du fait de la détection d'un conflit. Dans le premier cas, leur position relative dans un certain système local de coordonnées peut être calculée en précision finie. Dans le deuxième cas, cette position relative se détermine en remontant la chaîne de raisonnement logique qui a conduit à la contradiction.

• Mantyla [Mant-86], pour son algorithme d'intersection, utilise également des tolérances dépendant de la taille des objets. De plus, toutes les procédures géométriques sont implantées en une hiérarchie dans laquelle les modules de "haut niveau" font appel aux modules de "bas niveau". L'ordre d'exécution des procédures est soigneusement sélectionné (voir détails dans [LST-90]).

• Milenkovic [Mile-88], pour son algorithme d'*arrangement* de segments dans un plan, tient à jour un *historique* de toutes les décisions logiques (du type "au dessus de", "au dessous de", "sur") prises au cours de l'algorithme. Une nouvelle décision n'est insérée dans l'historique que si elle est compatible avec toutes les décisions déjà enregistrées.

• Milenkovic et Li [ML-90] composent avec l'imprécision numérique en s'attachant à garantir seulement que l'objet résultant de leur algorithme vérifie une certaine propriété *caractéristique*. Ainsi, étant donné un ensemble S de points du plan, ils construisent une *enveloppe convexe* $C_{\varepsilon,\delta}$ telle que: 1) $C_{\varepsilon,\delta}$ est effectivement un polygone *convexe* à sommets dans S, 2) aucun point de S ne se trouve à plus de δ, hors de $C_{\varepsilon,\delta}$, et 3) si l'on perturbe les sommets de $C_{\varepsilon,\delta}$ par moins de ε alors $C_{\varepsilon,\delta}$ reste *convexe*.

C.3.5. Arithmétique exacte

Le recours à une arithmétique *exacte* (le plus souvent, *rationnelle*) permet d'éviter systématiquement toutes les erreurs numériques. Les algorithmes géométriques qui les utilisent n'ont plus à se préoccuper du problème des imprécisions numériques. En revanche, les arithmétiques exactes sont très coûteuses, aussi bien en temps de calculs qu'en place-mémoire. De plus, les données numériques nécessitent une représentation particulière. C'est pourquoi le domaine d'application des arithmétiques exactes est restreint au *Calcul Formel* et à quelques rares algorithmes de *Géométrie Algorithmique*. Pour réduire le coût des arithmétiques rationnelles, diverses optimisations sont proposées dans la littérature:

• Greene et Yao [GY-86] ont choisi de renoncer à la *continuité* de l'espace. Les auteurs d'algorithmes raisonnent souvent (à tort) sur des paramètres *continus* bien que, lors de l'implémentation, ces paramètres ne peuvent prendre que des valeurs *discrètes* (des nombres flottants).

Partant de ce constat, Greene et Yao décrivent une version *discrète* d'un algorithme qui calcule toutes les intersections dans un ensemble de segments du plan, dont es extrémités sont les nœuds d'une *grille entière*. Les points d'intersection sont calculés en *arithmétique rationnelle* par la méthode de Bentley-Ottman [BO-79], puis "tirés" sur les nœuds les plus proches de la grille. Les segments originaux sont remplacés par des chaînes polygonales *monotones* qui tiennent compte d'un ensemble de contraintes préservant la topologie initiale. Un développement en *fractions continues* est utilisé pour le calcul des pentes des segments finaux.

• Sugihara et Iri [SI-89] observent que si les données géométriques initiales sont connues avec seulement un nombre fini n de bits de précision, alors la configuration *topologique* relative de deux ou plusieurs éléments géométriques peut être calculée exactement, dans une certaine précision finie m ($m \geq n$). Les auteurs ont expérimenté ce fait dans le cas bidimensionnel et proposent une généralisation 3D pour la conception d'un modeleur polyédrique.

Les arbres CSG considérés ont pour primitives des polyèdres dont chaque sommet a exactement *trois* faces incidentes. Les coefficients a, b, c et d des plans des faces de chaque primitive sont arrondis aux entiers les plus proches [Sugi-89] puis sont pris comme données de base. Les opérations booléennes ne créent pas de nouveaux plans et tout nouveau point d'intersection peut être caractérisé par trois plans concourants. Tous les calculs géométriques sont ramenés à l'étude du *signe* d'un *déterminant* 4x4 qui donne la configuration relative de quatre plans.

Les auteurs montrent que si $Max(|a|, |b|, |c|) \leq L$ et $|d| \leq L^2$, le développement d'un déterminant ne fait intervenir aucun terme intermédiaire qui soit supérieur à $16L^5$. Dans ces conditions, une arithmétique *entière*, bornée, sur environ $(4 + 5 * \log(L))$ bits ne produira pas de débordement. Sur une machine à entiers de p-bits, pour avoir $16L^5 \leq 2^p - 1$ (l'entier maximal), il faudrait $L \leq 42$ pour $p = 32$ et $L \leq 3565$ pour $p = 64$. Dans le cas général, seule une arithmétique *logicielle* à entiers *non bornés* laisserait ouverte la dynamique des données, indépendamment de la machine.

• Karasick, Lieber et Nackman [KLN-89] utilisent une arithmétique rationnelle pour leur algorithme de *triangulation* de Delaunay. Comme cet algorithme a souvent besoin de déterminer le *signe* d'un certain *déterminant* D, les auteurs exploitent la technique des *intervalles* pour encadrer la valeur de D. La méthode utilisée consiste en un processus itératif qui s'arrête en général sans avoir évalué explicitement la valeur de D. A l'étape k, chaque élément D_{ij} est encadré par un intervalle entier $[a_{ij}^k, b_{ij}^k]$ construit sur les k premiers *chiffres* de D_{ij} (dans une base donnée), puis D est développé en termes d'intervalles en utilisant une variante de la méthode de Gauss. Si l'intervalle résultant ne contient pas *zéro*, le *signe* de D est déduit et le processus s'arrête ; sinon, la méthode réitère en considérant un *chiffre* de plus dans la représentation de chaque élément D_{ij}, de façon à préciser les encadrements.

Bien que cette approche paraisse assez "lourde" à première vue, elle permet de réduire sensiblement les temps de calcul de l'algorithme.

• Gangnet et Thong [GT-91] utilisent également la technique des encadrements, mais les intervalles considérés ont pour bornes des nombres *flottants*.

• Ottman, Thiemt et Ullrich [OTU-87] font appel à un opérateur *produit scalaire exact* pour implanter un algorithme robuste d'intersection de segments du plan.

C.3.6. Arithmétique exacte réticente

Moreau [More-90] décrit une arithmétique "mixte" *flottante/rationnelle* dite "réticente" en ce sens qu'elle exploite au mieux les capacités de l'arithmétique flottante disponible et ne fait appel au calcul rationnel qu'en dessous de certains seuils, calculés à l'avance.

Dans une arithmétique flottante à p bits de précision (p est la taille en bits de la *mantisse*), on sait que l'erreur absolue sur le résultat d'une opération arithmétique $a \perp b$ ($\perp \in \{+, -, *, /\}$) est bornée par $\varepsilon_{a,b} = 2^{1-p} * |a \perp b|$ [Knut-81]. Cette erreur dépend bien sûr de la taille des nombres, mais si l'on peut prévoir que <u>tous</u> les nombres d'une application donnée seront compris dans un intervalle raisonnable $[-M, +M]$, alors l'erreur sur le calcul de $a-b$ (par exemple) sera bornée par $\varepsilon = 2^{2-p} * M$, quelles que soient les valeurs de a et b. Dans ces conditions, si $|a-b| > \varepsilon$ alors on est sûr que $a \neq b$ et on peut ordonner ces deux nombres en *arithmétique flottante* ; sinon, on fait appel à une comparaison en *arithmétique rationnelle*.

Nous pensons que le concept de "réticence" (vis à vis du calcul rationnel) n'est pas une propriété intrinsèque de l'arithmétique rationnelle utilisée, mais elle relève plutôt d'une optimisation de chaque algorithme d'application.

C.3.7. Arithmétique rationnelle paresseuse

Les algorithmes géométriques qui utilisent une *arithmétique rationnelle* n'ont pas besoin d'y recourir systématiquement, à chaque calcul numérique.

Certaines valeurs numériques (telles que la *puissance* d'un point par rapport à un plan, un *déterminant*, un *produit scalaire*, ...) ne sont calculées que dans le seul but d'en extraire le *signe*.

Par ailleurs, de nombreuses données géométriques (par exemple, des *coordonnées* de sommets) sont calculées exactement bien que les valeurs *exactes* ne soient jamais utilisées par la suite.

Nous présenterons au chapitre III une arithmétique rationnelle optimisée, dite *paresseuse* [BJMM-93a], qui n'effectue que les calculs exacts strictement nécessaires.

D. ETUDE DE QUELQUES ALGORITHMES CONNUS

Parmi tous les algorithmes connus de résolution d'opérations booléennes, seulement quelques uns abordent les problèmes évoqués précédemment et les traitent de façon plus ou moins satisfaisante.

On distinguera deux classes de méthodes: celles (dites *directes*) qui opèrent sur des objets représentés par leurs frontières et produisent une représentation par frontière, et celles (*indirectes*) qui utilisent une représentation intermédiaire autre que la représentation par frontière (typiquement, une subdivision spatiale).

Une étude sommaire de quelques algorithmes particuliers nous permettra de mieux situer l'algorithme d'intersection détaillé au chapitre III.

D.1. Méthodes directes

Les méthodes présentées sont classées en fonction de la classe des objets acceptés comme opérandes: convexes, eulériens, pseudo non eulériens, ou non eulériens (non régularisés).

D.1.1. Objets convexes

• Muller et Preparata [MP-78] décrivent un algorithme qui calcule l'intersection de deux polyèdres *convexes* en $O(n*\log(n))$, n étant le nombre total de sommets des deux polyèdres. Un polyèdre convexe P contenant l'origine peut être décrit comme l'intersection d'un nombre fini de *demi-espaces* H_i définis par des équations de la forme: $a_i.x + b_i.y + c_i.z \leq 1$. L'*enveloppe convexe* de tous les points (a_i, b_i, c_i) définit le *dual* P^* du polyèdre P.

Etant donnés deux polyèdres *convexes* P et Q contenant l'origine, Muller et Preparata démontrent que le polyèdre $P \cap Q$ est égal au *dual* de l'enveloppe convexe de $P^* \cup Q^*$.

L'enveloppe convexe de n points se détermine en $O(n*\log(n))$ [PS-85]. L'algorithme de Muller-Preparata utilise une structure de données très souple: la liste doublement chaînée d'arêtes (ou DCEL: "Doubly-Connected Edge List"), variante de l'"arête ailée" de Baumgart [Baum-72].

Cet élégant algorithme n'est malheureusement pas généralisable aux polyèdres non convexes.

• Hertel, Mantyla, Mehlhorn et Nievergelt [HMMN-84] utilisent aussi la structure de données DCEL et décrivent un autre algorithme qui calcule l'intersection, l'union ou la différence de deux polyèdres convexes P et Q, toujours en $O(n*\log(n))$,où n est le nombre total de sommets. La méthode consiste en une version tridimensionnelle de l'algorithme de Bentley-Ottman [BO-79]: un plan Γ balaie l'espace orthogonalement à une direction fixée arbitrairement et, au fur et à mesure du balayage, on tient à jour deux *arbres équilibrés* qui gardent la trace des arêtes et des faces de P ou Q coupées par Γ. Cet algorithme est sensible aux imprécisions numériques du fait qu'il exploite la cohérence *locale* induite par le balayage.

• Chazelle [Chaz-80] partitionne les polyèdres en parties *convexes*, puis effectue les opérations booléennes sur cette partition. Cependant, la méthode est assez lourde et de programmation difficile [Mant-82].

D.1.2. Objets eulériens

• Szilvasi-Nagy [Szil-84] décrit un algorithme d'intersection entre polyèdres *eulériens simples*, mais non nécessairement convexes. Les points d'intersection entre les arêtes d'un polyèdre et les faces de l'autre polyèdre sont calculés par projection sur les plans Oxy et Oxz au moyen de la *géométrie projective* de Monge.

Tirant parti des algorithmes décrits dans [BO-79] et dans [HMMN-84], l'auteur décrit un algorithme d'intersection en $O((n+k)*\log(n))$, où n est le nombre total de sommets des deux polyèdres et $k = O(n^2)$ est le nombre d'intersections entre les projections des arêtes sur le plan Oxy. Bien qu'exploitant la procédure de balayage de Bentley-Ottman [BO-79], l'auteur ne signale pas de problèmes dus aux imprécisions numériques. Enfin, les polyèdres résultats ne restent pas *eulériens* sous les opérations booléennes.

• Yamagushi et Tokieda [YT-84] simplifient les opérations booléennes en *triangulant* les faces potentiellement intersectantes. Les solides sont représentés par la structure "arête ailée" [Baum-75], généralisée aux faces trouées par adjonction d'arêtes "pont" qui relient chaque contour *interne* d'une face au contour *externe* de la face. Leur modeleur Freedom II accepte les solides dont la frontière comporte des morceaux de *quadriques* simples (cylindres, sphères ou cônes). Le module d'intersection de facettes triangulaires est suffisamment simple pour être *câblé*.

• Ansaldi, De Floriani et Falcidieno [AFF-85] décrivent la frontière d'un objet solide à l'aide d'une structure d'*hypergraphe*, dont les nœuds représentent les faces et les *hyperarcs* représentent les sommets et les arêtes du solide. Ils définissent un ensemble d'opérateurs d'Euler qui permettent la définition et la manipulation incrémentales des solides.

• Mantyla ([Mant-86], [Mant-88]) décrit un algorithme d'opérations booléennes fondé sur l'étude de voisinages *sommet/solide*. Le voisinage d'un sommet est défini par un cycle ordonné de faces et d'arêtes incidentes à ce sommet (Sect. C.1.2, Fig. 10d). L'algorithme utilise la structure de données "demi-arête" ("Half-Edge"), une autre variante de l'"arête ailée" [Baum-75]) et fait appel à la technique de section d'un solide par un plan.

Les cas particuliers sont pris en compte et la *classification* des voisinages repose sur des règles de décision garantissant la régularité et la cohérence topologique du solide résultat. Mantyla pense qu'une implémentation *robuste* de son algorithme s'obtient en hiérarchisant toutes les procédures géométriques, affectées de*tolérances* propres (cette idée est plus tard détaillée dans [LST-90]). Cependant, Mantyla admet que son approche est insuffisante pour le traitement correct des faces *quasi-coplanaires*.

• Paoluzzi, Ramella et Santarelli [PRS-89] décrivent une nouvelle représentation par frontière baptisée "triangle ailé" ("Winged-Triangle"), fondée sur une *décomposition simpliciale* (voir aussi [PSB-91], [FP-91], [PBCF-93]). L'élément topologique de base est la *facette triangulaire* (un 2-*simplexe*), définie comme un tuple $\tau = <s_1, s_2, s_3 ; \tau_1, \tau_2, \tau_3>$ de trois sommets s_i (des 0-*simplexes*) et des trois facettes τ_i adjacentes à τ (par les arêtes).

Ce schéma de représentation associe à chaque polyèdre (un 3-*complexe simplicial*) une "enveloppe" qui est une surface 2-*eulérienne* ("2-Manifold"), de sorte que les polyèdres *pseudo non eulériens* sont également acceptés par l'algorithme: un élément de frontière "non eulérien" tel que l'arête de la Figure 6(a) ou le sommet de la figure 6(b) est représenté, de façon interne, comme deux éléments *topologiquement* distincts.

La frontière d'un solide peut être constituée de plusieurs composantes connexes ("Shells"), chacune définie par un ensemble de *tuples*. L'algorithme d'union de deux solides A et B est relativement simple et sans cas particuliers ; il procède composante par composante, découpant chaque paire (τ_1, τ_2) de triangles de $\partial A \times \partial B$ en triangles non intersectants (la notion d'*arête* n'intervient plus dans les opérations booléennes).

La *complémentation* d'un objet revient simplement à *permuter* deux sommets et deux triangles dans chaque tuple de chaque composante connexe. Les opérations d'intersection et de différence se déduisent via les lois de De Morgan.

Les auteurs concluent que leur modeleur solide Minerva est *fermé* sous les opérations booléennes régularisées: lors de l'intersection de deux triangles, les points ou les segments d'intersection "non eulériens" sont tout simplement ignorés dans l'intersection. Cependant, les auteurs n'évoquent pas le problème des imprécisions numériques, qui rend hasardeuse la détection de tels points ou segments.

D.1.3. Objets pseudo non eulériens

• Laidlaw, Trumbore et Hughes [LTH-86] présentent un algorithme d'intersection entre deux solides polyédriques dont les faces sont des polygones convexes. L'algorithme découpe tous les polygones des deux solides A et B en sous-polygones *convexes* qui ne s'intersectent qu'en au plus un sommet ou une arête commun(e). Le mode de découpage des polygones dépend du type d'intersection (un polygone est subdivisé en 2 à 6 sous-polygones). Ensuite, chaque sous-polygone de chaque solide est *classifié* par rapport à l'autre solide. Pour cela, un *rayon* issu du centre du sous-polygone est lancé dans la direction de la *normale* au sous-polygone, puis intersecté avec tous les polygones de l'autre solide.

Le découpage des polygones produit généralement une myriade de petits polygones, dont le traitement donne prise aux incohérences. Moyennant une heuristique simple à base d'epsilons, l'algorithme arrive à discerner deux cubes centrés à l'origine, dont l'un est tourné d'un angle $\alpha > 0.5$ degrés autour de chaque axe de coordonnées.

Enfin, l'algorithme s'exécute en temps $O(m^2 + n^2)$, où m et n sont les nombres de sommets et de polygones après subdivision, respectivement (cette borne est contestée dans [Kara-89]).

• Segal et Sequin [SS-88] décrivent un algorithme qui élimine les intersections dans des ensembles de faces polygonales planes, arbitrairement complexes. En particulier, l'algorithme résout les opérations booléennes entre polyèdres dont la frontière est décrite par une hiérarchie *Solide/Faces/Contours/Arêtes/Sommets*.

A chaque sommet, arête ou face est associé une *tolérance* explicite qui évolue au cours de l'algorithme. La région de tolérance d'un sommet est une sphère centrée en ce sommet, celle d'une arête est un tronc de cône fermé par une demi-sphère à chaque extrémité et celle d'une face est une bande 3D qui inclut les régions de tolérance de tous les sommets de la face.

La région de tolérance associée à un point d'intersection entre deux arêtes a et b est la plus petite sphère contenant l'ellipse d'intersection entre les cônes de tolérance de a et b. L'équation du plan Γ d'une face f est calculée par la méthode des *moindres carrés* appliquée à tous les sommets de f ; la distance maximale à Γ de tous les sommets de f est prise comme *épaisseur* initiale de f.

Afin de contenir l'augmentation rapide des tolérances au fur et à mesure des intersections, une tolérance *maximale* est imposée, au delà de laquelle un message d'alerte est envoyé à l'utilisateur. Plus tard, Segal [Sega-90] introduit une gestion des tolérances plus sophistiquée, où un mécanisme de "retour arrière" (très coûteux) permet de réajuster les tolérances initiales.

• Hoffmann, Hopcroft et Karasick ([HHK-89], [Kara-89]) proposent un algorithme d'intersection qui est l'un des plus *robustes* face aux imprécisions numériques. Une présentation détaillée est donnée dans [Kara-89].

La structures de données utilisée ("Star-Edge") est particulièrement riche en informations d'incidence. La *frontière* d'un solide est définie par une liste de composantes connexes ("Shells"). Une *composante* est définie par une liste de faces (connexes), une liste d'arêtes et une liste de sommets. Une *face* est définie par un plan orienté, une liste de cycles d'arêtes orientées et une liste de sommets isolés dans la face. Un *cycle* est défini par une référence à la face à laquelle il appartient et une référence à une arête orientée quelconque de ce cycle. Une *arête* a est définie par une paire (v_1, v_2) de références à deux sommets et une liste d'arêtes orientées triées radialement autour de a, dans un plan de vecteur normal $v_2 - v_1$. Un *sommet* v est défini par ses coordonnées et, pour chaque face f incidente à v, une liste de toutes les arêtes orientées de f incidentes à v, triées radialement autour de v, dans le sens des aiguilles d'une montre. Une *arête orientée* $\vec{a}$ est définie par une référence à une arête a, un bit d'orientation, une référence au *cycle* qui contient $\vec{a}$ et une référence à l'arête orientée qui succède à $\vec{a}$ dans ce cycle. Enfin, un *sommet isolé* est défini par une référence à la face qui le contient et une référence au sommet correspondant.

L'algorithme intersecte chaque composante du solide A avec chaque composante du solide B, fusionne les composantes intersectantes, puis ajoute toutes les composantes de A (resp. B) qui sont contenues entièrement dans B (resp. A), sur la base d'un test d'inclusion. L'union ou la différence de deux solides se ramènent à l'intersection et à la complémentation. Le complémentaire d'un solide s'obtient en inversant les *normales* aux faces, l'*orientation* des arêtes et l'ordre *radial* de chaque liste d'incidence. L'essentiel de l'algorithme d'intersection réside dans la "fusion" de deux composantes connexes, qui est mené en trois grandes étapes :

1) pour chaque face f de A on calcule la *coupe* de B par le plan (orienté) Γ de f. Ceci est équivalent à une *classification* $M[\Gamma, B]$ qui partitionne Γ en trois types de régions 2D: Γ-*dans-B*, Γ-*hors-B* et Γ–*sur-B*. Cette partition est représentée par un *graphe* plan orienté G_Γ dont les arcs sont soit des arêtes de B, soit des segments d'intersection entre Γ et les intérieurs de faces de B. Le graphe G_Γ est construit en intersectant chaque arête de B avec Γ. Les régions Γ-*sur-B* sont distinguées en $(\Gamma$-*sur-B*$)^+$ et $(\Gamma$-*sur-B*$)^-$ selon que les faces correspondantes de B ont des normales dirigées dans le même sens ou dans le sens opposé à la normale au plan Γ. Les régions du type $(\Gamma$-*sur-B*$)^-$ ne peuvent pas appartenir à la frontière de $A \cap B$;

2) l'intersection de G_Γ avec la frontière de f consiste à combiner $M[\Gamma, f]$ (connue) avec la classification $M[\Gamma, B]$ induite par le graphe G_Γ. Pour cela, les arêtes de f et G_Γ sont intersectées deux à deux et les intersections sont utilisées pour subdiviser f en sous-faces homogènes par rapport à B. Parallèlement, certaines informations sont transférées à certaines faces de B, en vue de leur découpage ultérieur ;

3) sont ajoutées toutes les faces de A intérieures à B et toutes les faces de B intérieures à A.

La complexité de cet algorithme est en $O((m*n)\log(m*n))$, où m et n sont les nombres d'*arêtes orientées* de A et de B, respectivement.

La robustesse de l'algorithme tient à l'utilisation combinée des *tolérances* et du *raisonnement symbolique*, à travers un ensemble complexe de *tests d'incidence* [Kara-89].

D.1.4. Objets non régularisés

• Gursoz, Choi et Prinz [GCP-91] proposent un algorithme d'intersection sur les objets *hétérogènes* en dimension. Un objet est composé d'éléments de différentes dimensions: points, segments, polygones et polyèdres. L'algorithme opère en intersectant successivement toutes les paires d'éléments, dans un ordre particulier: sommet/sommet, sommet/arête, arête/arête ou sommet/face, arête/face et enfin face/face. Chaque *intersection* détectée entre deux éléments est enregistrée de telle manière que chacun des éléments référence l'autre. Une intersection peut générer de nouveaux sommets ou entraîner un découpage d'arêtes. Les cas particuliers sont pris en compte en associant à chaque élément une région de *tolérance* d'une certaine taille ε.

Une fois toutes les intersections calculées, les deux objets sont fusionnés en transformant chaque paire d'éléments coïncidents en une seule entité qui hérite de toutes les références attachées aux deux éléments. Ce processus est effectué itérativement, dans l'ordre décroissant des dimensions. L'algorithme s'achève par une phase de *sélection*.

Les auteurs présentent quelques résultats expérimentaux et concluent que leur algorithme est aussi robuste que les meilleurs algorithmes connus pour les solides eulériens ou pseodo non eulériens.

En particulier, en expérimentant le "test des deux cubes" décrit dans [LTH-86] avec une tolérance de $\varepsilon = 10^{-6}$, leur modeleur NOODLES est insensible aux angles α inférieurs à 0.00004 degrés et s'exécute correctement pour des angles supérieurs à 0.0964 degrés. Entre ces deux valeurs le modeleur avorte ou produit des résultats imprévisibles.

• Crocker et Reinke [CR-91] décrivent un autre algorithme opérant sur les objets *hétérogènes* en dimension, qui intège non seulement les éléments géométriques *linéaires* (segments, polygones et polyèdres), mais également des morceaux de *quadriques* simples (cylindres, sphères, cônes), des tores ou certaines *surfaces libres.*

Par ailleurs, Crocker et Reinke observent que le calcul de la frontière d'un solide booléen $A \% B$ est fondamentalement le même, quelque soit l'opération booléenne considérée: on calculera de toute façon tous les éléments de frontière nécessaires à chacun des solides $A \cap B$, $A \cup B$, $A - B$ et $B - A$. La distinction ne se fait que lors de la phase de *sélection* (voir C.1.1).

En partant de ce constat, les auteurs proposent un nouveau schéma de résolution d'arbres CSG, dans lequel certaines modifications de l'arbre peuvent être répercutées incrémentalement sur la représentation par frontière de l'objet final. Leur méthode, dite de *fusion-sélection*, consiste en deux grandes étapes :

Fusion :

a) calcul des intersections entre les éléments de frontière de toutes les primitives de l'arbre de construction et découper ces éléments le long de leus intersections ;

b) construction d'un *sur-ensemble* contenant une description complète des primitives initiales, toutes les intersections entre ces primitives et un *historique* décrivant l'appartenance des éléments découpés en termes de leurs primitives d'origine ;

c) calcul et stockage des relations d'adjacence entre tous les éléments du *sur-ensemble* ;

d) classification de chaque élément par rapport à toutes les primitives, au moyen l'*historique*.

Sélection : Marquer les éléments du *sur-ensemble* en fonction des opérations booléennes spécifiées dans l'arbre de construction.

Dans ce nouveau schéma, il devient possible d'ajouter de nouvelles primitives, de déplacer ou de supprimer des primitives existantes, sans réévaluer complètement l'arbre de construction. De même, on peut remplacer un opérateur booléen par un autre ou réaffecter tous les opérateurs, simplement en réexécutant l'étape de *sélection*. Le coût de mise à jour du *sur-ensemble* est inférieur à celui d'une réévaluation totale de l'arbre de construction.

L'inconvénient de cette méthode tient à la *taille* importante du *sur-ensemble* et à la nécessité d'un *rafraîchissement* après une opération de suppression d'une primitive.

Les auteurs n'abordent pas le problème des imprécisions numériques.

Enfin, on peut douter de l'intérêt pratique de cette méthode dans le domaine de la modélisation interactive d'objets solides: l'approche "Faire/Défaire" (ou "DO/UNDO") adoptée dans le modeleur DESIGNBASE [TSUC-86] est plus naturelle.

D.2. Méthodes indirectes

Ces méthodes effectuent les opérations booléennes sur des solides décrits dans une représentation autre que la représentation par frontière. Typiquement, ils exploitent une subdivision spatiale.

- Mantyla et Tamminen [MT-83] utilisent une structure de *répertoire* spatial (EXCELL) qui leur permet de définir des opérations booléennes localisées ;

- Navazo, Ayala et Brunet [NAB-86] représentent les polyèdres *eulériens* par la structure d'*arbre octal étendu* (ou "Extended Octree") qui trois nouveaux types d'octants "Face", "Arête" et "Sommet", qui contiennent soit une seule face, soit une seule arête et ses deux faces incidentes, soit un seul sommet et toutes les faces incidentes, respectivement.

L'avantage des arbres octaux étendus tient au fait qu'ils sont moins encombrants que les arbres octaux classiques et qu'ils peuvent être convertis exactement en représentations par frontière équivalentes.

Les opérations booléennes sur deux objets A et B décrits par des arbres octaux étendus sont linéaires en fonction de la taille des arbres. Elles s'effectuent récursivement par un parcours synchrone des deux arbres octaux: l'opération de base consiste à combiner deux octants terminaux de même taille, l'un de A et l'autre de B.

Le modeleur solide DMI utilise principalement la représentation par frontière: l'arbre octal est pris comme une représentation auxiliaire qui ne sert que pour la résolution des opérations booléennes.

Le problème des imprécisions numériques n'est évoqué que dans [ABBN-91]: les décisions géométriques locales (portant sur les octants) doivent tenir compte de la cohérence globale du solide. Pour cela, les auteurs limitent au maximum la redondance des données et des calculs et définissent un ensemble restreint de tests géométriques tenant compte de *tolérances* fixées globalement.

• Badouel et Hegron [BH-88] s'inspirent de l'arbre octal étendu défini dans [NAB-86] et proposent, pour les polyèdres *eulériens,* la structure plus compacte d'*arbre octal booléen*. Elle consiste à projeter un arbre CSG dans chaque octant d'un arbre octal étendu. L'arbre octal booléen permet d'optimiser l'évaluation d'arbres CSG en limitant les calculs géométriques à un espace minimal dans lequel les frontières des primitives se coupent effectivement et déterminent des portions qui contribuent à la frontière du solide final.

Les opérations booléennes (basées sur l'algorithme décrit dans [LTH-86]) sont simplifiées: elles se ramènent à des opérations entre octants.

Les auteurs n'abordent pas le problème de l'imprécision numérique et en particulier la propagation des erreurs d'un octant à l'autre.

• Thibault et Naylor [TN-87] utilisent les arbres binaires de partitionnement spatial (BSPT: "Binary Space Partitioning Trees") pour décrire les polyèdres réguliers. Dans un arbre BSP, les nœuds internes sont des plans orientés et les feuilles sont des régions convexes de l'espace. Chaque région est étiquetée pour indiquer si elle est à l'intérieur à l'extérieur du polyèdre, respectivement.

Les auteurs montrent: 1) comment convertir une représentation par frontière en un arbre BSP ; 2) comment effectuer une opération booléenne entre un arbre BSP et une représentation par frontière et produire un arbre BSP comme résultat; 3) comment évaluer un arbre CSG dont les primitives ont des frontières connue et produire un arbre BSP comme résultat.

Plus tard, Thibault, Naylor et Amanatides [NAT-90] introduisent des *tolérances* sur les plans de partitionnement.

• Tawfik [Tawf-91] exploite le concept d'*arrangement d'hyperplans* pour formuler un algorithme de résolution d'arbres CSG à primitives polyédriques. Il montre que son algorithme s'exécute en $O(n^3)$, où n est le nombre total de faces de toutes les primitives.

Cette borne, héritée de l'algorithme *incrémental* décrit dans [EOS-86], est *optimale* dans le pire des cas et améliore d'un facteur $\log(n)$ la complexité des meilleurs algorithmes connus. Cependant, la description des polyèdres en termes d'*arrangements* tient plus d'une *partition spatiale* que d'une représentation par frontière (en particulier, la notion de "face" doit être considérée dans le contexte des *arrangements*).

L'auteur ne mentionne pas d'implémentation pratique mais estime que l'algorithme est facile à mettre en œuvre et se prête bien à la modélisation interactive.

Enfin, il renvoie à la méthode "Simulation Of Simplicity" [EM-88] pour le traitement des cas particuliers et à la méthode "Epsilon Geometry" [GSS-89] pour la prise en compte des imprécisions numériques.

CONCLUSION

En dehors de toute considération commerciale, le foisonnement des systèmes de modélisation solide (Sect. B.2.4) témoigne objectivement de la difficulté de conception d'un modeleur universel, capable de répondre à tous les besoins des applications.

Dans son rapport sur l'état de l'art de la modélisation solide, Rossignac [Ross-92] relève au moins trois carences dans les systèmes de modélisation existants :

1) leur sensibilité aux imprécisions numériques, inévitables en arithmétique flottante: les concepteurs de modeleurs ont été nécessairement confrontés au problème des imprécisions numériques et ont dû le résoudre (au moins partiellement) par divers moyens. Malheureusement, les solutions ne sont pas toujours publiées dans la littérature (sans doute pour des raisons évidentes de confidentialité).

2) la pauvreté de leur domaine de représentation (aussi bien géométrique que topologique). Concernant la *géométrie* des solides, les modeleurs se limitent généralement aux objets polyédriques ou aux objets dont les surfaces sont soit des quadriques simples [Mill-88], soit certaines cubiques ([CK-83], [Casa-87]), soit des tores ou d'autres quartiques particulières [SA-85].

Les modeleurs surfaciques permettent une gamme plus riche de surfaces, qui répond mieux aux besoins des utilisateurs. Cependant, ces modeleurs ont évolué indépendamment des modeleurs volumiques et peu de développements ont été réalisés dans le sens d'une intégration (tout au plus, sait-on mélanger aux solides certaines quadriques ou cubiques simples).

Concernant la *topologie* des solides, le modèle classique des *r-ensembles* [Requ-80]) s'avère inadapté pour la description des assemblages d'objets hétérogènes en dimension et/ou en matériau [Ross-91]. De ce point de vue, le modèle *non eulérien* (ou "non manifold") semble s'imposer comme le nouveau modèle de base: en particulier il permet de mieux représenter les formes géométrique spécifiques à chaque domaine d'application [Wu-92]. D'autre part, le modèle *non eulérien* va dans le sens d'une meilleure communication entre les modeleurs volumiques et les modeleurs surfaciques.

3) leur utilisation difficile: malgré leurs interfaces graphiques sophistiquées, les modeleurs solides sont encore trop contraignants pour les utilisateurs finaux. Il y'a un besoin d'outils de plus haut niveau qui permettent de mieux saisir les intentions des utilisateurs. Deux nouvelles approches de modélisation semblent aller dans ce sens:

La première consiste en l'utilisation des *formes fonctionnelles* (ou "Features" [Ross-90], [Wu-92]), plus près de chaque catégorie d'utilisateurs (par exemple, un mécanicien préférerait manipuler directement des formes familières telles que congés, chanfreins, épaulements, queues d'aronde, ...).

La deuxième consiste en l'utilisation des systèmes de *contraintes* géométriques ([RR-86], [Verr-90], [Emme-91]). Cette approche est assez commode pour les utilisateurs, car elle permet de spécifier les dimensions et les positions des objets par un ensemble de contraintes géométriques (distances, angles, alignement, tangence, contacts ...). L'objet modélisé est la solution d'un système d'équations et d'inéquations posé par l'utilisateur et résolu par le logiciel de modélisation. Pour l'instant cette approche se limite aux systèmes relativement simples.

Chapitre II

UN ALGORITHME D'INTERSECTION DE POLYEDRES RATIONNELS DEFINIS PAR LEURS FRONTIERES

I. INTRODUCTION

Dans ce chapitre, nous présentons une version améliorée de l'algorithme de Michelucci [Mich-87], pour l'intersection de polyèdres *rationnels* représentés par leurs frontières. Cet algorithme se caractérise par trois points essentiels: 1) il utilise une structure de données simple, suffisamment générale pour permettre une représentation uniforme des cas particuliers, fort nombreux dans le domaine des solides *non eulériens* ; 2) il ne comporte qu'un unique cas particulier, dont le traitement ne perturbe pas le fonctionnement général ; 3) il produit des solides cohérents à partir d'autres solides cohérents: la cohérence topologique des résultats est garantie par l'emploi d'une *arithmétique rationnelle* particulière dont le principe et le coût seront discutés au chapitre III de cette thèse.

En Section II, nous précisons la classe des solides polyédriques acceptés par l'algorithme: les polyèdres *rationnels*. En Section III, nous décrivons la représentation par frontière utilisée en détaillant la structure de données sous-jacente. La simplicité et la généralité de cette structure de données sont mises en évidence. En Section IV, nous détaillons l'algorithme d'intersection de polyèdres: l'implémentation et les performances pratiques de cet algorithme sont abordées au Chapitre III de cette thèse. En Section V, nous montrons comment résoudre les opérations booléennes d'union ou de différence en termes d'intersection et de complémentation des solides. Nous indiquons comment convertir une représentation par arbre CSG en une représentation par frontière équivalente. En Section VI, nous tentons une analyse de la complexité de l'algorithme d'intersection. Nous concluons en VII.

II. SOLIDES POLYEDRIQUES RATIONNELS

Les objets solides considérés tout au long de ce chapitre sont essentiellement *polyédriques*. Nous les désignerons indifféremment par les termes "solides" ou "objets" polyédriques, voire simplement par "solides" ou "objets".

Formellement, un solide polyédrique est tout objet qui résulterait d'une séquence finie d'opérations booléennes régularisées appliquée à des *cubes*, ou alors le complémentaire régularisé d'un tel objet. Il s'agit donc d'un objet tridimensionnel régulier doté d'une frontière constituée de facettes planes, séparant l'espace en deux régions disjointes: l'intérieur et l'extérieur.

Un solide polyédrique peut être borné (i.e. de volume fini) ou non borné, auquel cas il est défini comme le complémentaire régularisé d'un solide borné. Dans tous les cas, la frontière est une surface fermée, bornée (i.e. d'aire finie), orientable, mais non nécessairement 2-*eulérienne* (voir Chapitre I, Section B.3.2). Un solide polyédrique peut avoir plusieurs composantes connexes et comporter un nombre quelconque d'anses (i.e. trous à travers le solide). La figure 1 montre un exemple de solide polyédrique possible.

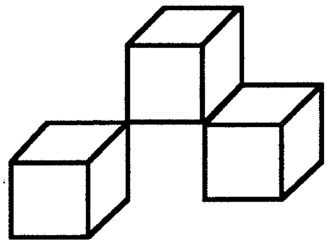


Figure 1 : Un exemple de solide polyédrique, pseudo eulérien. Il comporte une arête avec plus quatre faces incidentes et un sommet avec deux "cônes" de faces incidentes.

Par ailleurs, les solides sont supposés représentés par leur frontière, dans laquelle toutes les données géométriques (coordonnées des sommets et coefficients des équations des plans de faces) sont connues comme des nombres *rationnels*, fidèlement représentés. De tels solides seront dits *rationnels* : ils sont toujours numériquement cohérents. Par exemple, les coordonnées rationnelles d'un sommet vérifient exactement l'équation (à coefficients rationnels) du plan de chaque face incidente au sommet. La conversion d'un solide *flottant* cohérent en un solide *rationnel*, ainsi que la conversion inverse, seront abordés au chapitre III de cette thèse.

III. REPRESENTATION DES POLYEDRES PAR LEURS FRONTIERES

Nous allons décrire une représentation par frontière qui intègre, naturellement, les nombreux cas possibles pour les solides polyédriques. L'élément de base de cette représentation est le *pan*, généralisation tridimensionnelle de la notion de *brin* des *cartes planaires* [Mich-87]. Intuitivement, un pan est une portion de face, incidente à une arête donnée d'un côté bien déterminé de cette arête.

III.1. Notion de pan

Soit $a = [p, q]$ une arête située sur le plan Q d'une face f d'un solide, orientée de façon que l'extrémité p précède q dans l'ordre lexicographique (ou *xyz*-ordre) de $\Re^3$. Pour une certaine valeur s (= ±1), le triplet (a, f, s) définit un *pan* de f, incident à a et de *sens* s : la droite (p, q) partitionne Q en deux demi-plans, et la valeur de s permet d'identifier précisément le demi-plan qui contient la face f, au voisinage immédiat de l'arête a (Fig. 2). La notion de pan, introduite par Michelucci [Mich-87], permet une représentation explicite d'un *voisinage* par rapport à une face, de tout point intérieur à une arête de cette face [RV-85].

Soient $\vec{N}$ le vecteur *normal* à Q, dirigé de l'intérieur vers l'extérieur du solide, et $\vec{E}$ le "vecteur-arête" de a, dirigé de p vers q ($\vec{E} = \vec{pq}$). Le produit vectoriel $\vec{V}$ défini par $\vec{V} = s * (\vec{E} \wedge \vec{N})$, rapporté à un point quelconque de $]p, q[$ est par construction dans le plan Q, orthogonal à l'arête a et dirigé vers l'intérieur de f, localement à a (Fig. 2a). Ce vecteur sera appelé "vecteur indicateur" du pan (a, f, s), ou plus simplement "vecteur-pan". Lorsque la face f considérée est située de part et d'autre de l'arête a (Fig. 2b), cette arête est représentée avec deux pans $(a, f, \pm 1)$, de sens opposés (cette situation apparaît typiquement dans les solides *pseudo eulériens*).

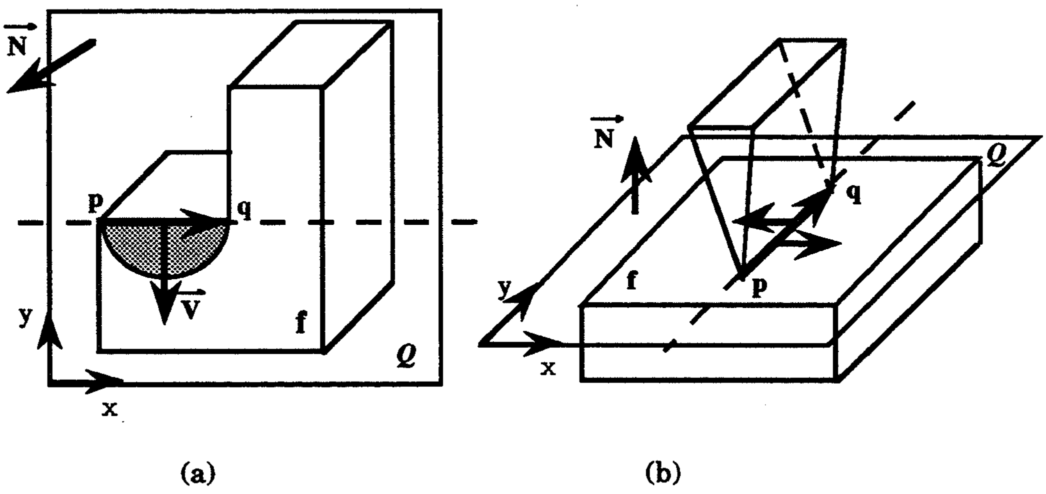


Figure 2. Caractérisation d'un pan: **a)** pan incident à une arête "eulérienne", **b)** paire de pans incidents à une arête non "eulérienne"

III.2. Notion de face

Dans notre représentation par frontière, une *face* est le plus grand sous-ensemble de la frontière d'un solide, porté par un même plan orienté. Pour cette raison, nous dirons que les faces sont *maximales*. Cette définition autorise des faces arbitrairement complexes: connexes ou non, convexes ou non, et comportant un nombre quelconque de trous (Fig. 3). Une face est définie simplement par une équation de plan et une liste de tous les pans qui la constituent. Les divers contours de la face ne sont pas décrits explicitement.

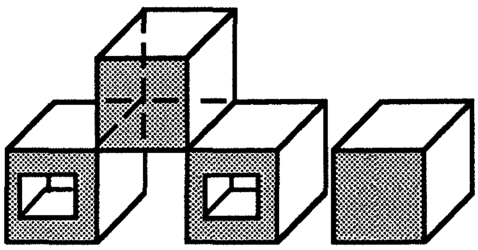


Figure 3. Le solide de la figure possède exactement 20 faces. La face avant (en gris) est constituée de 24 pans.

Classification point/face

Une face maximale possède un *intérieur* et un *extérieur*, que l'on peut caractériser, par exemple, en appliquant le principe de *parité* (Fig. 4).

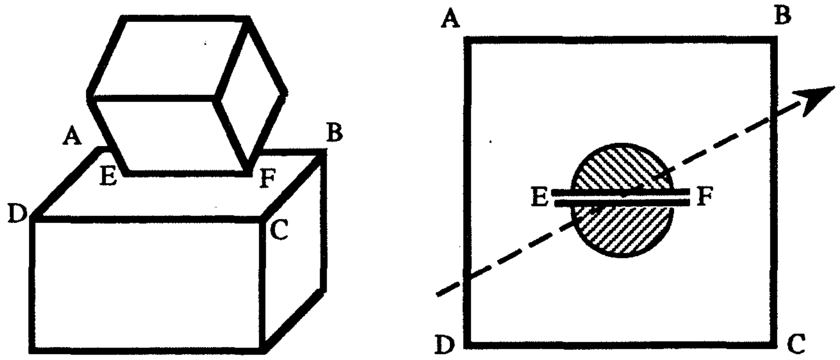


Figure 4. Dans l' objet de gauche, la face rectangulaire ABCD est représentée par six pans dont deux de sens opposés, incidents à l'arête non eulérienne EF (à droite). Lors de la classification d'un point M par rapport à cette face, l'intersection entre un rayon issu de M et l'arête EF est "comptée" deux fois, une fois pour chaque pans incident.

III.3. Structure de données

La structure de données utilisée pour décrire la frontière d'un solide polyédrique rationnel est constituée des enregistrements suivants :

Solide

- un booléen *borné* ? indiquant si le solide est borné ou non borné ;
- la liste de toutes les *faces* du solide, dans un ordre quelconque ;
- la liste de toutes les *arêtes* du solide, dans un ordre quelconque ;
- une boîte englobante du solide (voir IV.1) ;

Face

- l'équation du plan de la face, sous forme d'un quadruplet (a, b, c, d) de *rationnels*. Le vecteur $N = (a, b, c)$ donne une *normale* à la face, dirigée de l'intérieur vers l'extérieur du solide. Le quadruplet est normalisé de façon que la première composante non nulle de N soit égal à +1 ou -1 ;
- la liste de tous les *pans* de la face, dans un ordre quelconque ;
- une boîte englobante de la face (voir IV.1) ;

Pan

- un pointeur sur la *face* du pan (un pan appartient à une face unique) ;
- un pointeur sur l'*arête* du pan (un pan appartient à une arête unique) ;
- un entier valant ±1, donnant le *sens* du pan ;

Arête

- l'extrémité *gauche* de l'arête, représentée par un triplet p de coordonnées rationnelles.
- l'extrémité *droite* de l'arête, représentée par un triplet q de coordonnées rationnelles ;
- la liste des pointeurs sur tous les pans incidents à l'arête, dans un ordre quelconque ;
- une liste, initialement vide, de *sous-arêtes* qui sera mise à jour lors de l'intersection de deux solides (voir IV.2.1).

Graphe d'adjacence

Chaque arête est unique dans la structure de données ; elle est toujours stockée comme un couple (p, q), où p précède q dans l'ordre *lexicographique* de $\Re^3$. Les sommets d'arêtes n'ont pas besoin d'être uniques ; ce sont de simples triplets de coordonnées.

Le graphe d'adjacence associé à cette représentation par frontière est donné en Figure 5, ci-dessous :

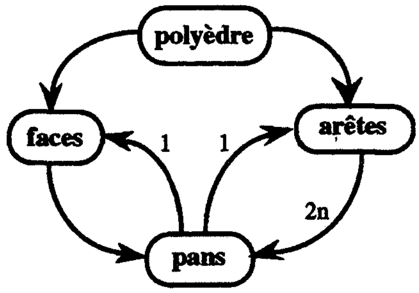


Figure 5. La frontière d'un solide polyédrique est définie par une liste de faces et une liste d'arêtes. Une face est définie par un plan orienté et une liste de pans. Une arête est définie par un couple de points et une liste de pointeurs sur tous les pans incidents (en nombre pair). Un pan est défini un pointeur sur une face, un pointeur sur une arête et un sens (±1).

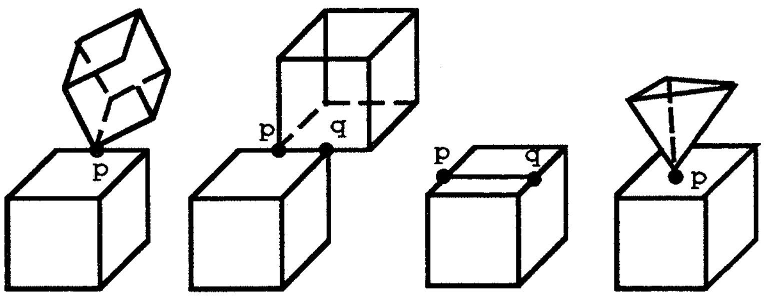
III.4. Contraintes de cohérence

Pour constituer la frontière d'un solide cohérent, les divers éléments de la structure de données doivent vérifier les trois conditions ci-dessous (Fig. 6) :

C1) deux arêtes distinctes ne peuvent se chevaucher, ni se couper en un point autre qu'une extrémité commune ;

C2) les faces sont *maximales* et deux faces distinctes ne peuvent s'intersecter que le long d'arêtes explicitement décrites dans la structure de données, sauf dans le cas particulier (unique) de la figure 6(d) ;

C3) une arête doit avoir un nombre *pair* de pans incidents et il n'existe pas d'arête "inutile" ayant seulement deux pans incidents appartenant à une même face (Fig. 6c).



(a) (b) (c) (d)

Figure 6. a) Deux arêtes ne peuvent se couper en un point intérieur à l'une d'elles: sur l'exemple, le solide doit avoir exactement 25 arêtes, dont 5 arêtes sont incidentes au point p. **b)** Deux arêtes ne peuvent se chevaucher: sur l'exemple, le solide doit avoir exactement 25 arêtes (dont [p, q]) ; en particulier, quatre arêtes sont incidentes au point p et quatre arêtes sont incidentes au point q. **c)** l'arête [p, q] est "inutile" car elle ne possède que deux pans incidents, appartenant à une même face. **d)** deux faces peuvent partager un sommet, isolé à l'intérieur de l'une d'elles. C'est là l'unique cas particulier de notre représentation par frontière.

III.5. Propriétés de cette représentation

La structure de données ci-dessus est extrêmement simple, comparée aux diverses variantes de la "Winged-Edge" [Baum-75]. Elle n'explicite pas tous les types possibles de relation d'incidence entre les éléments de frontière [Weil-85]. En particulier, elle ne possède pas de "sommets" uniques, auxquels on pourrait rattacher des liste d'arêtes incidentes. Elle ne donne pas directement les composantes connexes ("Shells") de la frontière du solide. Elle ne décrit pas les divers contours d'une face par des cycles orientés d'arêtes ("Loops").

En particulier, les contours délimitant les "trous" dans une face ne sont pas représentés explicitement. Bien entendu, toutes ces informations peuvent être explicitées au besoin, soit par un parcours simple de la structures de données, soit par des calculs géométriques simples. Une face *maximale* est caractérisée simplement par un plan orienté et une liste de pans. Nous verrons que ce type de faces n'est en fait qu'un artifice de représentation qui simplifie l'algorithme d'intersection, comme dans le modeleur solide PADL-2 [RV-85]). Finalement, la frontière d'un solide est totalement déterminée par la liste de tous ses pans.

A notre connaissance, cette structure de données est l'une des plus "pauvres" (en relations d'incidence) qui soit utilisable dans un domaine aussi complexe que la modélisation solide (voir [Ala-91] et [Ala-92]). Au passage, notons que notre représentation par frontière a exactement le même domaine de modélisation (à savoir, les polyèdres pseudo non eulériens) que la "Star-Edge" de Karasick [Kara-89], particulièrement riche et explicite.

Par sa simplicité, notre structure de données permet de représenter et de traiter de manière uniforme tous les cas particuliers, à l'exception d'un seul (un irréductible) (Fig. 6d), traité en Section IV.6.

IV. ALGORITHME D'INTERSECTION DE POLYEDRES RATIONNELS

Nous allons maintenant présenter en détail une version améliorée de l'algorithme proposé dans [Mich-87] pour l'intersection de polyèdres. Les polyèdres considérés sont représentés par leurs frontières décrites en arithmétique rationnelle. Les opérations booléennes d'union ou de différence se ramènent à l'intersection et à la complémentation, via les lois de De Morgan (voir Sect. V).

Tous les calculs numériques nécessaires à l'algorithme sont supposés s'effectuer sans erreurs, de sorte qu'aucune *incohérence topologique* ne puisse avoir lieu du seul fait de l'imprécision numérique. L'algorithme arrive toujours à son terme et produit des solides cohérents à partir d'autres solides cohérents.

A priori, une arithmétique rationnelle est nécessaire. De fait, l'algorithme fait appel à une arithmétique rationnelle optimisée qui n'effectue que les calculs exacts strictement nécessaires. Le reste des calculs sont effectués en arithmétique flottante. Le principe et le coût de cette arithmétique seront présentés au chapitre III de cette thèse.

Rapide survol

L'algorithme suit le schéma général décrit dans [RV-85] ; il comporte cinq grandes étapes, plus ou moins entrelacées :

Etape 1: Prétraitement rapide pour rechercher les paires (f, g) de faces (f de A et g de B) susceptibles de s'intersecter. Cette liste comporte au moins toutes les paires de faces qui s'intersectent effectivement. Parmi toutes les techniques possibles, nous nous sommes contentés de la technique classique des boîtes englobantes.

Etape 2: Intersection des faces candidates. Toutes les intersections trouvées sont stockées de manière adéquate en vue d'un traitement ultérieur.

Etape 3: Eclatement des frontières de A et B, le long de leurs intersections. Une structure de données intermédiaire est utilisée pour stocker un *sur-ensemble* de la frontière de $A \cap B$, constitué d'éléments non intersectants, issus de la frontière de A et/ou de B.

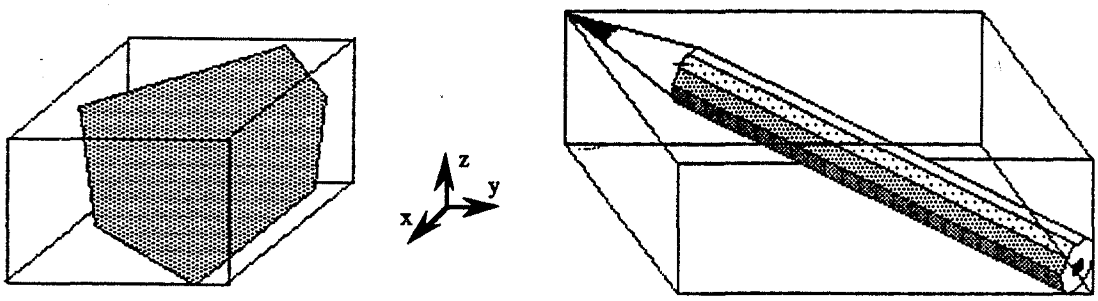
Etape 4: Classification [Tilo-80] de chaque élément du *sur-ensemble* par rapport à $A \cap B$ et sélection des éléments appartenant à la frontière du solide $A \cap B$;

Etape 5: Construction effective de la structure de données décrivant la frontière de $A \cap B$, suivie d'un post-traitement simple qui élimine les éventuelles arêtes "inutiles", produites par l'algorithme (Fig. 6c).

IV.1. Recherche des faces candidates

Pour éviter en pratique les $O(m*n)$ tests d'intersection entre chacune des m faces de A et chacune des n faces de B, nous avons utilisé le test, très simple, des boîtes englobantes, autour des solides et des faces ([LTH-86], [SS-88]).

La boîte la plus simple pour un solide ou une face consiste en la *boîte englobante des sommets* : c'est le plus petit parallélépipède dont les faces sont parallèles aux plans de coordonnées (ou boîte isothétique), qui contient tous les sommets du solide ou de la face (Fig. 7).



(a) **(b)**

Figure 7. Boîte isothétique: **a)** d'une face, **b)** d'un solide.

Notons que de telles boîtes n'englobent pas nécessairement les solides correspondants car ces derniers peuvent être non bornés (par exemple, le complémentaire d'un cube). On rappelle que les seuls solides non bornés considérés ici sont définis comme étant les complémentaires de solides bornés).

Le test des boîtes englobantes est utilisé comme un prétraitement totalement indépendant de l'algorithme d'intersection lui-même. Il consiste simplement à déterminer la liste L_C de toutes les paires de faces f de A et g de B, dont les boîtes ne sont pas *disjointes*.

Le pseudo-code ci-dessous donne le schéma classique de ce type de prétraitement : le test d'intersection de deux boîtes est trivial.

```
FacesCandidates (A, B: Polyèdre)
début
        Lc := Ø; # liste des paires de (pointeurs sur les) faces candidates
        si non BoîtesDisjointes (A->boîte, B->boîte) alors
                pour chaque face f de A faire
                        si non BoîtesDisjointes (f->boîte, B->boîte) alors
                                pour chaque face g de B faire
                                        si non BoîtesDisjointes (f->boîte, g->boîte) alors
                                                AjouterPaire ((f,g), Lc) ;
                                        fsi
                                fait
                        fsi
                fait
        fsi ;
        rendre Lc ;
fin
```

IV.2. Intersection des faces candidates

Dans cette deuxième étape de l'algorithme, toutes les paires de faces candidates sont intersectées. Toutes les informations nécessaires à l'éclatement des frontières de A et de B sont générées et stockées en vue de leur utilisation ultérieure. L'algorithme tient à jour une liste globale pour toutes les *arêtes d'intersection* et, pour chaque arête a de A ou de B, une liste de points qui sont les extrémités *gauches* des futures *sous-arêtes* de a.

IV.2.1. Définitions

Arêtes d'intersection

Une *arête d'intersection* est un segment qui résulte de l'intersection "stricte" entre l'intérieur d'une face f de A et l'intérieur d'une face g de B (Fig. 8). Un tel segment ne se trouve donc ni sur une arête de A, ni sur une arête de B. Chaque arête d'intersection est stockée sous forme d'un enregistrement (p, q, f, g), où p et q sont des triplets de coordonnées rationnelles, et où f et g sont des pointeurs sur les deux faces qui s'intersectent.

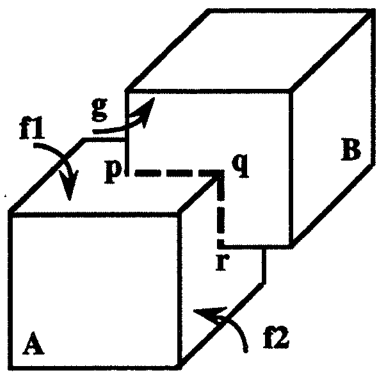


Figure 8. Les segments [p, q] et [q, r] sont des arêtes d'intersection pour les paires de faces (f_1, g) et (f_2, g), respectivement. Sur l'exemple, les deux cubes A et B ont exactement 6 arêtes d'intersection.

Sous-arêtes homogènes

Pour chaque arête a d'un solide, seront déterminés tous les points d'intersection entre a et toutes les faces de l'autre solide. Ces points d'intersection seront plus tard utilisés pour découper a en sous-arêtes *homogènes* par rapport à l'autre solide. Une sous-arête homogène de a est le plus grand segment ouvert de a qui est inclus tout entier soit à l'intérieur, soit à l'extérieur, soit sur la frontière de l'autre solide (Fig. 9).

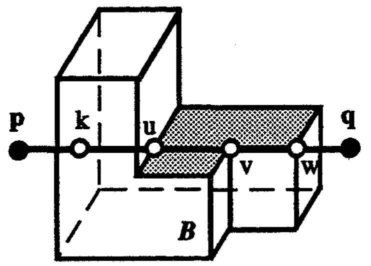


Figure 9. L'arête [p, q] du solide A (non représenté) intersecte la frontière de B aux points k, u, v et w. En triant ces points dans l'ordre lexicographique le long de la droite (p, q), on obtient toutes les sous-arêtes homogènes [p, k], [k, u], [u, v], [v, w] et [w, q] de l'arête [p, q]. La sous-arête homogène [u, v] est mémorisée avec un pointeur sur la face g (grisée) de B, qui la contient. La sous-arête [v, w] (qui se trouve sur une arête de B) n'a pas besoin de traitement particulier.

Chaque sous-arête de a est stockée sous forme d'un couple (p, g), contenant le triplet p de coordonnées rationnelles de l'extrémité *gauche* de la sous-arête et un pointeur sur l'éventuelle face g de l'autre solide, à l'intérieur de laquelle est incluse la sous-arête. Le cas des arêtes qui se chevauchent (par exemple, [v, w] en Fig. 9)) n'a pas besoin de représentation particulière (contrairement à [Mich-87]).

Faces transversales, coplanaires ou parallèles

L'algorithme d'intersection distingue trois types de paires de faces candidates: deux faces candidates f et g seront dites *transversales*, *coplanaires* ou parallèles, selon que les plans de f et g se coupent le long d'une droite D, sont géométriquement confondus ou sont strictement parallèles, respectivement. Cette classification est possible du moment que les solides considérés sont des polyèdres *rationnels* et tous les calculs géométriques sont effectués, sans erreur aucune, au moyen d'une *arithmétique rationnelle*. A priori, les faces *parallèles* peuvent être éliminées de la liste des faces candidates, de sorte que l'algorithme n'a plus à traiter que les faces *transversales* ou *coplanaires*.

IV.2.2. Intersection de deux faces transversales

Etant données deux faces transversales f et g, la procédure de calcul de $f \cap g$ doit tenir compte de tous les cas particuliers, impérativement. Cette procédure est l'une des plus importantes dans notre algorithme d'intersection de polyèdres. Les faces étant arbitrairement complexes, les cas particuliers deviennent nombreux et délicats.

Cependant, $f \cap g$ étant inclus dans la droite D d'intersection entre les plans de f et g, on peut procéder en trois étapes simples:

1) calcul de $f \cap D$, qui produit une première liste de segments de D ;

2) calcul, analogue, de $g \cap D$, qui produit une deuxième liste de segments de D ; et 3) *fusion* des deux listes, qui produit $f \cap g$ sous forme d'une autre liste de segments de D.

Formellement, l'intersection de f et g est équivalente aux deux *classifications droite/polygone*, M[D, f] et M[D, g]), et à leur *fusion* M[D, f] $\otimes$ M[D, g] (voir "Membership Classification" [Tilo-80]). Cependant, l'intersection d'une face (disons f) et de D revient à l'intersection de f avec le plan Q de l'autre face (g), de sorte que la droite D n'a pas besoin d'être explicitée.

IV.2.2.1. Intersection face /plan

Soient f une face de A et Q le plan d'une face transversale, appartenant à B. Un point d'intersection entre une arête a de f et Q sera appelé *contact* entre a et Q ; il peut être soit un point *intérieur* à a, soit une extrémité de a.

Contacts face/plan

Un contact entre une arête a de f et le plan Q de g est stocké comme un enregistrement à quatre champs (*point*, *indice*, *tangente*, *sécante*), définis comme suit (Fig. 10):

- le champ *point* mémorise le triplet de coordonnées rationnelles du contact ;

- le champ *indice* vaut 1 si a possède une extrémité de *puissance strictement positive* par rapport à Q, 0 sinon. Cette convention de signe est choisie arbitrairement ;

- dans le cas où les deux extrémités de a se trouvent de part et d'autre de Q, le champ *sécante* reçoit un pointeur sur a. Cette information n'est pas prévue dans [Mich-87]: on verra qu'elle permet d'éviter la création de sommets "inutiles" lors de l'intersection de deux polyèdres. ;

- dans le cas où les extrémités de a se trouvent toutes les deux dans Q, on considère qu'il y a <u>deux</u> contacts: un contact à l'extrémité *gauche* de a, dont le champ *tangente* reçoit un pointeur sur a ; et un autre contact à l'extrémité *droite* de a, dont le champ *tangente* vaut *nil* (qui désigne l'absence d'arête).

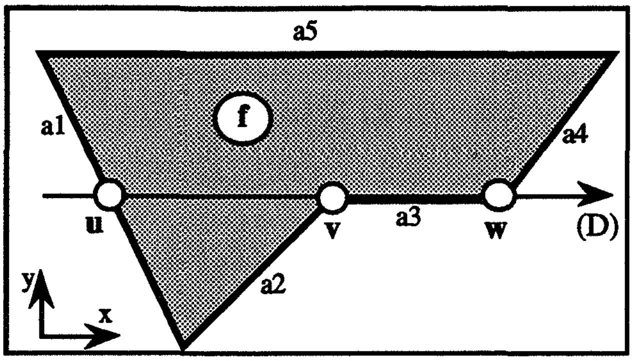


Figure 10. La face f présente 5 contacts avec le plan Q (droite D sur la figure), qui sont: c_1 = (u, 1, nil, a_1), c_2 = (v, 0, nil, nil), c_3 = (v, 0, a_3, nil), c_4 = (w, 0, nil, nil), c_5 = (w, 1, nil, nil). On note que c_1 porte une sécante a_1 ; c_3 porte une tangente a_3 ; c_2 et c_3 coïncident au point v ; c_4 et c_5 coïncident au point w. Sur l'exemple, on suppose que les points au "dessus" du plan Q (la droite D de la figure) ont des puissances positives.

Notons que chaque enregistrement "contact" possède deux champs "arête" distincts (tangente et sécante): pour un contact donné, au plus l'un de ces deux champs est différent de *nil*.

Calcul des contacts

Le pseudo-code ci-dessous montre comment calculer tous les contacts entre une face f et un plan Q. Les contacts sont mémorisés dans une liste intermédiaire L_f.

```
IntersectionFacePlan (f : Face ; Q : Plan)
début
  Lf := Ø ;
  pour chaque pan de f faire
        # soit a = [p, q,] l'arête du pan
        α := PuissancePointPlan (p, Q) ;
        β := PuissancePointPlan (q, Q) ;
        si (α > 0 et β < 0) ou (α < 0 et β > 0) alors
              # calculer Ω = ]p, q[ ∩ Q
              λ := α / (α - β) ; Ω := p + λ*(q - p) ;
              CréerContact (Ω, 1, nil, a)
        sinon si (α = 0 et  β = 0) alors
              # [p, q] ⊂ Q
              CréerContact (p, 0, a, nil) ;
              CréerContact (q, 0, nil, nil) ;
        sinon si (α = 0 et β > 0) alors CréerContact (p, 1, nil, nil) ;
        sinon si (α = 0 et β < 0) alors CréerContact (p, 0, nil, nil) ;
        sinon si (α > 0 et β = 0) alors CréerContact (q, 1, nil, nil) ;
        sinon si (α < 0 et β = 0) alors CréerContact (q, 0, nil, nil) ;
        sinon # ne rien faire
  fait ;
  TrierContacts (Lf) ; rendre Lf ;
fin
```

Une fois que tous les contacts f/Q ont été déterminés, ils sont ordonnés lexicographiquement, sur leurs triplets de coordonnées (procédure *TrierContacts*). Les contacts consécutifs, de mêmes coordonnées, correspondent nécessairement à une extrémité d'arête ; ces contacts sont fusionnés en un seul contact, qui hérite des mêmes coordonnées, d'un indice *cumulé*, et de l'éventuelle *tangente* ou *sécante* portée par l'un des contacts fusionnés.

Classification droite/face

Maintenant, deux contacts consécutifs i et j déterminent, implicitement, un segment $[i, j]$ de D, de classification *constante* par rapport à f: le segment ouvert $]i, j[$ se trouve tout entier soit à l'intérieur, soit à l'extérieur, soit sur la frontière (i.e. sur une arête) de f.

La classification se fait par une simple application du principe de *parité* : si i porte une *tangente* a alors le segment est *Sur* f et l'arête a est nécessairement confondue avec $[i, j]$; sinon le segment est *Dans* f (resp. *Hors* de f) si la *somme* des *indices* de tous les contacts qui précédent j est *impaire* (resp. *paire*).

Le champ *indice* du contact i est réutilisé pour mémoriser le résultat (*Dans*, *Sur* ou *Hors*) de la classification (Fig. 11).

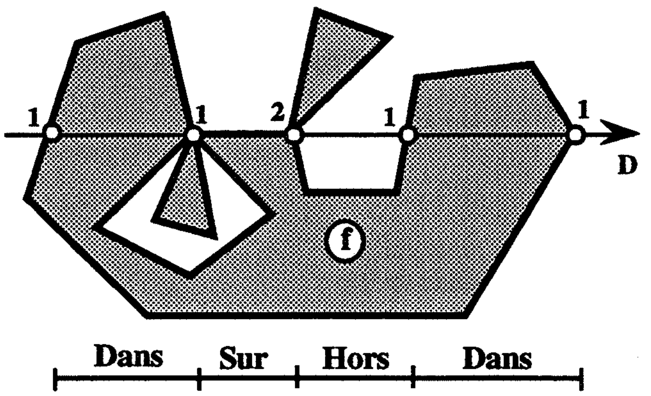


Figure 11. Intersection face/droite. La figure montre les indices après fusion des contacts confondus et le résultat de la classification (en bas).

IV.2.2.2. Fusion de deux intersections face/plan

Après avoir calculé successivement $f \cap D$ et de $g \cap D$, les deux listes de contacts L_f et L_g résultantes sont maintenant fusionnées. Le but est de déterminer les plus grands segments de D qui sont de classification *constante* par rapport à $f \cap g$; ils s'obtiennent par un parcours simultané des deux listes.

Soit $[p, q]$ le segment en cours de classification, l'extrémité p correspond à un certain contact i de l'une des faces (disons f) et éventuellement à un autre contact i' de l'autre face, confondu avec i. La classification de $[p, q]$ par rapport à $f \cap g$ se déduit de ses classifications par rapport à chacune des deux faces. Le seul calcul géométrique nécessaire est le test de coïncidence de deux points.

Au cous du parcours des deux listes, on se ramène itérativement à l'un des trois cas suivants :

1) si le segment $[p, q]$ est à la fois *Dans* f et *Dans* g alors il correspond à une *arête d'intersection* entre f et g (Fig. 12). Les extrémités p et q étant connues, un nouvel enregistrement (p, q, f, g) est ajouté à la liste (globale) des arêtes d'intersection.

Notons que f et g peuvent avoir plusieurs arêtes d'intersection, du fait que les faces ne sont pas nécessairement *convexes*, ni *connexes*. Cependant, chaque arête d'intersection est détectée une et une seule fois par l'algorithme.

Si le contact i en p porte une *sécante* a alors un enregistrement (p, nil) est ajouté à la liste des *sous-arêtes* de a. De même si un autre contact i' existe en p et porte une *sécante* b alors une *sous-arête* (p, nil) est ajoutée à b.

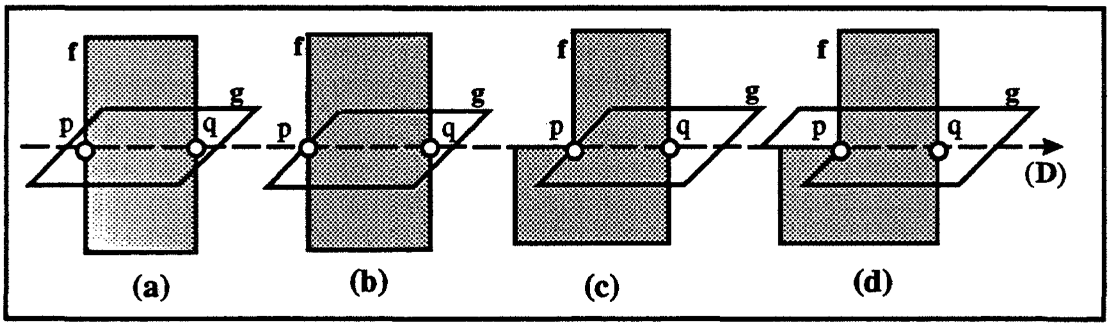


Figure 12. Détection d'une *arête d'intersection* [p, q] entre deux faces transversales f et g. **a)** L'extrémité p est un point d'intersection entre une arête *a* de f et l'intérieur de g. **b)** p est un point d'intersection entre une arête *a* de f et une arête *b* de g. **c)** p est un sommet de f, qui se trouve à l'intérieur d'une arête *b* de g. **d)** p est un sommet commun à f et à g. Dans tous les cas, une nouvelle arête d'intersection (p, q, f, g) est créée. Dans les cas (a), (b) et (c) une sous-arête (p, nil) est ajoutée à *a* et *b*.

2) si $[p, q]$ est *Sur* l'une des faces (disons f) et *Dans* l'autre face g, alors $]p, q[$ est inclus à la fois dans l'intérieur de g et dans une certaine arête a de f portée par D. Cette arête est mémorisée dans le champ *tangente* du contact i de f qui se trouve soit en p, soit immédiatement à gauche de p, sur la droite D. Dans les deux cas, une *sous-arête* (p, g) est ajoutée à cette arête a. Si, au point p, il existe également un contact i' de g, qui porte une *sécante* b, alors une *sous-arête* (p, nil) est ajoutée à b (Fig. 13).

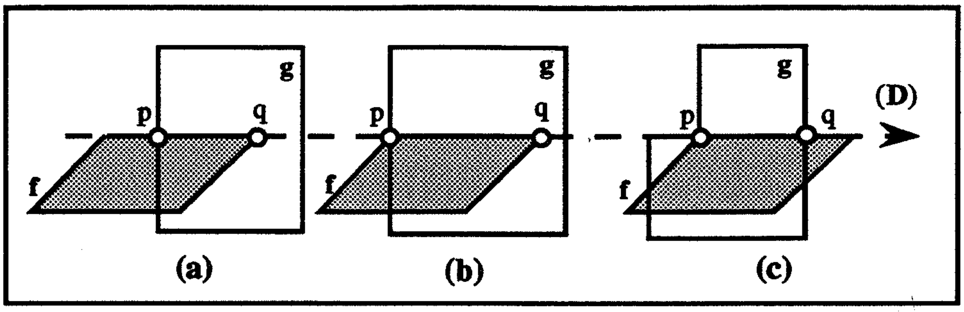


Figure 13. Détection d'une sous-arête [p, q] d'un solide, incluse dans une face de l'autre solide. **a)** p est un point d'intersection entre une arête *a* de f et une arête *b* de g. **b)** p est un sommet d'une arête *a* de f, qui se trouve à l'intérieur d'une arête *b* de g. **c)** p est un sommet commun à une arête *a* de f et à une arête *b* de g. Dans tous les cas, une sous-arête (p, g) est ajoutée à *a* et une sous-arête (p, nil) est ajoutée à *b*.

3) si $[p, q]$ est à la fois *Sur f* et *Sur g*, alors il y a deux arêtes a (de f) et b (de g), portées par D, qui se chevauchent partiellement ou totalement le long du segment $[p, q]$. L'une de ces arêtes (disons a) commence au point p, elle est mémorisée dans le champ *tangente* du contact i de f qui se trouve en p ; l'autre arête b est mémorisée dans le champ *tangente* du contact i' de g qui se trouve soit en p, soit immédiatement à gauche de p, sur la droite D. Dans tous les cas, une *sous-arête* (*p*, *nil*) est ajoutée à chacune des deux arêtes (Fig. 14).

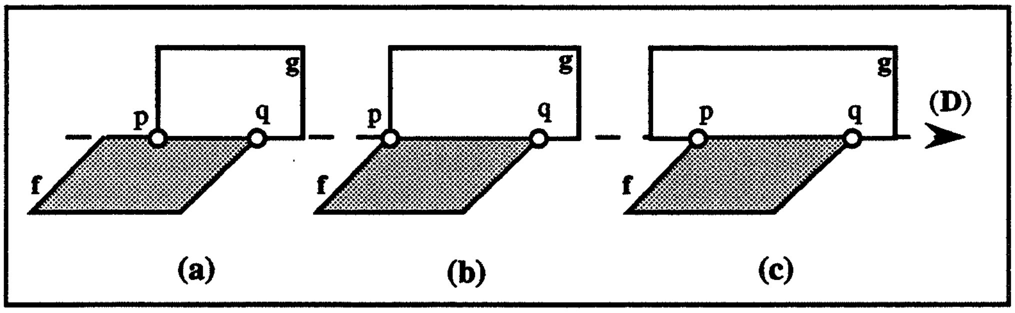


Figure 14. Détection d'un segment [p, q] né d'un chevauchement entre une arête *a* de f et une arête *b* de g. **a)** p est l'extrémité gauche de *b*. **b)** p est une extrémité gauche commune à *a* et à *b*. **c)** p est l'extrémité gauche de *a*. Dans tous les cas, une sous-arête (p, nil) est ajoutée à *a* et à *b*.

Remarques

• Un contact peut être soit un point d'intersection "stricte" entre une arête d'un solide et le contour ou l'intérieur d'une face de l'autre solide, soit une extrémité d'une arête de l'un et/ou l'autre solide. L'algorithme ne considère aucun de ces cas comme un cas particulier .

• Un même point d'intersection p entre une arête a de A et les diverses faces g de B est calculé plusieurs fois, une fois pour chaque face f de A incidente à a. Cependant, un et un seul enregistrement *sous-arête* (p, g) est ajouté à a, dont le champ "face" est soit *nil*, soit un pointeur sur l'unique face g de B qui contient la sous-arête en question.

• Enfin, contrairement à [Mich-87], le découpage effectif des arêtes initiales de A et B n'est pas effectué au fur et à mesure de l'intersection des faces. La liste complète des *sous-arêtes* d'une arête donnée ne sera disponible qu'une fois que toutes les faces auront été intersectées.

IV.2.3. Intersection de deux faces coplanaires

L'intersection de deux faces *coplanaires* f et g est une procédure délicate, même en l'absence d'erreurs numériques. En effet, les faces ne sont pas nécessairement connexes, ni convexes et peuvent comporter une hiérarchie de nombreux trous. Cependant, l'algorithme d'intersection n'a absolument pas besoin de traiter les faces coplanaires. De telles faces ne peuvent avoir d'*arêtes d'intersection* et toutes les *sous-arêtes* d'une arête a de f, qui résulteraient de l'intersection de a avec le contour ou l'intérieur de g seront (ou ont déjà été) détectées lors de l'intersection de g avec une autre face f' de A, incidente à a et non coplanaire avec g (Fig.15). L'existence de cette face f' est garantie par le fait que les solides n'ont pas d'arête "inutile", ayant seulement deux pans incidents appartenant à la même face (Sect. III.4).

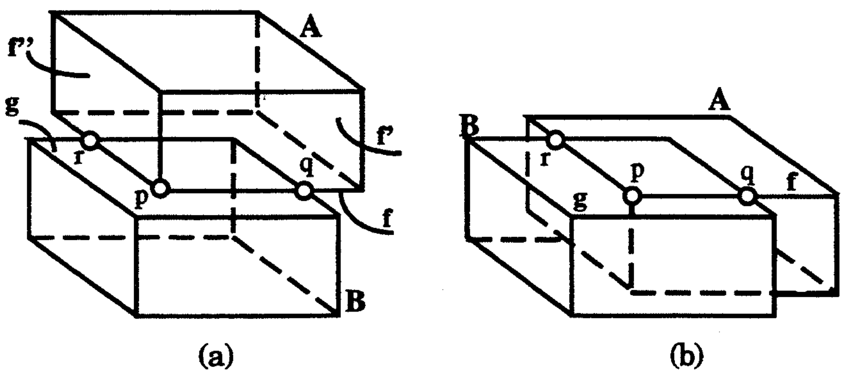


Figure 15. Faces coplanaires. **a)** La face inférieure f du cube A est coplanaire avec la face supérieure g du cube B ; ces deux faces n'ont pas besoin d'être intersectées car les sous-arêtes [p, q] et [p, r] seront détectées lors de l'intersection de g et des faces f' et f'', respectivement. **b)** Même remarque pour les deux faces supérieures f et g des cubes A et B, qui sont coplanaires et de même orientation.

IV.3. Eclatement des frontières des solides opérandes

A ce stade de l'algorithme, toutes les faces candidates ont été intersectées. Dans cette troisième étape, toutes les informations collectées sont utilisées pour éclater les frontières de A et de B en éléments non-intersectants.

Une structure de données intermédiaire est générée pour décrire un *sur-ensemble* de la frontière de $A \cap B$, qui regroupera: 1) toutes les arêtes *homogènes* issues du découpage des arêtes de A ou de B et toutes les *arêtes d'intersection* entre les faces de A et de B ; 2) tous les sommets d'arêtes, qu'ils soient des extrémités d'arêtes originelles de A et/ou de B, ou de nouveaux points d'intersection entre les frontières de A et de B ; et 3) tous les *pans* incidents à toutes les arêtes du *sur-ensemble*, quelque soit leur origine.

Afin d'unifier la représentation de toutes les arêtes (sous-arêtes ou arêtes d'intersection) et de tous les sommets (extrémités d'arêtes initiales ou nouveaux points d'intersection), nous utilisons une structure de données auxiliaire dans laquelle les faces ne subsistent qu'à travers les quadruplets de coefficients qui définissent leurs plans de support. Les sommets sont maintenant uniques et affectés chacun d'une listes d'arêtes incidentes. Les arêtes et les pans restent uniques dans la structure de données intermédiaire.

IV.3.1. Structure de données auxiliaire

Cette structure de données comporte les enregistrements suivants :

Sommet

- un triplet (x, y, z) de coordonnées rationnelles ;
- une liste de pointeurs sur toutes les *arêtes* incidentes ;

Arête

- deux pointeurs sur les *sommets gauche* et *droit* de l'arête ;
- liste de pointeurs sur les *pans* incidents à l'arête ;

Pan

- un pointeur sur l'*arête* du pan ;
- un quadruplet (a, b, c, d) de rationnels, hérité d'une *face* de A ou de B ;
- le *sens* du pan (± 1) ;
- un pointeur sur le *solide* (A ou B) d'où provient le pan ;
- un champ *utilité* qui sera utilisé plus loin, lors la sélection des pans.

La Figure 16 donne le *graphe d'adjacence* correspondant à la structure de données auxiliaire.

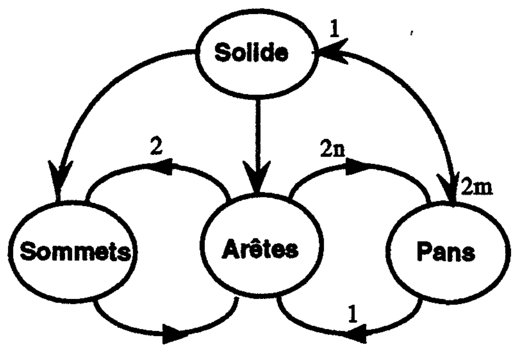


Figure 16. Le sur-ensemble de ∂(A∩B) est décrit par une liste de pans (en nombre pair), une liste d'arêtes et une liste de sommets. Un pan est défini par une arête, un plan orienté et une référence au solide (A ou B) d'origine. Une arête est définie par deux sommets et par la liste des pans incidents (en nombre pair). Un sommet est défini par ses coordonnées et par la liste de toutes les arêtes incidentes. La notion de face disparaît.

IV.3.2. Génération de la structure de données auxiliaire

En partant d'une structure de données vide, les éléments (sommets, arêtes et pans) du *sur-ensemble* sont créés et stockés, au fur et à mesure du découpage des arêtes de A et B. Un pan est créé comme un enregistrement du type (*arête, plan, sens, solide)*. La génération de la structure de données auxiliaire utilise les primitives de création suivantes :

• *CréerSommet* (p) : retourne un pointeur sur un *sommet* créé à partir d'un triplet p de coordonnées, donné en paramètre. L'unicité des sommets est assurée grâce à une table de sommets, à *adressage dispersé* ("Hash-table"), dans laquelle les triplets de coordonnées servent de *clés* de recherche.

• *CréerArête* (s_1, s_2) : retourne un pointeur sur une *arête* construite sur deux *sommets* s_1 et s_2, donnés en paramètre. Une référence à cette arête est ajoutée à la liste d'incidence de chacun des sommets. Deux arêtes *confondues*, issues de solides distincts, sont fusionnées en une seule arête qui hérite de tous les pans de A et B, incidents aux deux arêtes d'origine. La confusion d'arêtes est détectée grâce à une "hash-table" d'arêtes, dans laquelle le couple (s_1, s_2) sert de *clé* de recherche.

• *CréerPan* (a, Q, α, S) : crée un *pan* d'arête a, de plan Q, de *sens* α et de solide S (A ou B). Deux tuples tels que (a, Q, α, A) et (a, Q, α, B), ne différant que par le solide d'origine, sont fusionnés en un seul pan (a, Q, α, AB), dit *pan double*, dans lequel AB est un pointeur particulier signifiant que ce pan est *commun* aux deux solides (en pratique, $AB = nil$: simple "truc" de programmation qui évite de prévoir deux champs distincts dans l'enregistrement "pan"). Les pans *doubles* sont détectés grâce à une "hash-table" de plans, dans laquelle les quadruplets des plans servent de *clés* de recherche.

La structure de données est mise à jour en deux phases: génération des *arêtes d'intersection* et de leurs pans incidents et génération des arêtes issues du découpage des arêtes de A et B, successivement :

IV.3.2.1. Génération des arêtes d'intersection

Chaque *arête d'intersection* (p, q, f, g) est créée avec exactement quatre pans, deux pans de A, de *sens* opposés et héritant du quadruplet définissant le plan de f ; et deux pans de B, de *sens* opposés et héritant du quadruplet définissant le plan de g.

Le pseudo-code ci-dessous donne les détails de cette phase.

```
pour chaque arête d'intersection (p, q, f, g) faire
        # soient Qf et Qg les plans de f et g, respectivement ;
        s1 := CréerSommet (p) ;
        s2 := CréerSommet (q) ;
        h := CréerArête (s1, s2) ;
        CréerPan (h, Qf, -1, A) ;
        CréerPan (h, Qf, +1, A) ;
        CréerPan (h, Qg, -1, B) ;
        CréerPan (h, Qg, +1, B) ;
fait
```

IV.3.2.2. Génération des arêtes homogènes

Le découpage d'une arête originelle a d'un solide (disons A) nécessite un tri lexicographique de tous les enregistrements "sous-arêtes" mémorisés pour a. Ensuite, chaque sous-arête h est créée avec autant de pans de A que de pans originaux, incidents à a.

Si une sous-arête h est incluse dans une face g de l'autre solide (B), alors h reçoit deux pans supplémentaires, de sens opposés et héritant tous deux du quadruplet définissant le plan de g (Fig. 17).

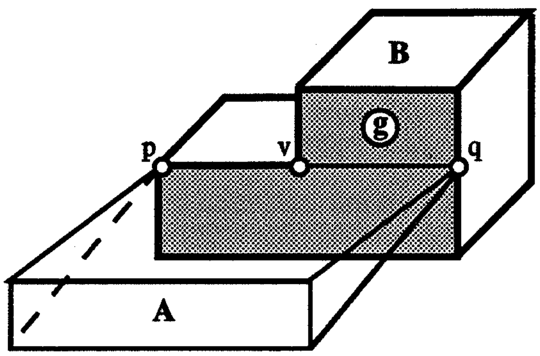


Figure 17. L'arête [p, q] du solide A (en trait fin) est découpée en deux sous-arêtes homogènes [p, v] et [v, q]. La sous-arête [p, v] est commune à A et B (en gras) ; elle est créée une seule fois avec deux pans de A et deux pans de B. La sous-arête [v, q] de A est incluse dans une face g (grisée) de B ; elle est créée avec deux pans de A et deux pans de B appartenant à g.

La procédure ci-dessous s'applique, de façon symétrique, à chacun des solides A et B, successivement :

EclaterSolide (*A* : Polyèdre)
début
  pour chaque arête *initiale* *a* de *A* faire
    trier lexicographiquement la liste des n sous-arêtes (p_i, g_i) de *a* ;
    s_1 := *CréerSommet* (p_1) ;
    pour i := 1 à n-1 faire
      $s2$:= *CréerSommet* (p_{i+1}) ;
      h := *CréerArête* (s_1, s_2) ;
      pour chaque pan *initial* de *A*, incident à *a* faire
        # le pan est de *sens* α et appartient à une face *f* de *A*, de plan *Q*
        CréerPan (*h*, *Q*, α, *A*) ;
      fait ;
      si ($g_i \neq nil$) alors
        # la sous-arête *h* est incluse dans une face g_i de *B*, de plan *Q*
        CréerPan (*h*, *Q*, -1, *B*) ;
        CréerPan (*h*, *Q*, +1, *B*) ;
      fsi ;
      $s_1 := s_2$;
    fait
  fait
fin

Notons qu'une sous-arête née d'un chevauchement de deux arêtes initiales *a* de *A* et *b* de *B* (par exemple [p, v] en Fig. 17) hérite de tous les pans de *A* incidents à *a* et de tous les pas de *B* incidents à *b*. Il n'y a donc pas besoin de mémoriser le fait qu'une sous-arête d'un solide se trouve incluse dans une arête de l'autre solide [Mich-87].

IV.4. Calcul de l'utilité des pans

Dans cette quatrième étape, chaque pan du *sur-ensemble* généré précédemment est examiné pour déterminer s'il doit faire partie ou non de la frontière de $A \cap B$. Le champ *utilité* de chaque pan est initialisé à la valeur *inconnu* ; sa valeur finale (*utile* ou *inutile*) sera connue à l'issue de cette étape.

IV.4.1. Définitions

Incidence sommet/pan

Afin d'alléger l'exposé, un pan incident à une arête *a*, elle-même incidente à un sommet *s*, sera dit incident à *s*. Bien que non explicitée dans la structure de données, la relation d'incidence sommet/pan est facile à établir à partir des relations sommet/arête et arête/pan.

Arêtes banales

Une arête sera dite *banale* si, et seulement si, tous les pans incidents appartiennent à un même solide, et rien qu'à lui. Autrement, l'arête est dite non banale : elle possède au moins un pan issu du solide *A* et au moins un pan issu du solide *B*. Sont non banales toutes les arêtes d'intersection ainsi que toutes les arêtes "non eulériennes" telles que [p, v] et [v, q] de la figure 17). Aucun calcul géométrique n'est nécessaire pour reconnaître une arête banale.

Pans voisins

Soient *P* un pan incident à une arête *a* et *s* un sommet de cette arête. Le *voisin* de *P* au sommet *s* est l'unique pan *V*, incident à s, de même solide que *P* (si *P* est un *pan double* alors *V* est également un *pan double*), de même plan orienté que *P* et tel qu'aucun autre pan vérifiant ces conditions ne peut se trouver "entre" *P* et *V*. Il est évident que *P* est aussi le voisin de *V* au même sommet *s*: les deux pans seront dits *voisins* au sommet *s*. Enfin, chaque pan *P* incident à *a* possède exactement deux voisins V_1 et V_2: l'un au sommet *gauche* et l'autre au sommet *droit* de *a* (Fig. 18).

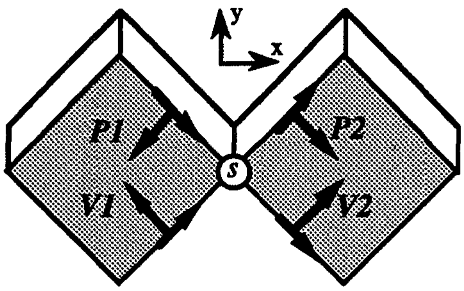


Figure 18. Les pans P_1 etV_1 sont voisins au sommet s.
Idem pour les pansP_2 etV_2.

Les structures de données auxiliaire permet d'accéder aisément à tous les pans incidents au sommet s. Parmi ces pans, on ne retient que ceux qui sont de même solide et de même plan orienté que P. Le plus souvent, seulement deux pans répondent à la question, l'un étant P et l'autre le voisin de P en s. S'il y a plusieurs pans candidats, un tri géométrique est nécessaire pour identifier le voisin V. Nous donnons ci-dessous une méthode simple permettant de trier radialement un ensemble de vecteurs coplanaires, à coordonnées rationnelles.

Angles rationnels

Etant donnée une liste de vecteurs $V(x, y)$ à composantes rationnelles, il est toujours possible de l'ordonner radialement autour de l'origine, par des calculs *rationnels*. Pour cela, on substitue à *l'angle polaire* de chaque vecteur $V(x, y)$ le nombre *rationnel* $r(x, y) \in [0, 8[$ défini par :

$$
\begin{aligned}
r(x, y) &= \frac{y}{x} && \underline{\text{si}}\ 0 \le y \le x\,; \\
&= 2 - \frac{x}{y} && \underline{\text{si}}\ 0 \le y\ \underline{\text{et}}\ |x| \le y\,; \\
&= 4 + \frac{y}{x} && \underline{\text{si}}\ x \le 0\ \underline{\text{et}}\ |y| \le |x|\,; \\
&= 6 - \frac{x}{y} && \underline{\text{si}}\ y \le 0\ \underline{\text{et}}\ |x| \le |y|\,; \\
&= 8 + \frac{y}{x} && \underline{\text{si}}\ y < 0 \le x\ \underline{\text{et}}\ |y| \le x.
\end{aligned}
$$

Cette valeur correspond à la longueur (sur le carré C de coté 2, centré à l'origine) du chemin allant du point (0, 1) au point d'intersection entre $V(x, y)$ et C (Fig. 19).

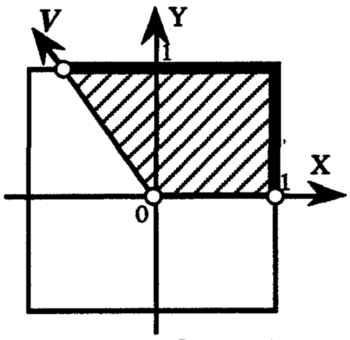


Figure 19. L'angle rationnel associé au vecteur V est donné par la longueur du chemin en gras.

IV.4.2. Méthodes de calcul de l'utilité d'un pan

Le calcul de l'utilité d'un pan incident à une arête a repose sur l'examen des *voisinages* de a par rapport aux solides A et B. Selon le cas, l'utilité s'obtient soit par des calculs locaux et simples, soit par des calculs géométriques plus coûteux (typiquement un tri radial). Il existe quatre méthodes de calcul d'utilité, que nous présentons ci-dessous, par coût croissant :

M1. Cas d'un pan double

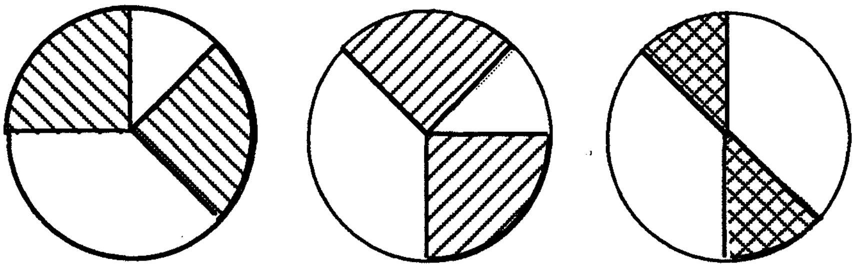
Un *pan double* est toujours *utile* ; aucun calcul géométrique n'est nécessaire. Ces pans ont été détectés lors de la génération du *sur-ensemble* de $\partial(A \cap B)$ (voir IV.3.2.2).

M2. Cas d'un pan incident à une arête banale : *cas* 1

Si l'un des sommets d'une arête *banale* a du solide (disons A) se trouve à l'extérieur de la *boîte englobante* de l'autre solide B, alors l'utilité d'un pan incident à a est évidente: le pan est *utile* si, et seulement si, B est un solide *non borné* (c'est le *complémentaire* d'un solide *borné*).

M3. Cas d'un pan incident à une arête non banale

Si a est une arête *non banale* alors elle possède, par définition, au moins un pan issu de A et au moins un pan issu de B. Considérons les *coupes* de A et B par un plan Q perpendiculaire à a et passant par le milieu de a. Dans ces coupes, les pans incidents à a sont représentés par leurs *vecteurs indicateurs* (voir III.1) et chacun des solides est représenté par une liste des *secteurs angulaires*, délimités par ces vecteurs indicateurs (Fig. 20).



A B $A \cap B$

Figure 20. Coupes des deux solides A et B par un plan orthogonal à une arête non banale a. Un pan incident à a n'est utile que si, et seulement si, son vecteur indicateur délimite un secteur angulaire intérieur à $A \cap B$.

Un pan incident à a est *utile* si, et seulement si, son vecteur indicateur délimite un secteur intérieur à $A \cap B$. On est donc ramené à un *tri radial* d'une liste de vecteurs coplanaires (au moyen d'angles *rationnels)* et à une intersection booléenne de deux listes de secteurs angulaires. Les *normales* aux plans des pans permettent d'identifier l'intérieur de chaque solide. Notons que cette procédure est semblable à l'intersection de deux listes de segments, telle que décrite en IV.2.2.2.

M4. Cas d'un pan incident à une arête banale : cas 2

Il reste enfin à déterminer l'utilité d'un pan incident à une arête *banale* a d'un solide (disons) A, dont les deux sommets se trouvent à l'intérieur de la *boîte englobante* de l'autre solide B. Par définition, tous les pans de a appartiennent à A, et rien qu'à lui. Dans ce cas, le calcul de l'utilité est fort coûteux en général, car il ne peut se faire par un test local. La méthode consiste à considérer un point quelconque I intérieur à a (par exemple, le milieu de a) et à le *classifier* par rapport à l'autre solide B. Un pan de a est alors *utile* si, et seulement si, le point I est *intérieur* à B.

La classification d'un point par rapport à un solide polyédrique est un problème classique, mais délicat. La solution la plus simple exploite le *principe de parité* [LTH-86]: on considère un rayon R issu de I et de direction arbitraire, et on compte le nombre d'intersections entre R et toutes les faces de B. Il est nécessaire de savoir reconnaître si un point d'intersection entre R et le plan d'une face de B se trouve ou non à l'intérieur de cette face. Ce problème peut se résoudre là aussi, en lançant un autre rayon dans le plan de la face et en comptant le nombre d'intersections entre ce rayon et les arêtes de la face.

IV.4.3. Propagation de l'utilité d'un pan

Les quatre méthodes ci-dessus doivent être appliquées dans l'ordre M1 à M4, à tous les pans auxquels elles peuvent s'appliquer. En particulier, la quatrième méthode ne doit être invoquée que si, et seulement si, les trois premières méthodes ne permettent pas de conclure. D'autre part, il existe des règles simples permettant de propager l'utilité d'un pan à nombreux autres pans. La technique de propagation permet d'éviter un volume appréciable de calculs géométriques (tri radial) ; elle est aussi utilisée, entre autres, dans [LAID 86] et [SS-88]. Plusieurs règles de propagation sont possibles, mais les deux suivantes sont les plus simples à mettre en œuvre :

> **R1**) tous les pans incidents à une même *arête banale* ont même utilité ;
> **R2**) deux pans *voisins* en un même sommet ont même utilité.

La procédure de propagation utilise une *pile* pour mémoriser les pans en instance de propagation; un pan n'est empilé qu'une seule fois (procédure *EmpilerSi*), il est dépilé dès que son utilité a été propagée.

```
EmpilerSi (P: Pan, u: Utilité)
début
        si (P->utilité = inconnu) alors
                P->utilité := u ;
                EmpilerPan (P) ;
        fsi
fin
```

L'utilité d'un pan se propage dans le graphe *virtuel* dont les nœuds sont les pans et dont les arcs sont définis par les deux relations binaires "incident à une même arête banale" et "voisin au même sommet". La procédure *Propager* ci-dessous donne le schéma général de la propagation :

```
Propager (lePan: Pan ; v : Utilité)
début
        EmpilerSi (lePan, v) ;
        tant que (la pile n'est pas vide) faire
                P := Dépiler () ;
                u := P->utilité ; # l'utilité de P est connue
                a := P->arête ; # l'arête attachée au pan P ; a = [s1 , s2]
                si a est une arête banale alors
                        pour chaque pan Q incident à a faire
                                EmpilerSi (Q, u )
                        fait
                fsi ;
                EmpilerSi (PanVoisin (P, s1), u) ;
                EmpilerSi (PanVoisin (P, s2),u) ;
fin
```

Technique de l'attribut/fonction

Pour éviter le calcul redondant du voisin d'un même pan en un même sommet, on fait appel à la technique *attribut/fonction*. Cela consiste à rajouter à chaque pan P deux champs *VoisinGauche* et *VoisinDroit* (initialisés à *nil*), destinés à garder la trace des voisins de P aux sommets *gauche* et *droit* de l'arête de P, respectivement. Chaque fois qu'est demandé le voisin V d'un pan P, la fonction *PanVoisin* commence par vérifier si V est déjà disponible ; si c'est le cas, elle rend simplement un pointeur sur ce pan V, sinon le calcul de V est effectué et le pointeur sur V est mémorisé dans le champ correspondant de P. D'autre part, lors du calcul du voisin d'un pan donné, on profite du tri radial effectué pour déduire les voisins d'autres pans intervenant dans le calcul.

IV.5. Construction du solide résultat

A ce stade de l'algorithme, chaque pan du *sur-ensemble* est marqué soit *utile*, soit *inutile* à $A \cap B$. La description finale de la frontière du solide $S = A \cap B$ peut être maintenant générée à partir de tous les pans trouvés *utiles*. Il reste simplement à reconstituer les *faces maximales* de S, de façon à obtenir une structure de données analogue à celle des solides opérandes A et B (voir III.3). Pour cela, il suffit de regrouper tous les pans coplanaires ayant le même quadruplet (a, b, c, d): une "hash-table" de plans est utilisée à cet effet.

Enfin, un post-traitement simple est effectué sur la structure de données résultante afin d'en éliminer les éventuelles arêtes "inutiles" (voir III.4), produites par l'algorithme. Une telle arête a exactement deux pans incidents appartenant à une même face ; l'arête est supprimée avec ses deux pans, par une simple mise à jour de pointeurs.

Contrairement à l'algorithme de Michelucci, notre version ne crée jamais de sommets "inutiles" (ayant exactement arêtes incidentes alignées). En effet, une arête d'un solide n'est découpée que si, et seulement si, elle intersecte strictement l'intérieur ou le contour d'une face de l'autre solide.

A l'issue de ce post-traitement, la frontière du solide $A \cap B$ est entièrement déterminée et peut à son tour constituer une entrée valide pour l'algorithme d'intersection (autrement dit, l'algorithme est *réentrant*).

IV.6. Traitement des sommets isolés

Par souci de clarté dans la présentation de l'algorithme d'intersection, nous avons volontairement omis le cas particulier, unique, d'un solide A possédant un sommet v isolé à l'intérieur d'une face f (Fig. 21). La prise en compte d'un tel sommet est impérative, faute de quoi l'algorithme risquerait de produire un solide incohérent lors de l'intersection de A et d'un autre solide B. En effet, lorsqu'une *arête d'intersection* (p, q, f, g) est trouvée entre f et une face g de B et que, par malchance, $v \in \,]p, q[$, l'algorithme produirait pour $S = A \cap B$ une arête $[p, q]$ qui est en violation du premier critère de validité énoncé en Section III.4: cette arête intersecterait d'autres arêtes de S, ailleurs qu'en une extrémité commune.

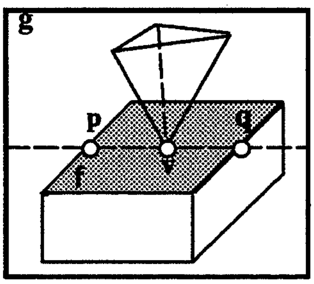


Figure 21. Sommet isolé. Le sommet v du solide A (l'union du cube et du tétraèdre) est isolé dans la face (grisée) f de A. Lors de la détection d'une *arête d'intersection* [p, q], entre f et une face g de B (non représenté), l'arête [p, q] doit être coupée en v, puis enregistrée sous forme de deux arêtes [p, v] et [v, p].

Représentation des sommets isolés

Il suffit d'associer *explicitement* à chaque face f d'un solide A une liste (vide, le plus souvent) de tous les sommets isolés dans f, chacun étant représenté par un simple triplet de coordonnées. La détection des *sommets isolés* de f se fait sans difficultés, au fur et à mesure de son intersection avec les diverses faces g de l'autre solide B.

Traitement des sommets isolés

Lors du l'intersection de A et B, chaque fois qu'est détectée une *arête d'intersection* $[p, q]$ entre deux faces transversales f de A et g de B, l'algorithme vérifie si par hasard l'une et/ou l'autre des deux faces possède un *sommet isolé* v, situé quelque part entre p et q (i.e. $v \in \,]p, q[$). Auquel cas, cette *arête d'intersection* est tout simplement stockée sous forme de deux enregistrements (p, v, f, g) et (v, p, f, g), puis v est supprimé de la liste des sommets isolés.

Par conséquent, la prise en compte du cas particulier des *sommets isolés* n'implique que des ajouts mineurs, aussi bien aux structures de données qu'à l'algorithme d'intersection.

V. RESOLUTION D'ARBRES CSG

L'algorithme d'intersection décrit ci-dessous est *réentrant*. Il peut donc être utilisé pour concevoir un algorithme d'*évaluation de frontière* [RV-85], qui convertit une représentation CSG en une représentation par frontière équivalente.

V.1. Complémentaire d'un solide

Le calcul de la frontière du complémentaire d'un solide se fait en temps linéaire, en fonction du nombre de pans du solide. Il consiste simplement à inverser l'orientation de la frontière du solide opérande :

```
Complément (A : Polyèdre)
début
        inverser le valeur du champ borne? de A ;
        pour chaque face f de A faire
                changer le signe de chaque coefficient définissant le plan de f ;
                pour chaque pan P de f faire
                        inverser le sens de P
                fait
        fait
fin
```

V.2. Union et différence de deux solides

Les opérations booléennes d'union ou de différence régularisées se ramènent sans problème aux opérations d'intersection et de complémentation, par application des lois de De Morgan. Bien entendu, l'union de deux solides dont les boîtes englobantes sont strictement disjointes revient simplement à une *concaténation* de frontières ; de même, l'intersection (régularisée) de deux solides dont les boîtes englobantes sont disjointes est nécessairement *vide*.

Par conséquent, il devient possible de calculer, récursivement, la frontière de tout objet solide défini par un arbre CSG dont les *primitives* sont des *solides polyédriques* rationnels, de frontière connue. Il suffit de savoir convertir en polyèdres *rationnels*, des primitives (habituellement) décrites comme des polyèdres *flottants*. Cette conversion sera abordée au chapitre III. Par ailleurs, une optimisation des algorithmes d'évaluation d'arbres CSG sera présentée au Chapitre IV.

VI. ANALYSE DE COMPLEXITE

Dans notre algorithme, l'essentiel des calculs s'effectue au cours de l'étape d'intersection des faces. Etant donnés deux solides A et B de F_A et F_B faces, respectivement, le prétraitement pour la recherche des faces candidates a un coût *théorique* $K = O(F_A * F_B)$. Soient f et g deux faces transversales de P_f et P_g pans, respectivement. Les $O(P_f)$ *contacts* entre f et le plan de g sont calculés en O(1) chacun, puis ordonnés lexicographiquement en $O(P_f * \log(P_f))$. Après un traitement symétrique pour la face g, de coût $O(P_g * \log(P_g))$, les deux listes ordonnées de contacts sont fusionnées en $O(P_f + P_g)$. Par conséquent, l'intersection des $O(F_A * F_B)$ paires de faces candidates a un coût global de :

$$T = O\Big(\sum_{(f,\, g)} P_f * \log(P_f) + P_g * \log(P_g)\Big).$$

Difficulté de l'analyse

La formule ci-dessus est difficile à préciser davantage: le nombre de pans par face est difficile (voire impossible) à expliciter en fonction des autres paramètres de la représentation par frontière. Cette difficulté est due essentiellement au fait que la classe des solides acceptés par l'algorithme s'étend à tous les polyèdres réguliers, y compris les polyèdres non eulériens. Or, dans cette classe des polyèdres, il n'existe pas en général de relations combinatoires simples entre les cardinalités des divers éléments de la frontière: en particulier, la formule généralisée d'Euler ne s'applique pas à cette classe. Tout au plus les auteurs se contentent-ils d'une analyse grossière qui permet de donner une idée de la complexité de leur algorithme [LTH-86], [PRS-89].

Toutefois, afin de fixer les idées, nous contournerons la difficulté en considérant les sous-classes de polyèdres dans lesquels le nombre de pans par face est le même quelle que soit la face. Les quatre exemples ci-dessous, concernent des polyèdres de différentes complexités :

Exemple 1 :

Soit A le polyèdre obtenu par une triangulation classique de la frontière d'une sphère. Ce polyèdre étant eulérien, les nombres de sommets V_A, d'arêtes E_A et de faces F_A vérifient la relation simple d'Euler: $V_A - E_A + F_A = 2$. Par conséquent, le nombre d'arêtes est de $E_A = O(F_A)$, le nombre total de pans est de $P_A = O(F_A)$ et chaque face f comporte exactement $P_f = 3$ pans.

Exemple 2 :

La Figure 22 montre un polyèdre *eulérien* A, défini comme l'union de n^3 ($n = 4$) cubes disjoints. Le nombre de faces maximales est $F_A = 6n$, n faces dans chaque direction principale (soit n^2 faces au sens habituel) ; le nombre d'arêtes est de $E_A = 12n^3$ (12 arêtes par cube) ; le nombre total de pans est de $P_A = 24n^3$ et chaque face maximale f comporte exactement $P_f = 4n^2$ pans. Par conséquent, $F_A = O(n)$, $E_A = O(F_A{}^3)$, $P_A = O(F_A{}^3)$ et $P_f = O(F_A{}^2)$.

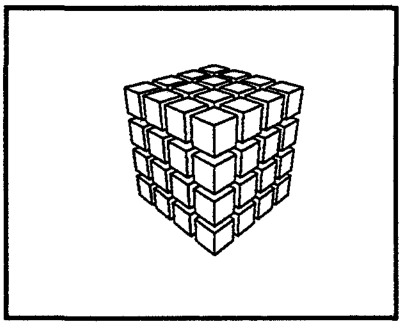


Figure 22.

Exemple 3 :

La Figure 23 montre une "éponge" de Sierpinski au niveau de récursion $n = 2$. Malgré sa complexité apparente, c'est un polyèdre *eulérien*. Laborieusement, on dénombre $O(3^n)$ faces, $O(n^2)$ arêtes, $O(n^2)$ pans et $O(n)$ pans par face. Par conséquent, $F_A = O(3^n)$, $E_A = O(F_A{}^2)$, $P_A = O(F_A{}^2)$ et $P_f = O(F_A)$.

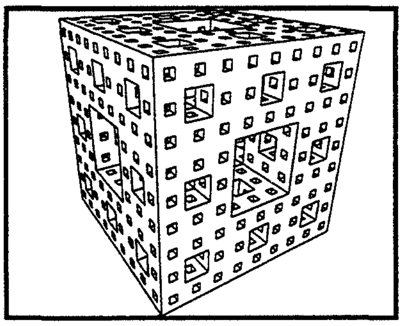


Figure 23

Exemple 4 :

La Figure 24 montre un polyèdre *non eulérien* A, défini comme l'union de $n^3/2$ (sur l'exemple, $n = 5$) cubes adjacents par les arêtes (exemple extrait de [Kara-89]). Cet objet a bien plus d'arêtes que de faces. Pour tout n donné, on dénombre $O(n)$ faces, $O(n^3)$ arêtes, $O(n^3)$ pans et $O(n^2)$ pans par face. Par conséquent, on a encore : $F_A = O(n)$, $E_A = O(F_A{}^3)$, $P_A = O(F_A{}^3)$ et $P_f = O(F_A{}^2)$.

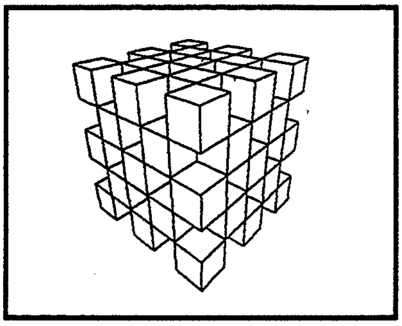


Figure 24

Dans ces quatre classes de polyèdres le nombre P_f de pans par face est soit constant, soit linéaire ou quadratique en fonction du nombre de faces. Le coût de l'intersection de deux polyèdres est au pire:

$$T = O\{K*(F_A{}^2*\log(F_A) + F_B{}^2*\log(F_B)\},$$

où $K = O(F_A*F_B)$. Et, si l'on pose $F_A = F_B = n$ (pour fixer les idées), on obtient :

$$T = O(n^4*\log(n))$$

Remarque

En considérant de nombreux autres exemples de polyèdres "fractals", il apparaît que le nombre de pans par face ne peut être plus que *quadratique* en fonction du nombre de faces maximales (ou plans orientés). Peut-être y a t-il là une relation combinatoire générale. Nous le pensons effectivement, mais nous n'avons pas su le prouver formellement.

Discussion

La complexité ci-dessus est difficile à comparer avec celles des algorithmes proposés dans la littérature. Le fait est que les auteurs n'utilisent pas les mêmes paramètres dans leurs analyses et l'expression d'un paramètre donné en fonction des autres paramètres ne va pas de soi comme nous l'avons signalé précédemment. Par ailleurs, les définitions mêmes de ces paramètres varient d'un auteur à l'autre: par exemple, les faces sont triangulaires dans [YT-84] ou [PRS-89], convexes dans [LTH-86], et connexes dans [SS-88] ou [Kara-89]. Dans notre cas, une face peut être arbitrairement complexe: elle est décrite par une simple liste de pans situés sur un même plan orienté.

A notre connaissance Karasick [Kara-89] est le premier à tenter d'approfondir son analyse de complexité dans le domaine des polyèdres non eulériens. Il faut dire que la représentation par frontière utilisée ("Star-Edge") se prête bien aux dénombrements: nous pensons que c'est la représentation la plus riche et la plus explicite qui soit proposée en modélisation solide. Par ailleurs, Karasick remet en cause les complexités quadratiques (en fonction de la taille n des entrées) exhibées dans [LTH-86] et [PRS-89], arguant qu'une analyse plus fine révélerait des bornes en $O(n^4)$.

VII. CONCLUSION

La méthode de résolution d'opérations booléennes présentée dans ce chapitre est fondée sur trois points essentiels :

1) elle utilise pour les solides une représentation par frontière très générale qui permet d'unifier de nombreux cas particuliers. Le domaine de représentation offert est au moins aussi étendu que celui de la représentation constructive classique, où les primitives sont des solides polyédriques quelconques de frontières connues. Ceci est normalement obligatoire pour tout algorithme de conversion CSG/*frontière* ;

2) elle traite uniformément tous les cas particuliers (sauf un). Pour un problème aussi difficile que la résolution d'opérations booléennes, une telle méthode conduit à des algorithmes "lisibles" moins difficiles à mettre en œuvre ou à modifier ;

3) on n'a pas à se préoccuper du problème de l'imprécision numérique inhérente à l'arithmétique *flottante* standard. Plutôt que de recourir à des solutions partielles dépendant de chaque type d'algorithmes, nous avons opté pour l'emploi d'une arithmétique *exacte*, de façon à ce qu'aucune *incohérence topologique* ne puisse avoir lieu du seul fait d'une *incohérence numérique*. L'algorithme d'intersection arrive toujours à son terme et produit toujours un résultat cohérent à partir de deux objets cohérents.

Il reste maintenant à s'occuper des implications en temps et en espace de l'utilisation d'une arithmétique exacte. Ce problème sera largement abordé au chapitre III, où il sera question d'une arithmétique rationnelle optimisée qui n'effectue que les calculs exacts strictement nécessaires.

Chapitre III

Implantation de l'algorithme. Difficultés dues à l'imprécision. Résultats et Tests.

I. INTRODUCTION

Dans ce chapitre, nous décrivons une implantation d'un module d'évaluation d'arbres CSG basé sur l'algorithme d'intersection décrit au chapitre II. L'utilisation d'une arithmétique rationnelle fait surgir de nombreux problèmes pour lequels nous proposons diverses solutions.

En Section II, nous revenons sur les solides rationnels et présentons une solution permettant leur construction à partir des solides flottants habituels. De même, nous abordons le problème de l'approximation d'un solide rationnel par un solide flottant cohérent.

En Section III, nous présentons la première version de l'algorithme, basée sur l'utilisation d'une arithmétique rationnelle classique. Les implications en temps et en mémoire sont importantes, mais prévisibles. Nous décrivons une première optimisation qui consiste à faire quelques "économies" de calculs rationnels en effectuant certains calculs en arithmétique flottante.

En Section IV, nous présentons le principe d'un nouveau type d'arithmétique rationnelle qui a conduit à une librairie arithmétique très performante, dite "paresseuse" [BJMM-93a]. L'utilisation de cette librairie a sensiblement amélioré les performances de l'algorithme.

En Section V , nous montrons que les performances des algorithmes géométriques qui utilisent une arithmétique paresseuse varient d'une implantation à l'autre [BJMM-93b]. En particulier, nous décrivons quelques techniques permettant de tirer le meilleur parti de cette arithmétique.

En Section **VI**, nous présentons une methode de calcul de clés nécessaires à l'emploi des *tables à adressage dispersé* (ou "Hash-Tables") [BJMM-93c]. L'arithmétique paresseuse ne permet pas a priori l'utilisation de telles tables car les valeurs exactes des nombres ne sont pas toujours connues, par définition du concept d'arithmétique paresseuse.

En Section **VII**, nous présentons quelques résultats de tests permettant d'estimer les performances comparées des diverses versions de l'algorithme.

Nous concluons en **VIII**.

II. SOLIDES RATIONNELS ET PROBLEMES LIES

II.1. Définition

Un solide polyédrique *rationnel* est un solide représenté par sa frontière, dans laquelle les *coordonnées* des sommets et les *coefficients* des équations des plans (des faces) sont représentés par des nombres *rationnels*. Un solide rationnel est numériquement *cohérent*, en ce sens que tout sommet a des coordonnées (x, y, z) qui vérifient exactement l'équation $(ax + by + cz + d = 0)$ du plan de chaque face incidente.

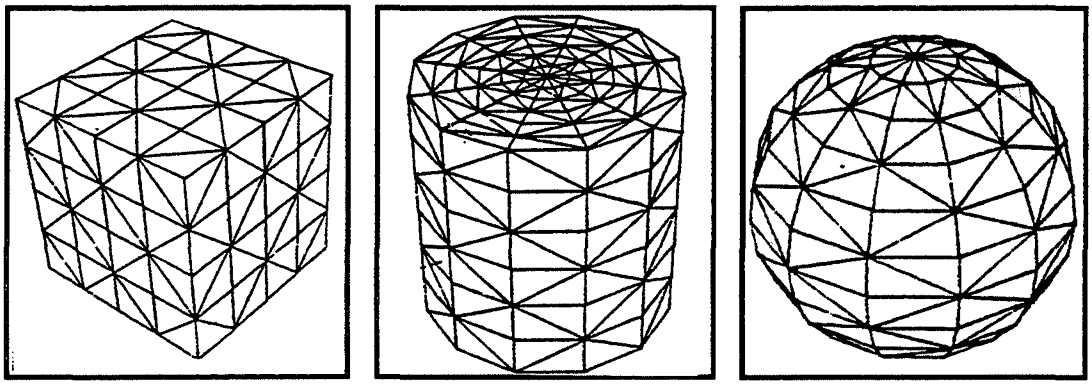
Les algorithmes de conversion *CSG/Frontière* (tels qu'ils sont utilisés dans la plupart des modeleurs polyédriques) prennent en entrée des arbres CSG dont les primitives sont habituellement des polyèdres dont la frontière est représentée par des données géométriques flottantes. Or l'algorithme opère toujours sur des polyèdres rationnels et produit d'autres polyèdres rationnels. De ce point de vue, nous nous sommes heurtés à deux problèmes pratiques:

1) comment convertir un polyèdre flottant cohérent en un polyèdre rationnel ?
2) Comment approcher un polyèdre rationnel par un polyèdre flottant qui soit cohérent.

II.2. Conversion d'un solide flottant cohérent en solide rationnel

La difficulté tient au fait qu'une face de primitive comporte en général plus de trois sommets: on ne peut alors garantir l'existence d'un plan unique contenant tous les sommets de la face.

Notre solution consiste à *trianguler* toutes les faces des primitives (Fig. 1), puis à forcer (*caler* ou *recaler*[1]) chaque sommet flottant sur le sommet le plus proche à coordonnées rationnelles: cela revient à remplacer chacune des coordonnées par une approximation rationnelle raisonnable (nous présenterons en II.5 trois méthodes d'approximation possibles).



(a) (b) (c)

Figure 1. Triangulation des primitives usuelles:
a) le cube, **b**) le cylindre et **c**) la sphère.

Chaque facette triangulaire appartient à un plan unique dont l'équation peut être calculée sur les coordonnées rationnelles des trois sommets A, B et C: le produit vectoriel rationnel $\overrightarrow{AB} \times \overrightarrow{AC}$ donne un vecteur $\vec{N} = (a, b, c)$ *normal* à la facette et dirigé vers l'extérieur de la primitive (A, B et C sont donnés dans l'ordre *trigonométrique*). Enfin, le produit scalaire rationnel $d = -(\vec{N} \bullet \overrightarrow{OA})$ donne le quatrième coefficient qui définit l'équation ($ax + by + cz + d = 0$) du plan de la facette.

Sugihara et Iri [SI-89] ont rencontré un problème similaire dans leur modeleur polyédrique. Leur solution est basée sur une méthode *duale* de la notre ; elle consiste à limiter les primitives à des polyèdres très simples dont chaque sommet a exactement *trois* faces incidentes, de façon que chaque sommet puisse être toujours défini comme l'intersection de trois plans. Ensuite, chaque plan flottant de chaque primitive est *calé* sur un plan à coefficients *entiers*, qui approche le mieux le plan original (ce problème est dit *approximation diophantine simultanée* [Sugi-89]).

Cette solution exclut donc les primitives usuelles telles que le cône, la sphère ou le tore "facétisés" (par la méthode habituelle). Nous estimons que notre solution est plus naturelle et plus facile à mettre en œuvre.

[1] Dans [Mich-87], l'opération qui consiste à approcher un sommet flottant par un sommet rationnel est désignée par le terme "calage" (ou "recalage"). Nous adopterons ce terme dans la suite de l'exposé.

II.3. Rotation d'un polyèdre rationnel

Ayant résolu le problème de la conversion d'un polyèdre flottant en un polyèdre rationnel, il nous fallait surmonter une autre difficulté: étant donné un polyèdre rationnel, comment lui appliquer une *rotation* dont la *matrice* associée contient des éléments non rationnels (par exemple le rotation d'angle $\alpha = 45$ degrés), sans altérer la *topologie* initiale du solide ?

A ce propos, Milenkovic et Nackman [MN-90] posent le problème (dit "*p-k-approximation*) suivant: étant donné un réseau de polygones (ou de polyèdres) dont les coordonnées des sommets sont connues avec p chiffres significatifs, comment lui faire subir une rotation sans détruire ni modifier sa cohérence topologique, en se contentant seulement de p chiffres significatifs (la rotation rend non significatifs les k bits de poids faible) pour la représentation des coordonnées transformées ? Ils démontrent que ce problème est NP-*Complet*.

Notre solution consiste simplement à appliquer directement aux primitives toutes les transformations affines spécifiées dans l'arbre CSG. Cette opération est entièrement effectuée en arithmétique flottante. L'arbre CSG obtenu ne contient plus que des opérations booléennes ($\cap, \cup, -$). Empiriquement, nous n'avons pas rencontré de difficulté particulière dans cette phase de l'algorithme.

II.4. Méthodes d'alignement d'un flottant sur un rationnel

Nous décrivons maintenant trois méthodes possibles pour approcher un nombre flottant par un nombre rationnel.

II.4.1. Décodage de la représentation binaire

Un nombre *flottant* peut être considéré comme un nombre *rationnel* codé dans le format binaire propre à une machine donnée. Par conséquent, il est toujours possible de le transcrire dans tout autre format choisi pour la représentation des rationnels. Cette méthode permet de convertir un flottant en rationnel sans aucune perte d'information. Cependant, elle est coûteuse et produit des rationnels inutilement encombrants: de toute façon, les nombres flottants ne sont que des *approximations* de données *réelles*.

II.4.2. Utilisation d'une grille rationnelle

Les sommets initiaux sont forcés sur les nœuds les plus proches d'une *grille cubique* régulière dont le *pas* est fonction de la précision souhaitée. Moyennant une *homothétie* adaptée, cela revient à *arrondir* chaque coordonnée initiale sur le nombre entier le plus proche. L'avantage de cette méthode est qu'elle produit toujours des nombres rationnels *bornés*.

Pour simplifier, considérons une grille de pas 1 et supposons que toutes les coordonnées appartiennent à l'intervalle $[0..G]$, pour G suffisamment grand. Alors, l'équation de chaque plan (calculée sur trois points donnés) a des coefficients a, b, c et d tels que $Max(|a|, |b|, |c|) \leq 2G^2$ et $|d|) \leq 6G^3$. De même, chaque point d'intersection entre trois plans donnés a des coordonnées rationnelles $\frac{p}{q}$ telles que $|p| \leq 48G^6$ et $|q| \leq 144G^7$.

Ces bornes sont indépendantes de la profondeur de l'arbre *CSG*. Pour une précision de 1 *mm* dans un univers de 1 Km^3, soit $G = 10^6$, les entiers les plus grands ne peuvent dépasser $144*10^{42}$ ($\leq 2^{147}$), de sorte qu'une arithmétique entière *bornée* de 256 bits serait suffisante. Cependant, afin de laisser ouverte la *dynamique* des données (contrairement à l'approche de Sugihara et Iri [SI-89]), l'arithmétique rationnelle que nous avons utilisée représente les grands entiers par des listes de *chiffres* dans une certaine *base* B (typiquement B = 32768, sur une machine 32 bits).

II.4.3. Développement en fractions continues

Il est bien connu que tout nombre *réel* x peut être représenté par un *développement en fractions continues* (*DFC*) [HW-60], de la forme

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \dots}}},$$

où les a_i sont les termes d'une suite entière, finie si x est *rationnel*. Si l'on note $[u]$ la partie entière d'un réel u, la suite ci-dessous fournit les termes successifs $(a_n)_n$:

$$\left\{ \begin{array}{c} x_0 \leftarrow x \\ a_n \leftarrow [x_n] \\ x_{n+1} \leftarrow \dfrac{1}{x_n - a_n} \end{array} \right\},$$

et les deux suites entières $(p_n)_n$ et $(q_n)_n$ définies par :

$$\left\{\begin{array}{c} p_0 \leftarrow a_0 \\ p_1 \leftarrow a_0 * a_1 + 1 \\ p_n \leftarrow a_n * p_{n-1} + p_{n-2,\ n\geq 2} \end{array}\right\} \text{ et } \left\{\begin{array}{c} q_0 \leftarrow 1 \\ q_1 \leftarrow a_1 \\ q_n \leftarrow a_n * q_{n-1} + q_{n-2,\ n\geq 2} \end{array}\right\}$$

permettent de construire une suite $(r_n)_n$ de *rationnels*, convergeant vers x et dont les termes $r_n = \frac{p_n}{q_n}$ constituent des approximations rationnelles (irréductibles), de qualité et d'encombrement croissants.

D'après la théorie des fractions continues, ces approximations sont les meilleures possibles, au sens de la "taille" du numérateur et du dénominateur.

La suite $(|x - r_n|)_n$ est décroissante et vérifie : $\frac{1}{q_n * q_{n+2}} < |x - r_n| < \frac{1}{q_n^2}$, $\forall n \geq 0$.

II.4.4. Contraintes

Quelque soit la méthode de *recalage* adoptée, elle ne doit pas détruire la *cohérence topologique* des primitives. Par exemple, elle doit toujours produire deux sommets *rationnels* distincts à partir de deux sommets *flottants* distincts. De même, elle ne doit pas introduire d'intersections parasites. Empiriquement, nous n'avons pas rencontré de difficultés pour les primitives usuelles.

Nous avons effectivement implanté la méthode *DFC* dans laquelle nous arrêtons le développement à un rang n tel que $\left|\frac{x - r_n}{x}\right| \leq \varepsilon$, où ε est un seuil de précision laissé comme paramètre. Il va de soi que plus grande est la précision demandée, plus encombrants seront les rationnels obtenus et plus lents seront les calculs rationnels qui les feront intervenir (voir Sect. VII).

II.5. Approximation d'un solide rationnel par un solide flottant cohérent

Problème

Un autre problème, inverse du problème II.2, consiste à approcher un solide *rationnel* par un solide *flottant* qui soit *cohérent* et de même *topologie*. Ce problème est délicat: deux sommets *rationnels* très voisins peuvent devenir numériquement indiscernables en représentation *flottante*. En effet, la précision *flottante* étant toujours limitée, il est impossible de récupérer toute la précision avec laquelle sont connues les données géométriques des solides *rationnels*.

Formulation générale

En fait, ce problème est très général ; il se pose également en *arithmétique flottante* pour tous les algorithmes manipulant des représentations par frontière de solides. Etant donnée une représentation *S cohérente*, sous quelles conditions peut-on garantir qu'une transformation géométrique appliquée à *S* préserve la *cohérence* de *S* ? Ce problème difficile a été abordé par Segal et Sequin [SS-85] dans le cas simple des transformations *affines* (utilisées par exemple lors d'un changement de système de coordonnées). Dans ce cas, les auteurs montrent que la condition de cohérence s'exprime en fonction de la précision de la machine et du *conditionnement* de la matrice de transformation *M*. Notons qu'en arithmétique flottante, on a $M \times M^{-1} \neq I_d$, en général.

Une approche de solution

Dans le cas général, Segal et Sequin [SS-85] préconisent un processus de réaménagement (ou *consolidation*) dont le but est d'éliminer les "petits" détails de la représentation par frontière. Plus précisément, toute paire d'éléments (sommets, arêtes ou faces) qui sont à moins d'une distance minimale μ (fixée à l'avance) l'un de l'autre doivent être soit "fusionnés", soit éloignés l'un de l'autre de façon à maintenir cette distance μ.

Cependant, ces opérations de réaménagement ne doivent pas être effectuées itérativement, élément par élément, car les dernières opérations pourraient défaire le travail des premières. La détermination de la position finale de chaque élément doit être effectuée *globalement,* une et une seule fois.

Une méthode envisageable consiste à *caler* chaque sommet sur le nœud le plus proche d'une grille régulière de pas μ, les sommets entraînant avec eux les arêtes et les faces incidentes.

La méthode ci-dessus a un grand défaut: elle incorpore de larges portions de la grille dans la représentation, sous forme de petits éléments "en escalier". D'autres "passes" de consolidation sont nécessaires pour corriger ces éléments: Segal et Sequin [SS-85] ont laissé ce problème ouvert .

Notre position face à ce problème

En ce qui nous concerne, nous pensons que le problème de la conversion en flottant des solides rationnels produits par notre modeleur relève de l'*interface* avec les divers autres modeleurs solides. Dans la mesure ou nous fournissons une description rationnelle des solides, il serait dommage que les modeleurs ne la prennent pas comme entrée, pour ensuite l'utiliser à diverses fins: en CFAO, par exemple, les équations rationnelles des faces de pièces mécaniques sont très utiles pour l'usinage des surfaces ou l'affectation de tolérances de fabrication. Chaque modeleur peut alors transformer la description rationnelle de façon à l'adapter à ses propres besoins.

Dans la version courante de notre modeleur, nous produisons deux formats de sortie pour les solides: un format rationnel exact (où les grands entiers sont restitués sous forme de chaînes de caractères) et un format flottant approché qui sert notamment à la visualisation des résultats.

III. UTILISATION D'UNE ARITHMETIQUE RATIONNELLE

Parmi les solutions possibles au problème des imprécisions numériques, l'utilisation d'une arithmétique exacte est la plus simple et la plus commode pour les concepteurs d'algorithmes. Les algorithmes peuvent être implantés tels quels, sans se préoccuper des erreurs numériques.

III.1. Implications

Malheureusement, les implications en temps et en mémoire des arithmétiques exactes découragent souvent les concepteurs d'algorithmes, de sorte que le domaine d'application de ces arithmétiques se trouve limité au *Calcul Formel* et à quelques rares algorithmes géométriques: [GY-86], [Mich-87], [KLN-89] et [More-90], entre autres.

Nous avons nous-mêmes utilisé une arithmétique rationnelle pour implanter l'algorithme d'intersection de polyèdres rationnels. Les performances obtenues sont très médiocres, mais prévisibles. Entre autres contraintes, la récupération de l'espace mémoire occupé par les rationnels devenus inutiles est impérative ; elle n'est malheureusement pas automatique en C ou en C++ (les langages les plus couramment utilisés en synthèse d'images). Notre implantation en C++ a nécessité l'écriture d'une procédure de *glanage* de cellules ; celle-ci est aussi utilisée pour récupérer l'espace mémoire occupé par les autres données (faces, arêtes, pans, ...) devenues inutiles. Par ailleurs, la librairie rationnelle utilisée [More-90] n'était pas elle-même très performante (utilisation de listes au lieu de tableaux dynamiques).

Notre deuxième objectif fut alors de réfléchir aux optimisations possibles qui ramèneraient les temps de calculs à des proportions raisonnables. Il faut dire que notre objectif était d'emblée la conception et l'implantation d'un algorithme qui produise toujours des solides *cohérents* à partir d'autres solides *cohérents*.

III.2. Une première optimisation : le calcul "réticent"

De nombreux tests géométriques (par exemple, les tests *point/plan* ou *point/droite*) se ramènent à l'étude du *signe* de certains résultats numériques (typiquement, des *produits scalaires* ou des *déterminants*). Ces données, bien que *temporaires* (elles ne sont pas stockées dans les structures de données), étaient systématiquement calculées en arithmétique rationnelle.

En fait, la plupart des décisions géométriques auraient pu être prises sur la base de calculs combinant précision flottante et précision rationnelle de manière à garantir la *cohérence* des décisions. Dans [KLN-89] et [GT-91] les auteurs exploitent la technique d'*encadrement* par intervalles *entiers* ou *flottants*, respectivement.

Principe

Moreau [More-91] décrit une approche mixte *Flottant/Rationnel* dite "réticente", en ce sens qu'elle ne fait appel à l'arithmétique rationnelle qu'en dessous de certains seuils, calculés (à la main) pour une application et une machine données. Chaque nombre rationnel a dispose d'une approximation flottante $\tilde{a}$ et chaque test géométrique est implanté comme une fonction F dotée d'un *seuil de fiabilité* précalculé ε_F.

A chaque appel de F, le test est d'abord effectué entièrement en arithmétique flottante, puis, si le résultat n'est pas compatible avec ε_F, et seulement dans ce cas, l'ensemble du test est réexécuté en arithmétique rationnelle.

Par exemple, la comparaison "réticente" de deux nombres est implantée sous la forme suivante :

> *comparaisonRéticente* (a, b: Rationnel)
> <u>début</u>
>   si ($\tilde{a} < \tilde{b} - \varepsilon_{comp}$) <u>rendre</u> "$a < b$" ;
>   si ($\tilde{a} > \tilde{b} + \varepsilon_{comp}$) <u>rendre</u> "$a > b$" ;
>   <u>rendre</u> *comparaisonExacte* (a, b)
> <u>fin.</u>

Etant donnée une *dynamique* pour <u>tous</u> les nombres susceptibles d'apparaître dans l'application, les epsilons sont calculés une fois pour toutes pour chaque test géométrique.

Cette approche nécessite une refonte importante des programmes et des structures de données et un calcul laborieux d'epsilons. Nous ne l'avons intégrée que de manière superficielle (notamment, nous n'avons pas systématisé le calcul des epsilons). L'algorithme résultant est à peine moins lent que la version rationnelle, tout au plus nous gagnons un facteur 2 (sans doute parce que nous nous sommes montrés trop "prudents" dans le choix des epsilons).

IV. UTILISATION D'UNE ARITHMETIQUE PARESSEUSE

La raison pour laquelle la version rationnelle de l'algorithme est très lente est due au fait qu'un volume considérable de calculs est effectué exactement, bien que les valeurs exactes ne soient pas utilisées par la suite. Plus précisément, outre le constat fait en III.2 concernant les calculs *temporaires*, on peut faire un constat analogue pour les données géométriques *définitives*, destinées à être stockées dans les structures de données (essentiellement, les coordonnées des sommets ou des points d'intersection et les coefficients des équations de plans).

IV.1. Motivations

Les données géométriques sont calculées *exactement* même si un grand nombre d'entre elles finissent par être "inutiles" dans un certain sens (par exemple, lors du calcul de la frontière ∂S du solide $S = A \cap B$, les sommets de *A classifiés A-hors-S* ne sont pas retenus dans la description de ∂S). En général, ces données "inutiles" ne peuvent pas être évitées systématiquement dans les algorithmes de résolution d'opérations booléennes, du fait que ces algorithmes sont fondés sur le principe même de "génération et test" (on y reviendra au chapitre IV). Il n'en demeure pas moins que si un sommet finit par être "inutile", ses coordonnées auront été déterminées *inutilement* par des calculs exacts. On peut penser alors que ce sommet aurait pu être représenté et manipulé seulement par une *approximation suffisante*.

Sur la base des deux constats précédents, un groupe de recherche de l'Ecole Nationale Supérieure des Mines de Saint-Etienne a introduit le concept original d'*arithmétique rationnelle paresseuse* [BJMM-93a]. Le principe fondamental de cette arithmétique consiste à combiner calculs *flottants* et calculs *rationnels* de façon que ces derniers soient toujours *retardés* jusqu'à ce qu'ils deviennent éventuellement indispensables. Le groupe des "paresseux" est composé de D. Michelucci, l'"initiateur", de J.M. Moreau et de deux chercheurs en thèse: P. Jaillon et moi-même.

Nous présentons ci-dessous les points clés de cette arithmétique paresseuse.

IV.2. Représentation des nombres paresseux

Chaque nombre *paresseux* x est représenté par un *intervalle flottant* contenant la valeur rationnelle exacte de x et par une *définition* qui permet de calculer au besoin cette valeur exacte. Une définition peut être soit un nombre rationnel (la valeur même de x) représenté d'une manière ou d'une autre, soit une expression symbolique représentant x comme la *somme*, le *produit*, l'*opposé* ou l'*inverse* d'autres nombres *paresseux*. Les expressions de la forme $a-b$ ou a/b sont représentées par $a+(-b)$ et $a*(1/b)$, respectivement.

Une telle définition est toujours possible pour un nombre rationnel. Elle consiste fondamentalement en un *arbre* dont les nœuds internes désignent des opérateurs arithmétiques *unaires* ou *binaires* et dont les feuilles désignent des nombres rationnels ordinaires. Certains nœuds peuvent être partagés, de sorte que la structure des définitions est celle d'un *graphe orienté sans cycles*, plutôt que la structure d'*arbre*.

IV.3. Opérations sur les nombres paresseux

Chaque opération élémentaire (*somme*, *produit*, *opposé*, *inverse*) s'effectue en général en temps *constant*. Elle consiste simplement à créer un nouveau nœud pour le nombre paresseux résultat. Ce nœud stocke: l'opérateur demandé, la (ou les) référence(s) aux nombre(s) paresseux opérande(s) et un intervalle construit sur ceux des opérandes. La valeur exacte du résultat n'est pas calculée immédiatement, elle restera inconnue dans la plupart des cas.

Les règles utilisées pour la composition des intervalles s'inspirent de l'*arithmétique des intervalles* ([Moore-66], [Knut-81] ou [KM-82]. Soient $\boxed{\perp}$ la version flottante de l'opérateur arithmétique $\perp \in \{+, -, *, /\}$ et ∇ (resp. Δ) l'opérateur d'*arrondi* au nombre flottant immédiatement *inférieur* (resp. *supérieur*) [KM-82]. Etant donnés deux flottants u et v, $(u \perp v)$ n'est pas en général un nombre flottant, mais l'intervalle $[\nabla(u \boxed{\perp} v), \Delta(u \boxed{\perp} v)]$ en fournit le meilleur encadrement flottant.

Arithmétique d'intervalles

En dehors des *débordements* [Knut-81], les règles ci-dessous s'appliquent à tous les nombres paresseux x et y, d'intervalles respectifs $[x_1, x_2]$ et $[y_1, y_2]$:

$$[x_1, x_2] + [y_1, y_2] = [\nabla(x_1 \boxed{+} y_1), \Delta(x_2 \boxed{+} y_2)] \,;$$
$$[x_1, x_2] * [y_1, y_2] = [\nabla(\mathrm{Min}(x_i \boxed{*} y_j)), \Delta(\mathrm{Max}(x_i \boxed{*} y_j))] \,;$$
$$-[x_1, x_2] = [\boxed{-} x_2, \boxed{-} x_1] \,;$$
$$[x_1, x_2]^{-1} = [\nabla(1 \boxed{/} x_2), \Delta(1 \boxed{/} x_1)], \text{ pourvu que } 0 \notin [x_1, x_2].$$

Le principe fondamental de l'arithmétique paresseuse repose sur le fait que les intervalles d'encadrement sont souvent *suffisants* pour la *comparaison* ou le calcul du *signe* des nombres paresseux. A l'évidence, deux nombres paresseux d'intervalles *disjoints* peuvent être ordonnés sans l'aide de leurs valeurs exactes De même, le signe d'un nombre paresseux dont l'intervalle ne *contient pas zéro* se détermine trivialement.

IV.4. Evaluation des nombres paresseux

La valeur exacte n'est calculée explicitement que dans l'une des trois situations suivantes:

1) lorsqu'on veut déterminer le *signe* d'un nombre paresseux dont l'intervalle *contient zéro* ou plus généralement lorsqu'on veut comparer deux nombres paresseux dont les intervalles *ne sont pas disjoints* ;

2) lorsqu'on veut calculer l'*inverse* d'un nombre paresseux dont l'intervalle *contient zéro* ;

3) lorsqu'on veut la valeur exacte d'un autre nombre paresseux qui référence le nombre considéré.

Dans les cas (1) et (2), on peut parfois conclure sans recourir à l'évaluation *totale* de l'arbre définissant le nombre paresseux. La méthode consiste à évaluer seulement les valeurs aux nœuds les plus "profonds" de l'arbre (par exemple les nœuds juste au "dessus" des feuilles), puis remonter l'arbre en *précisant* les intervalles par une technique analogue à celle de Cameron [Came-91]. Le plus souvent, le nouvel intervalle obtenu au nœud racine devient suffisant (il ne contient plus *zéro*).

D'autres techniques d'optimisation sont détaillées dans [Jaill-93], le but étant d'éviter au maximum l'évaluation systématique (coûteuse) des nombres paresseux. Lorsque cette évaluation est inévitable, la méthode la plus simple consiste en une procédure qui calcule récursivement la valeur en chaque nœud de l'arbre définissant le nombre paresseux

IV.5. Librairie arithmétique paresseuse

Une *librairie* C++ indépendante d'*arithmétique paresseuse* a été implantée pour moitié par P. Jaillon [Jaill-93] et pour autre moitié par les autres membres du groupe des "paresseux" [BJMM-93a]. Elle est accessible à tous les programmes d'application en C ou C++, sans modifications notables des sources. Il suffit de remplacer les types standard float ou double par le type Lazy (via une macrodéfinition #define): le langage C++ permet la redéfinition des opérateurs arithmétiques.

La librairie assure sa propre gestion mémoire grâce à un module glanage basé sur la technique des *compteurs de références*. Bien entendu, c'est à la charge du programmeur d'allouer et de libérer la mémoire correspondant à ses structures de données: par exemple, des sommets ou des plans.

Les premiers programmes testant et utilisant la librairie paresseuse sont dus à D. Michelucci (algorithme de Bentley-Ottman [BO-79]) et à moi-même (intersection de polyèdres [BMP-93]). Ces programmes ont notamment permis d'améliorer et d'enrichir la librairie par de nouvelles notions: gestion des définitions *isomorphes* (Sect. V.1), formulation du concept de programmation *avertie/naïve* (Sect. V.4), calcul de clés pour l'*adressage dispersé* (Sect. VI), ...

V. IMPLICATIONS DE L'ARITHMETIQUE PARESSEUSE

L'utilisation de la librairie paresseuse n'est pas totalement *transparente* vis à vis des programmes d'application: elle est seulement aussi transparente que l'utilisation de l'arithmétique flottante standard. Pour un même algorithme, les performances de la librairie paresseuse varient d'une *implantation* à l'autre. Autrement dit, la librairie profitera davantage à un programmeur *averti* qu'à un programmeur non expérimenté qui méconnaîtrait (ou négligerait) le mécanisme de propagation de l'imprécision numérique. Nous expliquerons ce phénomène dans le cas des algorithmes géométriques et en particulier dans le cas de l'algorithme d'intersection de polyèdres.

V.1. Expressions morphologiquement équivalentes

Considérons un programme C ou C++ exploitant une arithmétique paresseuse, qui effectue le test suivant, d'apparence anodine :

if ($\psi(a) == \psi(b)$) ... ,

où ψ est une certaine fonction rendant un nombre paresseux, a et b étant typiquement des pointeurs sur des structures représentant deux entités géométriques (par exemple des sommets).

Dans le cas où a et b sont deux pointeurs sur la même entité, la première version de l'arithmétique paresseuse évaluait les deux expressions $\psi(a)$ et $\psi(b)$, puis comparait les valeurs rationnelles obtenues. Dans ce cas, le test ci-dessus coûtait inutilement deux évaluations et une comparaison rationnelles.

Pourtant, ces calculs rationnels auraient pu être facilement évités si le programmeur avait plutôt programmé son test sous la forme:

if (($a == b$) || ($\psi(a) == \psi(b)$)) ...

V.1.1. Notion d'isomorphisme d'expressions

En fait, le problème ci-dessus est suffisamment simple pour être résolu au niveau de l'arithmétique paresseuse elle-même. La solution consiste, lors de la comparaison des nombres paresseux, à prendre en compte le cas des nombres paresseux ayant des définitions *isomorphes*[2].

La détection d'isomorphisme entre deux nombres paresseux a et b s'effectue selon le schéma récursif suivant :

```
Isomorphes (a, b : Paresseux) : Booléen
début
    si (a = b) rendre vrai ; { a et b sont à la même adresse mémoire }
    si (a.intervalle ∩ b.intervalle = ∅) rendre faux ;
    si (classe (a) ≠ classe (b)) rendre faux ;
    si (classe (a) = Opposé ou classe (a) = Inverse)
        rendre Isomorphes (a.filsUnique, b.filsUnique) ;
    si (classe (a) = Somme ou classe (a) = Produit)
        rendre Isomorphes (a.filsGauche, b.filsGauche) et
               Isomorphes (a.filsDroit, b.filsDroit) ;
    rendre feuillesEgales (a, b) { comparaison de deux rationnels }
fin.
```

A priori, on pourrait craindre une très grande lenteur du test d'isomorphisme ; notamment dans le cas de deux expressions de grande taille ne différant qu'aux feuilles: deux *déterminants* par exemple. Cependant, la prise en compte des intervalles fait que, statistiquement, le non isomorphisme est détecté en temps quasi constant. En Section VI, nous verrons comment la notion de *"clé modulo p"* permet d'accélérer aussi le test d'isomorphisme.

V.1.2. Exploitation de l'isomorphisme

Le test d'isomorphisme a été effectivement implanté dans une deuxième version de la librairie paresseuse [Jaill-93] ; il permet de détecter l'égalité de nombres paresseux isomorphes, sans évaluation. D'une manière générale, la fonction ci-dessous montre comment exploiter l'isomorphisme (et les intervalles), lors des comparaisons :

---

[2] Le terme d'isomorphisme est utilisé dans un sens moins strict que celui qu'on lui connait en théorie des graphes. Informellement, deux nombres paresseux isomorphes ont la même définition mathématique.

```
comparaisonParesseuse (a, b : Paresseux): -1..1
début
    { Cette fonction rend le signe de (a-b) }
    si (a = b) rendre 0 ; { a et b sont à la même adresse mémoire }
    si (a.intervalle ∩ b.intervalle = ∅) alors
        rendre si (a.intervalle.min > b.intervalle.max) alors +1 sinon -1 ;
    si Isomorphes (a, b) rendre 0 ;
    rendre comparaisonExacte (a, b)
fin.
```

Pour l'instant, l'isomorphisme n'est testé que lors des comparaisons. Une variante intéressante serait de le détecter systématiquement à la création de chaque nombre paresseux, de manière à ne stocker qu'une et une seule fois tous les arbres isomorphes. Alors, l'égalité de deux nombres paresseux isomorphes se réduirait à un test d'égalité de deux adresses.

V.2. Expressions non isomorphes mais algébriquement équivalentes

Le test d'isomorphisme ne résout pas tous les problèmes. Même si l'isomorphisme est géré à la création des nombres, il reste à détecter l'égalité de deux nombres paresseux non isomorphes mais *algébriquement* équivalents telles que $(a*c + b*c)$ et $(a + b)*c$, où a, b et c sont des feuilles ou des sous-arbres partagés. Notons que deux expressions ne partageant aucune feuille ou sous-arbre ne peuvent être algébriquement équivalentes, mais deux expressions algébriquement équivalentes n'ont pas nécessairement le même ensemble de feuilles: par exemple $(c + a*b) - c$ et $b*a$.

Malheureusement, l'arithmétique paresseuse est en général incapable de conclure à l'égalité de deux expressions non isomorphes et algébriquement équivalentes, autrement que par une évaluation rationnelle. Tout au plus, la librairie paresseuse sait-elle détecter l'égalité de deux expressions simples telles que $(a+b)$ et $(b+a)$ ou $(a*b)$ et $(b*a)$, grâce à la technique présentée en Section VI.

La librairie intègre tout de même les simplifications symboliques très limitées, du type:

$a + 0 \rightarrow a$,	$a * 1 \rightarrow a$, (élément neutre) ;
$a * 0 \rightarrow 0$,	(élément absorbant) ;
$-(-a) \rightarrow a$,	$1/(1/a) \rightarrow a$, (idempotence) ;
$a + (-a) \rightarrow 0$,	$a * (1/a) \rightarrow 1$,(éléments inverses).

Il est vrai que les logiciels de *calcul formel* savent détecter toutes les identités algébriques, voire l'équivalence de deux systèmes d'équations algébriques (par exemple via les bases de Gröbner [WBLR-85]), mais les temps de calcul sont nécessairement très importants.

V.3. Programmation avertie en arithmétique flottante

Le problème des expressions non isomorphes et algébriquement équivalentes se pose également en arithmétique flottante, où les opérations de multiplication ou d'addition ne préservent plus les propriétés d'associativité ou de distributivité ([Mull-89], [Gold-91]).

Nous présentons ci-dessous quelques exemples de "savoir faire" utilisés par les programmeurs avertis pour éviter les méfaits de l'imprécision inhérente à l'arithmétique flottante.

Exemple 1

Considérons un algorithme géométrique qui détermine le point d'intersection I entre deux droites du plan (D) : $ax + by + c = 0$ et (D') : $a'x + b'y + c' = 0$. Analytiquement, les coordonnées X et Y de I sont données par les formules exactes :

$$X = \frac{b*c' - b'*c}{a*b' - a'*b} \text{ et } Y = \frac{a*c' - a'*c}{a*b' - a'*b},$$

mais leur calcul en arithmétique flottante est toujours entaché d'erreur.

Ultérieurement, l'algorithme peut avoir besoin de déterminer la position de certains points par rapport à certaines droites, par un calcul de *puissances*. En particulier, les puissances $(aX + bY + c)$ et $(a'X + b'Y + c')$ du point I par rapport aux droites (D) et (D') ont fort peu de chances d'être nulles, en dehors de quelques cas particuliers. Une telle erreur est généralement fatale au bon déroulement du programme ou à la *cohérence* de ses résultats.

Pour surmonter le problème ci-dessus, un programmeur *averti* notera sur I l'information *symbolique* indiquant que ce point provient de l'intersection des deux droites (D) et (D'), information qu'il pourra utiliser lorsqu'il aura besoin de connaître la *puissance* de I par rapport à (D) ou à (D'). Cependant, cette solution nécessite une modification des structures de données et des algorithmes (typiquement, une adjonction de pointeurs).

Un autre bon réflexe de programmeur averti consiste à ne pas utiliser des expressions différentes (même algébriquement équivalentes) pour une coordonnée (disons) Y, telles que

$$\frac{a*c' - a'*c}{a*b' - a'*b}, \frac{-(a*X + c)}{b}, \text{ ou } \frac{-(a'*X + c')}{b'} ;$$

car, évaluées en arithmétique flottante, ces expressions donneraient des résultats différents.

Exemple 2

Dans certains cas, ce genre d'erreur peut être évité sans modification explicite des structures de données. Ainsi, dans les algorithmes d'intersection de polyèdres, où les arêtes des diverses faces sont souvent *orientées* selon une certaine convention, il faut prendre soin de calculer les intersections toujours de la même façon. Par exemple, une même arête géométrique AB, adjacente à deux faces distinctes f et g, est utilisée comme $\overrightarrow{AB}$ dans f et comme $\overrightarrow{BA}$ dans g. Dans ces conditions, l'intersection d'un plan P avec successivement f et g donne deux points $I_f = \overrightarrow{AB} \cap P$ et $I_g = \overrightarrow{BA} \cap P$ dont les coordonnées ne s'expriment pas par des expressions *isomorphes*: l'évaluation en arithmétique flottante donne deux résultats différents.

Une solution simple consiste à toujours considérer les arêtes sous une forme telle que l'extrémité initiale est *lexicographiquement* inférieure à l'extrémité finale. Ainsi, les coordonnées de I_f et I_g seront-elles *isomorphes*, donc égales après évaluation en flottant.

Cependant, comme le montre le troisième exemple ci-dessous, il convient d'être prudent lorsqu'on effectue un tri lexicographique sur des coordonnées flottantes.

Exemple 3

Dans les algorithmes d'intersection, les cas particuliers donnent souvent prise aux incohérences, du fait de l'imprécision numérique inhérente à l'arithmétique flottante. Ainsi, dans l'algorithme Bentley-Ottman [BO-79] pour l'intersection de segments du plan, les segments "verticaux", parallèles à la barre de balayage, requièrent un soin particulier, faute de quoi on risque d'altérer la cohérence globale de l'*y-ordre* maintenu lors du balayage (Fig. 2).

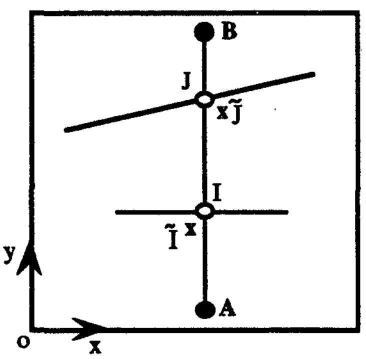


Figure 2. Si l'on n'y prend pas garde, le calcul flottant donne des valeurs approchées $\tilde{I}$ et $\tilde{J}$ pour les points d'intersection I et J, le long de la droite (A,B): ceci est surtout grave pour les abscisses. Le tri lexicographique donne (pour cet exemple): $\tilde{I} < A < B < \tilde{J}$ alors que l'ordre correct est $A < I < J < B$.

Dans le même ordre d'idées, nous avons constaté que la version flottante, *naïve*, de l'algorithme d'intersection avortait souvent lorsque l'objet CSG comportait des faces "verticales" ou "horizontales" (parallèles aux plans de coordonnées). Nous décrirons plus loin l'artifice de programmation qui a permis de résoudre ce problème.

V.4. Comment tirer le meilleur parti de l'arithmétique paresseuse ?

Le savoir faire relevant de la programmation *avertie* permet d'éviter les conséquences désastreuses de l'imprécision numérique, dans de nombreux cas géométriques *simples*. Cependant, les cas *complexes* requièrent une arithmétique exacte ou du moins des techniques beaucoup plus sophistiquées, par exemple le *raisonnement symbolique* ([HHK-89], [Kara-89]).

L'arithmétique rationnelle paresseuse évite toutes les imprécisions numériques, par contre ses performances dépendent sensiblement du mode de programmation des algorithmes d'application.

V.4.1. Programmation avertie versus programmation naïve

En programmation *avertie*, les seuls calculs rationnels effectués sont ceux nécessaires aux cas *complexes*: les cas *simples* étant résolus par les techniques relevant de l'expérience du programmeur (modification idoine des structures de données, génération d'expressions *isomorphes*, ...).

En programmation *non avertie* (ou *naïve*), même les cas *simples* provoquent des évaluations rationnelles de la part de la librairie paresseuse.

Donc, pour tirer le meilleur parti de l'arithmétique paresseuse, il vaut mieux d'abord implanter une version *flottante* standard, qui soit la plus *robuste* possible ; ensuite, lorsque tous les cas géométriques *simples* ont été résolus et que seuls les cas *complexes* mettent l'algorithme en échec, on fait alors appel à l'arithmétique paresseuse qui achève de résoudre ces derniers cas, et seulement ceux-là, par des calculs rationnels.

Nous avons pu vérifier le bien fondé de cette démarche lorsque nous avons tenu compte du cas particulier des faces "horizontales" ou "verticales", souvent rencontrées en modélisation de solides polyédriques.

V.4.2. Application à l'algorithme d'intersection

Lors de l'intersection de deux faces f et g issues de deux solides distincts, chaque arête $[p_0, p_1]$ d'une face (disons f) est intersectée avec le plan Γ de l'autre face (g).

Soit $ax + by + cz + d = \Gamma(x, y, z) = 0$, l'équation définissant Γ (les équations de plans étant "normalisées" de façon que la *première* composante *non nulle* du vecteur *normal* (a, b, c) ait pour valeur ± 1).

V.4.2.1. Méthode d'intersection naïve

Dans la première version de l'algorithme, le point d'intersection $I = (X, Y, Z)$ entre l'arête $[p_0, p_1]$ et Γ était calculé par la méthode classique suivante :

$\alpha_0 := \Gamma(x_0, y_0, z_0)$ (*puissance* de $p_0 = (x_0, y_0, z_0)$ par rapport à Γ) ;
$\alpha_1 := \Gamma(x_1, y_1, z_1)$ (*puissance* de $p_1 = (x_1, y_1, z_1)$ par rapport à Γ) ;
$\lambda := - \alpha_0 / (\alpha_1 - \alpha_0)$ ($\lambda \in [0, 1]$ tel que $I = p_0 + \lambda^*(p_1 - p_0)$) ;
$X := x_0 + \lambda^* (x_1 - x_0)$;
$Y := y_0 + \lambda^* (y_1 - y_0)$;
$Z := z_0 + \lambda^* (z_1 - z_0)$.

Implanté de cette manière, ce calcul d'intersection causait souvent un arrêt anormal de la version flottante de l'algorithme, lorsque les primitives de l'objet CSG comportaient des faces "horizontales" ou "verticales". Cependant, nous avons constaté que l'erreur fatale au programme disparaissait lorsque nous appliquions une *rotation aléatoire* à l'ensemble de l'arbre CSG (ou à certaines des primitives).

Evidemment, l'utilisation de la librairie d'arithmétique paresseuse permettait toujours un déroulement correct du programme, mais des calculs rationnels coûteux étaient effectués, qui auraient pu être évités simplement par quelques "précautions" de programmation. Nous sommes arrivés à la conclusion ci-dessous.

V.4.2.2. Inconvénients de la méthode naïve

Dans le cas d'une face g "horizontale" ou "verticale", l'équation du plan Γ est de la forme: $ax + d = 0$, $by + d = 0$, ou $cz + d = 0$. Pour fixer les idées, supposons que Γ soit de la forme $\Gamma(x, y, z) = x + d = 0$. Alors tout point d'intersection I entre Γ et une arête $[p_0, p_1]$ de f a une abscisse X dont la valeur $(-d)$ est indépendante des coordonnées de p_0 et de p_1. Pourtant, X était calculé invariablement sous la forme :

$\alpha_0 = a^*x_0 + b^*y_0 + c^*z_0 + d$;
$\alpha_1 = a^*x_1 + b^*y_1 + c^*z_1 + d$;
$X = x_0 - \alpha_0 / (\alpha_1 - \alpha_0)^*(x_1 - x_0)$.

Une fois que tous les points d'intersection entre les arêtes de f et Γ étaient déterminés (Chap. II, Sect. IV.2.2.1), l'algorithme effectuait un tri *lexicographique* de tous ces points, le long de la droite d'intersection entre les plans de f et g.

En arithmétique flottante, les diverses valeurs de X sont généralement différentes, si bien que le tri lexicographique donnait des résultats incohérents.

Le même problème se posait en 2D avec l'algorithme de Bentley-Ottman [BO-79], lorsqu'on trie les points d'intersection entre un segment "vertical" (parallèle à la droite de balayage) et des segments "obliques".

L'arithmétique paresseuse, elle, est forcée de recourir à des calculs exacts car elle doit comparer des abscisses X égales (à $-d$) mais définies par des expressions *non isomorphes*.

V.4.2.3. Prise en compte des faces horizontales ou verticales

L'idée est de particulariser les plans "horizontaux" ou "verticaux", par exemple en notant cette information dans un champ supplémentaire associé à chaque enregistrement "Plan". Le *type* de chaque plan (*X-constant*, *Y-constant*, *Z-constant*, ou *Oblique*) est déterminé une, et une seule fois, à la création du plan. Le coût de cette opération est largement compensé par la suite: deux plans non *Obliques* de même type (resp. de types distincts) peuvent être déclarés *parallèles* (resp. *intersectants*), sans aucun calcul.

Grâce à cette optimisation très simple, le programme est devenu plus robuste en arithmétique flottante, et plus rapide en arithmétique paresseuse: elle évite en effet de nombreux calculs rationnels, effectués dans la version *non avertie* de l'algorithme.

V.4.3. Résumé

L'imprécision numérique étant la principale source d'incohérences topologiques dans les algorithmes géométriques, l'utilisation d'une arithmétique rationnelle paresseuse constitue une solution aussi radicale mais plus économique qu'une arithmétique rationnelle classique. L'arithmétique paresseuse combine de façon cohérente calculs *flottants* et calculs *rationnels* de telle manière que les algorithmes d'application n'ont à intervenir à aucun moment dans le choix de l'un ou l'autre niveau de précision. Parmi tous les calculs demandés, seuls les calculs rationnels strictement nécessaires sont effectués.

Cependant, les utilisateurs de l'arithmétique paresseuse doivent tenir compte des points suivants :

• La programmation *non avertie* conduit à des programmes qui sont très instables en arithmétique flottante et fiables mais trop lents en arithmétique rationnelle paresseuse.

• La programmation *avertie* permet d'obtenir des algorithmes relativement robustes en arithmétique flottante, et totalement robustes, et rapides, en arithmétique paresseuse.

• L'arithmétique paresseuse est d'un emploi aussi transparent que celui de l'arithmétique flottante... mais pas davantage .

• En pratique, il vaut mieux implanter d'abord une version *flottante* standard, qui résolve les cas *simples*, et ensuite seulement faire appel à l'arithmétique paresseuse pour résoudre les cas *complexes*. Cette démarche obéit bien au principe fondamental de l'arithmétique paresseuse: ne recourir aux calculs rationnels que lorsque les calculs flottants donnent des résultats non fiables.

• La programmation *avertie* est peu naturelle. L'idéal serait que les *compilateurs* eux-mêmes, ou des *préprocesseurs*, sachent transformer automatiquement un programme *naïf* en un programme *averti* équivalent, via les manipulations symboliques et algébriques adéquates. Automatiser une telle transformation paraît difficile. Rappelons que ce problème n'est pas introduit par l'arithmétique paresseuse: il se pose déjà avec l'arithmétique flottante, qui fait échouer des programmes corrects écrits de façon naïve. Nous n'avons pas trouvé mention de ce problème dans la littérature.

VI. CLES MODULAIRES EN ARITHMETIQUE PARESSEUSE

L'algorithme d'intersection utilise intensivement les tables à *adressage dispersé* (ou "Hash-Tables" [Knut-81]), qui permettent d'accélérer les recherches dans les listes de sommets ou de plans de faces. L'utilisation de l'arithmétique *paresseuse* pose alors un problème: les coordonnées des sommets ou les coefficients des plans étant représentés par des nombres paresseux (dont la valeur n'est pas toujours disponible), comment leur attacher des *clés* de codage ("Hash-Codes") sans évaluer les nombres paresseux ? Une autre difficulté est liée au fait qu'il faut associer impérativement la même *clé* à deux nombres de même *valeur* mais de définitions différentes, par exemple les nombres "2 * 4" et "3 + 5". A priori, les intervalles des nombres paresseux ne peuvent pas servir de *clés*.

Une première solution est de ne plus utiliser de telles tables, et donc de modifier l'algorithme. Il est cependant préférable d'enrichir la librairie d'arithmétique paresseuse, en permettant d'associer une *clé* à tout nombre paresseux.

Dans cette section, nous construisons une fonction H qui, pour $p \in N^*$ et $\Omega \notin [0, p[$ fixés, associe à tout nombre paresseux a une *clé* $H(a) \in [0, p[\cup \{\Omega\}$, telle que:

$$H(a_1) \neq H(a_2) \Rightarrow valeur(a_1) \neq valeur(a_2).$$

Nous montrons que dans la plupart des cas cette fonction n'a pas besoin d'évaluer les nombres paresseux. D'autre part, H rend toujours la même *clé* pour des nombres de même *valeur* mais définis par des expressions différentes. Moyennant cette fonction H, il devient possible de définir une fonction ψ qui attache à chaque sommet ou plan X une *clé* $\psi(X) \in [0, p[\cup \{\Omega\}$ vérifiant:

$$\psi(X_1) \neq \psi(X_2) \Rightarrow X_1 \neq X_2.$$

VI.1. Le corps Z/pZ

Etant donné un nombre entier p, *premier*, la relation binaire $x \equiv y\ [p]$ (lire x congru à y modulo p) définie sur Z par :

$$x \equiv y\ [p] \Leftrightarrow (x - y) \in pZ\ (\Leftrightarrow \exists k \in Z \mid (x - y) = k * p)$$

induit une partition de Z en p classes d'équivalence qui constituent le *corps fini* Z_p (ou Z/pZ).

Les propriétés de Z_p sont bien connues. En particulier, pour tout $y \in Z_p^*$, il existe $y^{-1} \in Z_p^*$ tel que $(y * y^{-1}) \equiv 1\ [p]$ (y^{-1} est l'*inverse* de y dans Z_p). Le calcul des *inverses* dans Z_p sera abordé en Section VI.4. Chaque classe C de Z_p est caractérisée par son unique représentant $r \in [0, p[$ qui vérifie $x \equiv r\ [p]$, $\forall x \in C$. Autrement dit, r est le *reste* de la division euclidienne de x par p (noté $x \% p$ dans toute la suite). Nous désignerons par $GCD(x, y)$ le plus grand diviseur commun à deux éléments x et y appartenant à Z.

VI.2. Clé d'un nombre rationnel

La *clé* d'un nombre *rationnel* irréductible x/y ($x \in Z$, $y \in Z^*$, $GCD(x, y) = 1$) est définie par:

$$H(x/y) = \left\{ \begin{array}{c} (x * y^{-1}) \% p, \text{ si } y \notin pZ \\ \Omega, \text{ sinon} \end{array} \right\},$$

où Ω est un entier fixé une fois pour toutes, vérifiant $\Omega \notin [0, p[$ (en pratique, il suffit de prendre $\Omega = p$). Nous compléterons en définissant les deux règles cohérentes $0^{-1} = \Omega$ et $\Omega^{-1} = 0$.

En fait, pour le calcul de $H(x/y)$, $y \notin pZ$, le rationnel x/y n'a pas besoin d'être irréductible. En effet, soient u et v les deux entiers de Z, tels que $x = d * u$ et $y = d * v$, où $d = GCD(x, y)$. Du fait que $y \notin pZ$, il découle $d \notin pZ$ et $v \notin pZ$, de sorte que:

$$x * y^{-1} = (d * u) * (d^{-1} * v^{-1}) = u * v^{-1}.$$

VI.3. Clé d'un nombre paresseux

Voyons maintenant comment calculer la clé d'un nombre rationnel paresseux de valeur non nécessairement connue. Récursivement, un nombre paresseux c est soit un *rationnel* de valeur connue, soit un nombre défini comme la *somme* $(a + b)$ ou le *produit* $(a * b)$ de deux nombres paresseux a et b, ou comme l'*opposé* $(-a)$ ou l'*inverse* $(1/a)$ d'un nombre paresseux a.

Les propriétés bien connues du corps Z_p permettent très souvent de calculer la clé de c, directement à partir des clés de a et b :

• Dans le cas général où $H(a) \neq \Omega$ et $H(b) \neq \Omega$, on applique récursivement les propriétés suivantes :

$H(a + b) = (H(a) + H(b)) \% \ p$;

$H(a * b) = (H(a) * H(b)) \% \ p$;

$H(-a) = (-H(a)) \% \ p = p - H(a)$;

$H(1/a) = (H(a))^{-1} \% \ p$, sachant que $0^{-1} = \Omega$.

• Dans le cas particulier où $H(a) = \Omega$ ou $H(b) = \Omega$, toute clé $h \in [0, p[$ obéit aux règles ci-dessous, faciles à vérifier :

$\Omega + h = h + \Omega = \Omega, \ \forall h \in [0, p[$;

$\Omega * h = h * \Omega = \Omega, \ \forall h \in [1, p[$;

$\Omega * \Omega = \Omega$;

$-\Omega = \Omega$;

$\Omega^{-1} = 0$, par définition.

Dans les deux cas ci-dessus, la clé d'un nombre est calculée sans faire intervenir explicitement sa valeur rationnelle, laquelle peut d'ailleurs être inconnue. Puisque la clé d'un *rationnel paresseux* est une fonction du *rationnel*, on est assuré de toujours trouver la même *clé* quelles que soient les expressions arithmétiques dont ce rationnel est la valeur: par exemple, le rationnel 8 a bien la même clé, qu'il soit défini par "2 * 4" ou par "3 + 5".

• Il reste deux cas indésirables ($\Omega + \Omega$) et ($0 * \Omega$ ou $\Omega * 0$), conduisant à une indétermination. Nous nous contenterons de deux exemples simples pour illustrer cette indétermination :

***Exemple* 1** : $\Omega + \Omega = ?$

Pour $a = 1/p$ et $b = k - 1/p$, $k \in [0, p[$, on a $H(a) = H(b) = \Omega$ et $H(a + b) = H(k) = k$. Par conséquent, $H(a + b)$ peut prendre toute valeur k dans $[0, p[$.

D'autre part, pour $a = b = 1/p$, on a $H(a) = H(b) = \Omega$ et $H(a + b) = H(2/p) = \Omega$, de sorte que $H(a + b)$ peut prendre également la valeur Ω.

***Exemple* 2** : $0 * \Omega = \Omega * 0 = ?$

Pour $a = p$ et $b = k/p$, $k \in [1, p[$, on a $H(a) = 0$, $H(b) = \Omega$ et $H(a * b) = H(k) = k$. Par conséquent, $H(a * b)$ peut prendre toute valeur k dans $[1, p[$.

Pour $a = p$ et $b = 1/p^2$, on a $H(a) = 0$, $H(b) = \Omega$ et $H(a * b) = H(1/p) = \Omega$, de sorte que $H(a * b)$ peut prendre la valeur Ω.

Enfin, pour $a = 0$ et $b = 1/p$, on a $H(a) = 0$, $H(b) = \Omega$ et $H(a*b) = H(0) = 0$, de sorte que $H(a * b)$ peut prendre aussi la valeur 0.

En cas d'indétermination, le plus simple pour déterminer la *clé* d'un nombre *paresseux* est de calculer systématiquement sa valeur rationnelle et d'en déduire la *clé* comme indiqué en Section VI.2.

En fait, pour p suffisamment grand, les cas d'*indétermination* deviennent assez rares, en pratique. Empiriquement, on constate qu'ils ne se produisent qu'une fois sur p opérations.

VI.4. Calcul d'inverse dans Z/pZ

Dans Z_p, la *somme*, le *produit* ou l'*opposé* se calculent en $O(1)$, ce qui n'est pas le cas du calcul d'*inverse*. Nous présentons, ci-dessous, deux algorithmes de calcul d'inverse dans Z_p.

Le premier algorithme est fondé sur le théorème de Fermat: pour tout $p \in N^*$, p *premier* et pour tout $x \in Z_p^*$, on a $x^{p-1} \equiv 1$ $[p]$. Par conséquent, $x^{-1} \equiv x^{p-2}$ $[p]$. Cet algorithme nécessite $O(\log(p))$ multiplications dans Z_p. Sur une machine 32-bits, tout entier *premier* p inférieur à 2^{16} garantit qu'aucune de ces multiplications ne provoquera de *débordement*.

Le deuxième algorithme est fondé sur le théorème de Bachet [Sch-86] (plus connu sous le nom de théorème de Bezout): pour tout $(x, y) \in Z^2$, il existe $(u, v) \in Z^2$ tel que $u * x + v * y = GCD(x, y)$. En appliquant l'algorithme étendu d'Euclide pour le calcul de $GCD(x, y)$, avec $y = p$, on obtient le couple (u, v) d'entiers qui vérifie la relation de Bezout $u * x + p * y = 1$ (car $GCD(x, p) = 1$). D'où $(u * x) \equiv 1\ [p]$ et donc $x^{-1} \equiv u\ [p]$. Cet algorithme nécessite également $O(\log(p))$ opérations élémentaires dans Z (théorème de Lamé [Knut-81], page 343). Pour tout *entier* machine p, ces opérations s'effectuent sans *débordement* dans l'arithmétique *entière* de la machine.

En pratique, il est plus judicieux de précalculer tous les inverses dans Z_p et de les mémoriser sous forme d'une *table d'inverses*.

Du fait que dans Z_p, on a:

$$\forall i \in]0, p[, (p-i)^{-1} = (-i)^{-1} = -(i^{-1}),$$

il suffit de stocker les inverses i^{-1} (i =1, 2, ..., q), où q = $(p-1)/2$. De cette manière, toute opération élémentaire dans Z_p s'effectue en $O(1)$, moyennant un *prétraitement* en $O(p*\log(p))$ et une table d'inverses de taille $O(p)$.

Sur une machine 32-bits, une valeur raisonnable pour p consiste en le plus grand entier *premier* inférieur à 2^{16} (soit $p = 65521$). Pour cette valeur, la table des inverses occupera environ 64 Ko de mémoire.

VI.5. Exploitation des clés au niveau de la librairie paresseuse

Le concept de *clé modulo p* est jusqu'ici présenté comme relevant des besoins des algorithmes géométriques qui utilisent la librairie d'arithmétique paresseuse. C'est précisément pour répondre aux besoins de l'algorithme d'intersection de polyèdres que le calcul de *clés* a été intégré à la librairie. Cependant, les clés peuvent servir également pour les besoins propres de la librairie paresseuse. En effet, lors de la comparaison de deux nombres paresseux a et b d'intervalles intersectants, il est souvent possible de conclure à la non égalité de a et b, sans recourir à leur évaluation, du fait que :

$$H(a) \neq H(b) \Rightarrow valeur(a) \neq valeur(b).$$

Toutefois, on ne peut pas *ordonner* deux nombres paresseux de clés distinctes.

Dans la version actuelle de la librairie paresseuse, les clés sont systématiquement calculées et stockées dès la création des nombres paresseux. Elles sont utilisées lors du test d'isomorphisme (deux nombres paresseux de clés différentes ne sont pas isomorphes).

VI.6. Une variante optimisée de calcul de clés

Dans cette section, nous présentons une variante intéressante de la méthode de calcul de clés modulo p. Elle consiste à représenter la clé d'un nombre paresseux a, *implicitement*, sous forme d'un *couple* (η, δ) d'entiers tel que:

$$H(a) = (\eta * \delta^{-1}) \% p, \text{ chaque fois que } \delta \neq 0.$$

Le couple associé à un *rationnel* x/y est tout simplement $(x \% p, y \% p)$. Tous les couples $(k * \eta, k * \delta)$, $k \neq 0$, représentent la même clé $H(a)$ et tous les couples $(k, 0)$ tels que $k \neq 0$ représentent la clé "infinie" Ω. Enfin, le couple spécial $(0,0)$ indique l'indétermination.

VI.6.1. *Opérations sur les clés*

Etant donnés deux nombres paresseux a et a' dont les clés sont définies par les couples $(\eta, \delta) \neq (0, 0)$ et $(\eta', \delta') \neq (0, 0)$, respectivement, les couples associés aux nombres paresseux $a+a'$, $a*a'$, $-a$ ou $1/a$ se déduisent, récursivement, à travers les règles suivantes :

$$(\eta, \delta) + (\eta', \delta') = (\eta * \delta' + \eta' * \delta, \delta * \delta');$$
$$(\eta, \delta) * (\eta', \delta') = (\eta * \eta', \delta * \delta');$$
$$-(\eta, \delta) = (-\eta, \delta);$$
$$\frac{1}{(\eta, \delta)} = (\delta, \eta), \text{ avec } \frac{1}{(0, \delta)} = (\delta, 0), \text{ par définition.}$$

On notera que toutes les opérations s'effectuent en temps constant, y compris la division qui se traduit maintenant par une simple *permutation* des éléments d'un couple: il n'y a plus besoin de calcul d'inverse.

Les quatre règles ci-dessus sont parfaitement équivalentes aux règles définies dans la première méthode (Sect. VI.3), y compris les règles particulières qui font intervenir Ω. Considérons, par exemple, la première règle :

- dans le cas général où $H(a) \neq \Omega$ et $H(a') \neq \Omega$ (i.e. $\delta \neq 0$ et $\delta' \neq 0$), on a:

$$\begin{aligned}(\eta, \delta) + (\eta', \delta') &= \eta * \delta^{-1} + \eta' * \delta'^{-1} \\ &= \eta * \delta^{-1} * \delta' * \delta'^{-1} + \eta' * \delta'^{-1} * \delta * \delta^{-1} \\ &= (\eta * \delta' + \eta' * \delta) * (\delta * \delta')^{-1} \\ &= (\eta * \delta' + \eta' * \delta, \delta * \delta');\end{aligned}$$

- dans le cas particulier où l'une des deux clés est Ω et l'autre diffère de Ω, on a nécessairement $\delta * \delta' = 0$ et $(\eta * \delta' + \eta' * \delta) \neq 0$. Par conséquent, on retrouve bien la règle: $\Omega + h = \Omega$, $\forall h \in [0, p[$.

VI.6.2. Indéterminations

Bien entendu, les cas d'indétermination de la première méthode ($\Omega+\Omega$, $0*\Omega$, $\Omega*0$) n'ont pas disparu. Ils apparaissent maintenant sous la forme :

$$(\eta, 0) + (\eta', 0) = (0, 0), \forall\eta \neq 0, \forall\eta' \neq 0 ;$$
$$(0, \delta) * (\eta', 0) = (0, 0), \forall\eta' \neq 0, \forall\delta \neq 0 ;$$
$$(\eta, 0) * (0, \delta') = (0, 0), \forall\eta \neq 0, \forall\delta' \neq 0.$$

En cas d'indétermination, le plus simple pour déterminer la clé d'un nombre paresseux consiste, là encore, à calculer explicitement sa valeur rationnelle x/y et à en déduire le couple qui définit la clé.

VI.6.3. Avantages de cette deuxième méthode

Cette deuxième méthode est plus "concise" que la première, car elle n'utilise qu'un seul jeu de règles pour la combinaison des clés.

Pour ses besoins propres, la librairie paresseuse n'a plus besoin de calcul d'inverse dans Zp: les couples associés aux clés suffisent pour la comparaison des clés des nombres paresseux. En effet, pour tous nombres a et a', de couples respectifs (η, δ) et (η', δ'), on a :

$$H(a) = H(a') \Leftrightarrow \eta * \delta^{-1} = \eta' * \delta'^{-1} \Leftrightarrow \eta * \delta' = \eta' * \delta.$$

Les valeurs *explicites* des clés ne sont pas calculées systématiquement, mais seulement à la demande des programmes d'application utilisant la librairie paresseuse, et donc beaucoup plus rarement[3]. Par conséquent, plutôt qu'une table d'inverses de taille $O(p)$, on peut utiliser l'un des algorithmes en $O(\log(p))$ décrits en VI.4, et envisager une plus grande valeur de p. Sur une machine 32-bits, l'algorithme étendu d'Euclide permet de prendre p égal au nombre de Mersenne $M = 2^{31} - 1$ (= 2147483647). Pour cette valeur, la probabilité d'une indétermination se réduit à $1/M \approx 4.65*10^{-10}$ (contre $1.5*10^{-5}$, pour $p = 65521$).

Les *débordements* lors des calculs dans Z/MZ peuvent être gérés selon la technique classique, décrite dans [Knut-81].

Cette méthode n'a pas encore été implantée dans la librairie paresseuse.

---

[3] Lorsque la clé d'un couple (η, δ) est calculée, on en profite pour remplacer le couple (η, δ) par le couple équivalent $(\eta * \delta^{-1}, 1)$. Cette mesure simple diminue le nombre d'inversions dans Z_p.

Remarques finales

Nous terminons cette Section sur les *clés modulo p* par les deux remarques suivantes :

• Les clés permettent de détecter les égalités simples entre $a+b$ et $b+a$, ou entre $a*b$ et $b*a$ (expressions non isomorphes mais algébriquement équivalentes): lors de la création du noeud résultat, les deux opérandes a et b sont ordonnés par valeurs croissantes des clés.

• Une solution efficace mais "aventureuse" consiste à décréter systématiquement égaux deux nombres paresseux de clés égales et d'intervalles intersectants, en arguant qu'il n'y a qu'une chance sur p de se tromper (p étant le nombre total de clés distinctes, il peut valoir 2 milliards !). Après tout, de nombreux auteurs d'algorithmes ne prennent pas moins de risques lorsqu'ils utilisent l'heuristique traditionnelle des *epsilons*, qui juge égaux deux nombres flottants différant de moins d'un certain epsilon, sans autre forme de procès [LTH-86]. Cette "aventure" n'a pas été tentée.

VII. RESULTATS EXPERIMENTAUX

L'algorithme a été implanté en langage C++, sur une station HP/apollo Série 400 (processeur 68040 à 33 MHz, 16Mo de RAM et 400 Mo de disque), sous le système Domain OS Ver. 10.4. Il comporte environ 6000 lignes de programme. Il a été intégré au système de synthèse d'images Illumines, développé à l'Ecole des Mines de Saint-Etienne [BP-90].

Le programme prend en entrée un fichier texte contenant une description CSG en langage CASTOR [Beig-88], évalue l'arbre CSG et produit une représentation par frontière sous divers formats, dont un format *rationnel.*

Plusieurs tests ont été effectués afin d'étudier l'influence de divers paramètres sur les performances globales de l'algorithme. C'est ainsi que plusieurs optimisations intéressantes ont pu être apportées à l'algorithme et à la librairie paresseuse elle-même.

Nous présentons ci-dessous quelques résultats obtenus sur des scènes de diverses complexités. L'interprétation des résultats n'est pas toujours évidente, du fait de l'interaction de plusieurs paramètres contradictoires.

VII.1. Performances comparées des diverses versions de l'algorithme

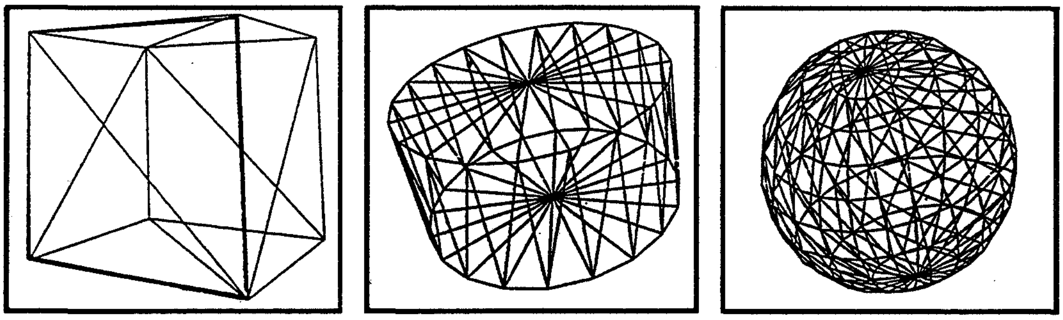
Les tables **T1** et **T2**, ci-dessous, synthétisent quelques résultats extrêmes qui permettent d'estimer les coûts et les gains comparés des trois versions de l'algorithme.
Les paramètres de comparaison retenus sont les suivants:

1) la précision de *recalage* des coordonnées flottantes initiales ;

2) le nombre total de facettes triangulaires appartenant à toutes les primitives de l'arbre CSG (Fig. 3) ;

3) le nombre d'opérations arithmétiques (tous type confondus) qui ont dû être effectuées en arithmétique rationnelle ;

4) le temps de calcul global (incluant la saisie et la sortie des données), exprimés en unités CPU (à titre indicatif, une unité équivaut à environ un soixantième de seconde).



(a) (b) (c)

Figure 3. Primitives utilisées: **a)** le cube (12 facettes), **b)** le cylindre (80 facettes) et **c)** la sphère (360 facettes).

Pour l'alignement des coordonnées flottantes initiales (sommets de primitives) sur des nombres rationnels, nous avons utilisé la méthode de développement en fractions continues (*DFC*, Sect. II.4.3), dans laquelle le seuil de précision ε est laissé en paramètre. Les tests montrent que $\varepsilon = 10^{-6}$ constitue une précision raisonnable: de plus petites précisions risquent d'être insuffisantes vu la précision des flottants initiaux et de plus grandes précisions accroissent inutilement la taille des nombres rationnels, ainsi que les temps d'exécution des opérations rationnelles. En fait, le paramètre ε est important surtout dans la version rationnelle pure de l'algorithme ; il influe beaucoup moins sur la version paresseuse.

La table T1 montre que pour $\varepsilon = 10^{-6}$, le rapport en temps *Paresseux/Rationnel* varie entre 20 et 150 (la première colonne renvoie aux images se trouvant en fin de chapitre).

DFC ε =E-6	Nombre de facettes	Version rationnelle		Version paresseuse		Rapports $\frac{\text{Rationnel}}{\text{Paresseux}}$	
		Opér.	Temps	Opér.	Temps	Opér.	Temps
Fig. 5	96	72247	67145	629	1515	115	44
Fig. 12	252	549635	708435	1156	4270	475	150
Fig. 22	10300	6401412	1669743	90376	83441	71	20

Table T1.

La table T2 montre une autre série de tests, effectués à $\varepsilon = 10^{-9}$. Elle inclut les temps obtenus par la version flottante de l'algorithme, ainsi que les tailles maximales atteintes par les grands entiers. Certains des exemples ont causé l'arrêt anormal du programme en version flottante: sans doute en raison de cas particuliers pour lesquels nous n'avons prévu de solution *avertie*. Toujours est-il qu'après perturbation des données (par une rotation aléatoire), ces mêmes exemples ont fini par aboutir en arithmétique flottante.

DFC ε=E-9	Nombre de Facettes	Temps (unités CPU)			Nb. chiffres (max)		Rapports (temps)		% Opérations P/R
		Flot.	Rat.	Par.	Rat.	Par.	P/F	R/P	
Fig. 5	96	88	97630	1263	76	3	14	77	0
Fig. 14	192	217	223202	2619	78	3	12	85	0
Fig. 12	252	399	2173277	9518	98	3	24	228	0
Fig. 11	600	862	3667275	20966	98	3	24	175	0
Fig. 10	372	147	21247	3806	26	13	26	69	12.79
Fig. 18	556	719	2150076	16336	96	44	23	132	0.15
Fig. 9	160	44	242592	1119	72	10	25	217	1.80
Fig. 17	372	152	25805	4003	32	15	26	6	12.51
Fig. 16	172	49	3563	1157	15	9	24	3	16.21

Table 2.

Les temps de calcul en version rationnelle atteignent quelques heures et les grands entiers atteignent des valeurs considérables (jusqu'à 98 chiffres en base 32768): ces conditions ne peuvent qu'augmenter le rapport *Rationnel/Paresseux*. En effet, la version paresseuse de l'algorithme s'avère jusqu'à 228 fois plus rapide que la version rationnelle pure et seulement une vingtaine de fois plus lente que la version flottante (lorsqu'elle n'avorte pas).

Ce rapport s'explique par la très faible proportion *Paresseux/Rationnel* en termes d'opérations exactes effectuées (moins de 20 %). Autrement dit, les *intervalles* des nombres paresseux suffisent la plupart du temps à la comparaison des nombres. Dans de nombreux cas, l'algorithme paresseux se termine sans qu'aucune opération rationnelle ne soit effectuée.

VII.2. Sensibilité de l'algorithme

Nous avons également expérimenté le test décrit dans [LTH-86] et devenu un classique du genre.

Soient A un cube unité centré à l'origine et $A(\alpha)$ un autre cube obtenu à partir de A après trois rotations d'angle α autour de chaque axe de coordonnées, successivement. Ensuite, l'algorithme est exécuté pour calculer le solide $A \cap A(\alpha)$ pour différentes valeurs de α.

Les modeleurs polyédriques qui tiennent compte des faces coplanaires produisent, normalement, des résultats cohérents aussi bien pour des grandes valeurs de α que pour $\alpha = 0$. Cependant, comme signalé dans [LTH-86], de nombreux modeleurs échouent pour $\alpha \leq 2$ Degrés.

Plus précisément, au dessus d'un certain angle α_2 ils donnent toujours un résultat correct ; en dessous d'un angle α_1 ($\alpha_1 < \alpha_2$), A et $A(\alpha)$ sont indiscernables ; enfin, entre α_1 et α_2 les modeleurs ont des résultats imprévisibles (voir [HHK-89], [SS-90] et [GCP-91]): ils avortent par suite d'une erreur fatale ou produisent des résultats incohérents.

La version *paresseuse* de notre algorithme arrive toujours à son terme: il n'y a pas d'angle α_2, et la valeur de α_1 peut être rendue aussi petite que l'on veut, voire même nulle si on adopte la méthode de conversion exacte *Flottant/Rationnel* (Sect. II.4.3). Par contre, la limitation due à la précision finie des flottants initiaux subsiste: bien que les nombres flottants puissent être convertis en des rationnels sans perte de précision, ces flottants ne sont que des valeurs approchées de données réelles. Il est donc inutile (et coûteux) de vouloir aligner ces flottants exactement sur des rationnels.

En pratique, pour avoir $\alpha_1 = 10^{-4}$, il suffit d'utiliser un développement en fractions continues, avec $\varepsilon = 10^{-6}$.

VIII. CONCLUSION

La réalisation pratique d'un algorithme de résolution d'opérations booléennes nous a permis d'aborder certains problèmes ouverts de modélisation solide. En particulier, cela nous permis de mieux comprendre les effets néfastes de l'imprécision numérique sur la cohérence des représentations par frontière. Les diverses solutions proposées dans la littérature sont souvent partielles et contraignantes. L'utilisation du concept d'arithmétique paresseuse résout le problème des imprécisions numériques en les "reléguant" à un niveau inférieur, relevant de la librairie paresseuse.

La version courante de l'algorithme est encore susceptible de plusieurs améliorations. Certains choix de programmation (par exemple, listes versus tableaux, gestion mémoire rudimentaire) s'avèrent maladroits. Nous pensons que leur correction profiterait globalement à l'algorithme, sans remettre en cause sa philosophie.

D'autre part, il reste encore des modules de l'algorithme qui gagneraient à être repensés en termes de programmation avertie.

Enfin, il faut dire que l'arithmétique paresseuse n'était pas prévue au départ, de sorte que nous avons dû procéder par "retouches" successives des programmes. C'est d'ailleurs de cette manière que nous avons réalisé que l'arithmétique paresseuse n'était pas totalement transparente vis à vis des programmes d'application: plusieurs variantes allant dans le sens de cette transparence restent à intégrer dans la librairie paresseuse [Jaill-93].

IX. QUELQUES EXEMPLES

Nous donnons ci-dessous une description sommaire des quelques exemples référencés dans les tables T1 et T2 de la page 106.

Fig. 1-3 Union, intersection et différence de deux cubes dont l'un est déduit de l'autre par trois rotations d'angle $\alpha = 3$ degrés autour de chaque axe de coordonnées, successivement.

Fig. 4-6 Union, intersection et différence de deux groupes de quatre cubes, l'un des deux groupes ayant subi une rotation.

Fig. 7-9 Union, intersection et différence de deux cylindres dont les axes sont orthogonaux mais non concourants.

Fig. 10 Différence entre un cube et une sphère centrés à l'origine. Le diamètre de la sphère est légèrement supérieur au coté du cube. Voir aussi Fig. 17.

Fig. 11 Différence entre une sphère et une sphère et trois cylindres d'axes orthogonaux.

Fig. 12 Différence entre un cube et trois cylindres d'axes orthogonaux.

Fig. 13-15 Union, intersection et différence de deux groupes de huit cubes, l'un des deux groupes ayant subi une rotation.

Fig. 16 Différence entre un cube et deux cylindres orthogonaux et de rayons distincts.

Fig. 17 Intersection entre un cube et une sphère centrés à l'origine. Le diamètre de la sphère est légèrement supérieur au coté du cube. Voir aussi Fig. 10.

Fig. 18 Un cendrier modélisé à l'aide d'une sphère, de deux cylindres et de quatre cubes.

Fig. 19 Intersection de cinq cubes pris dans une certaine orientation.

Fig. 20 Différence de cinq cubes pris dans une certaine orientation.

Fig. 21 Eponge de Sierpinski.

Fig. 22 Forme architecturale à 3 x 2 travées.

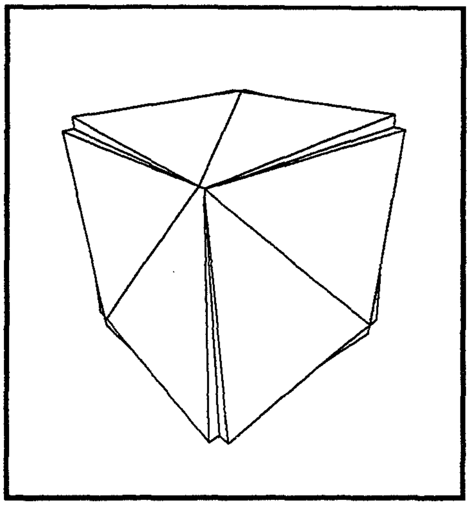


Figure 1.

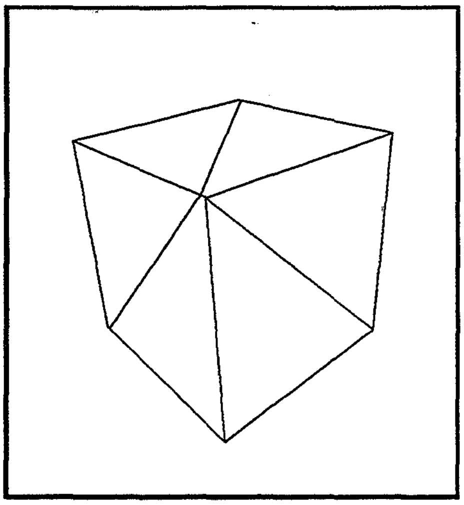


Figure 2.

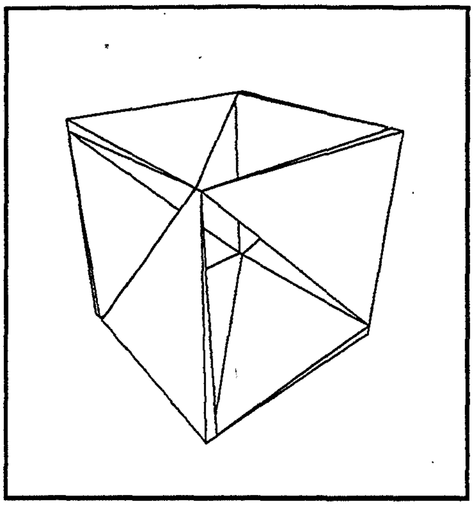


Figure 3.

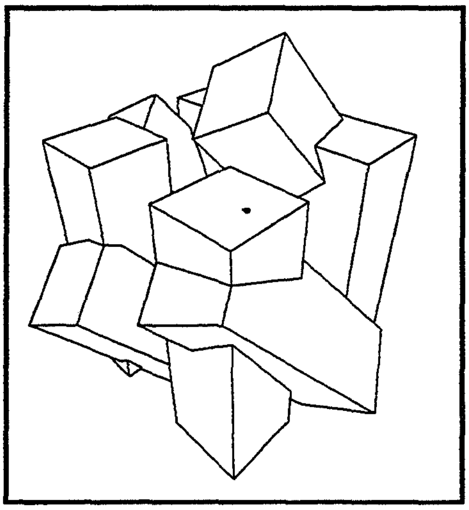


Figure 4.

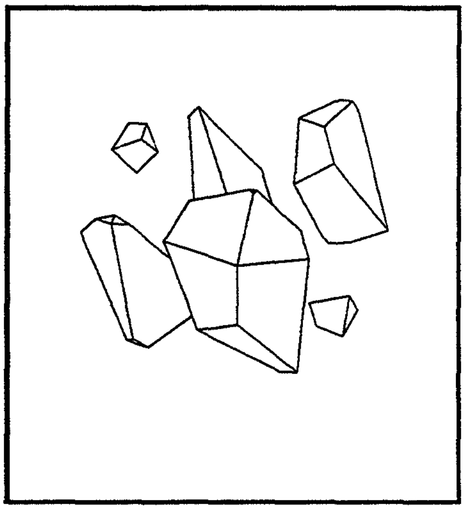


Figure 5.

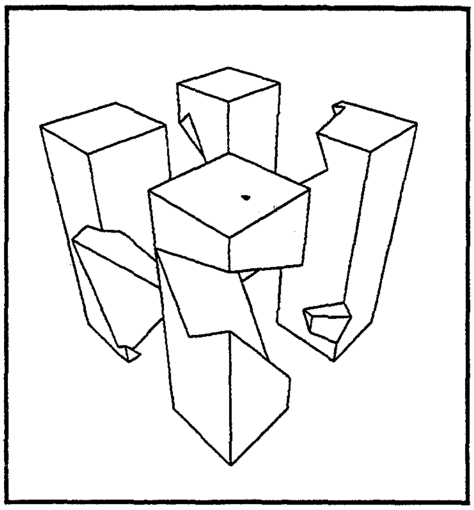


Figure 6.

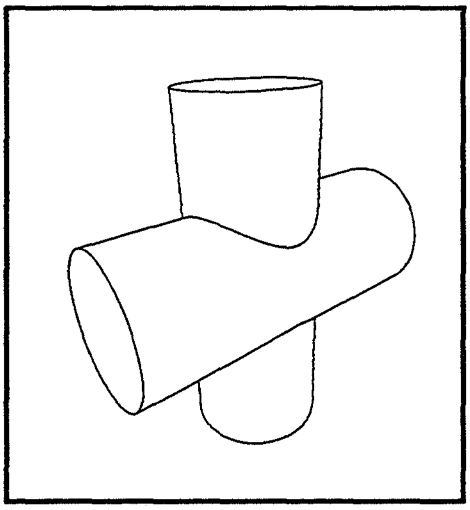


Figure 7.

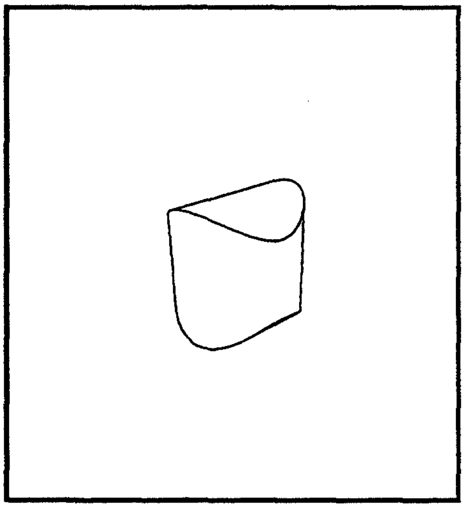


Figure 8.

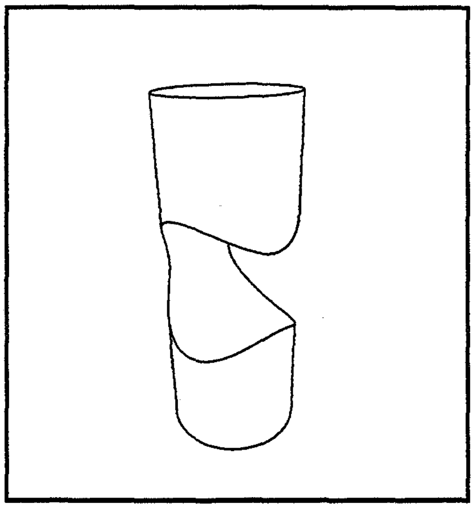


Figure 9.

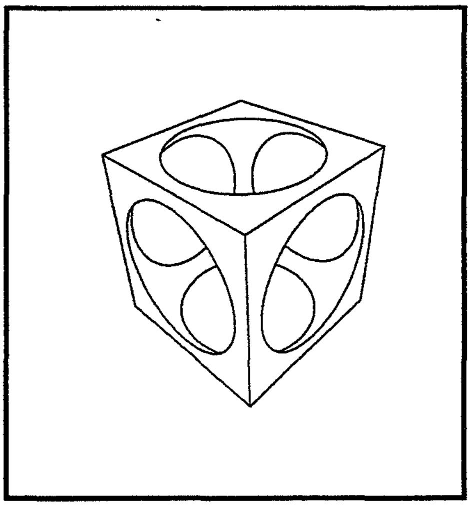


Figure 10.

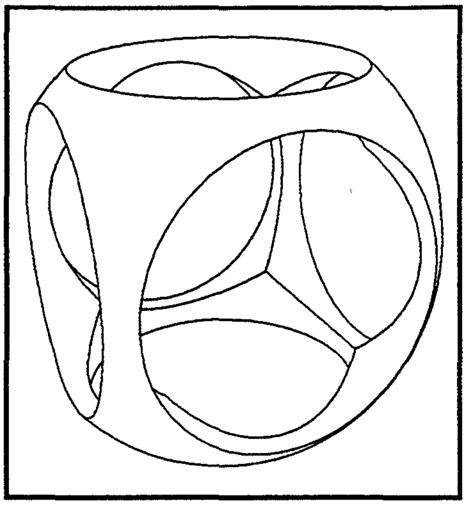


Figure 11.

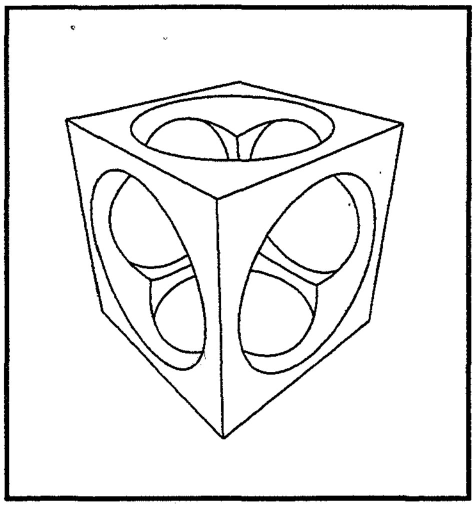


Figure 12.

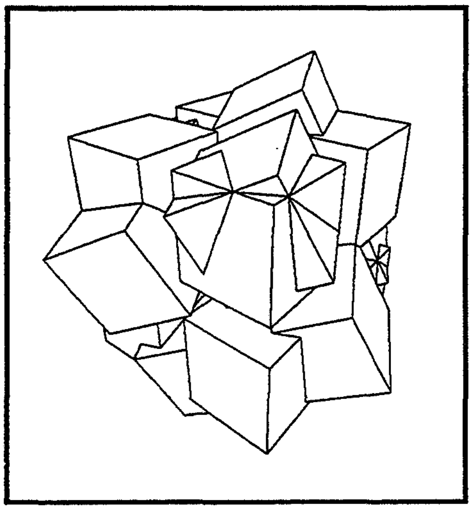


Figure 13.

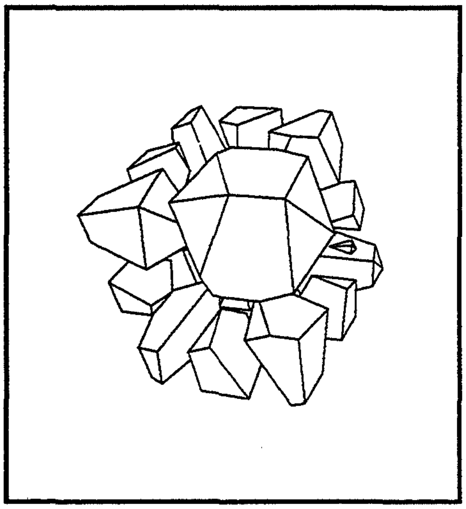


Figure 14.

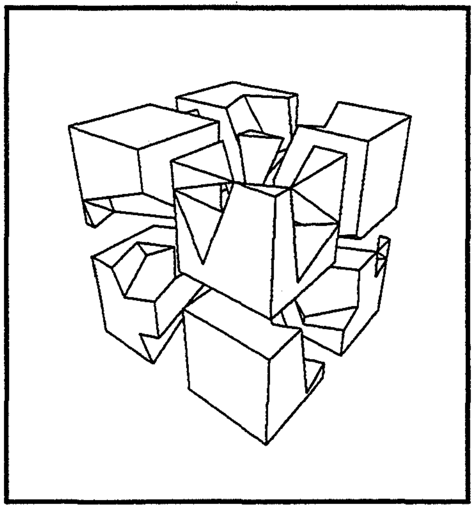


Figure 15.

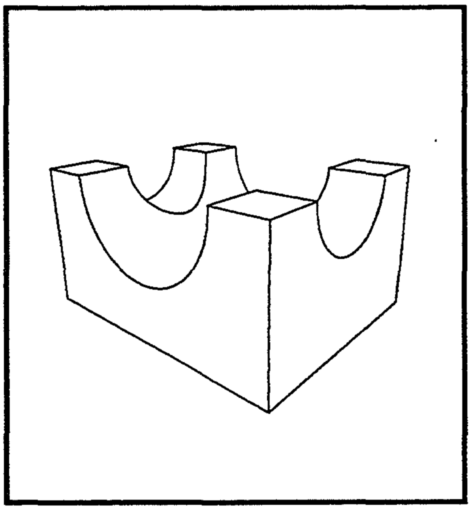


Figure 16.

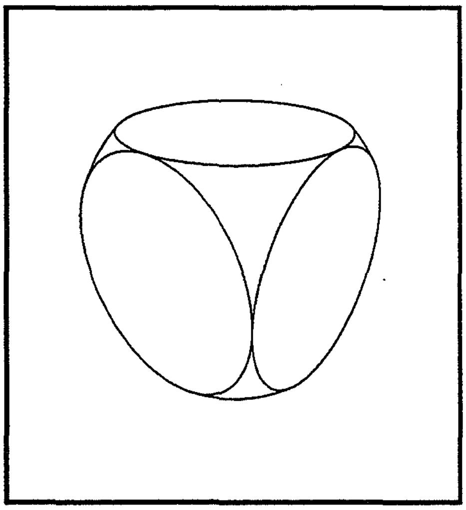


Figure 17.

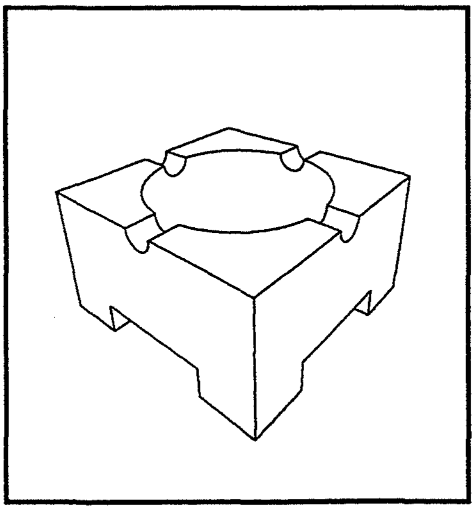


Figure 18.

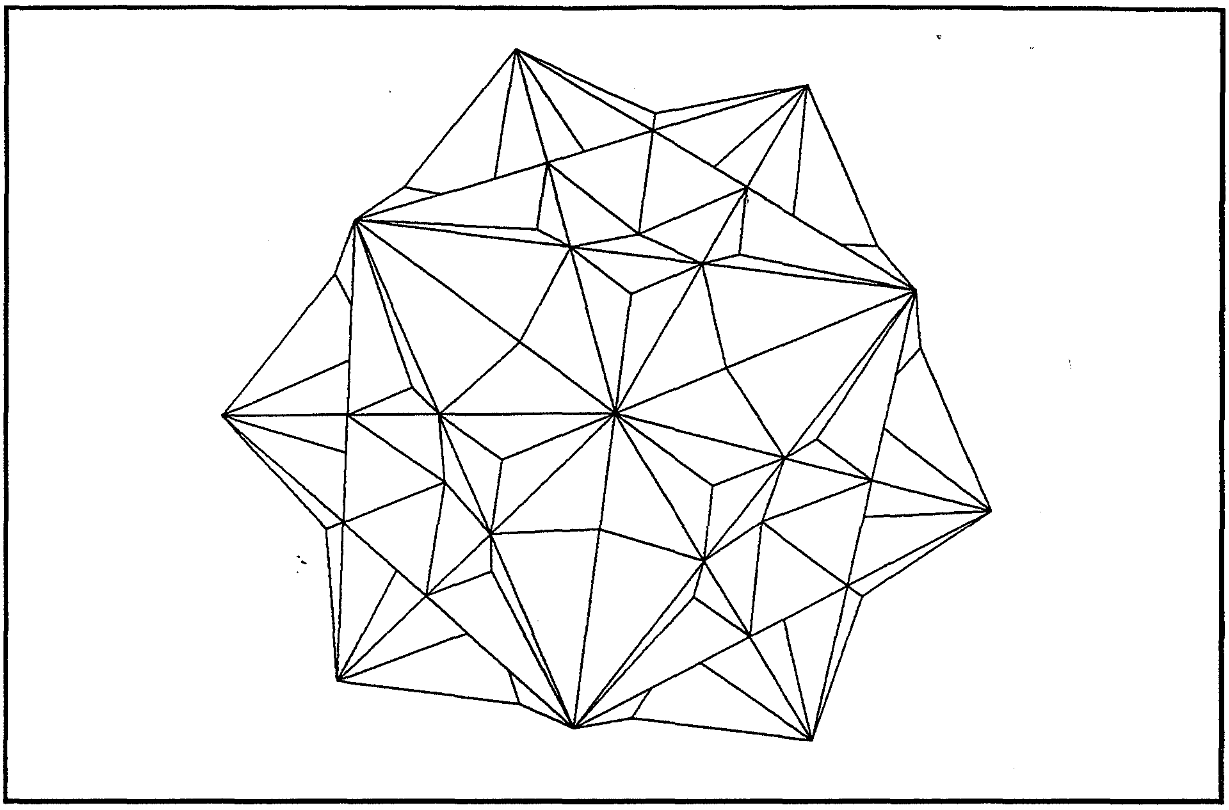


Figure 19.

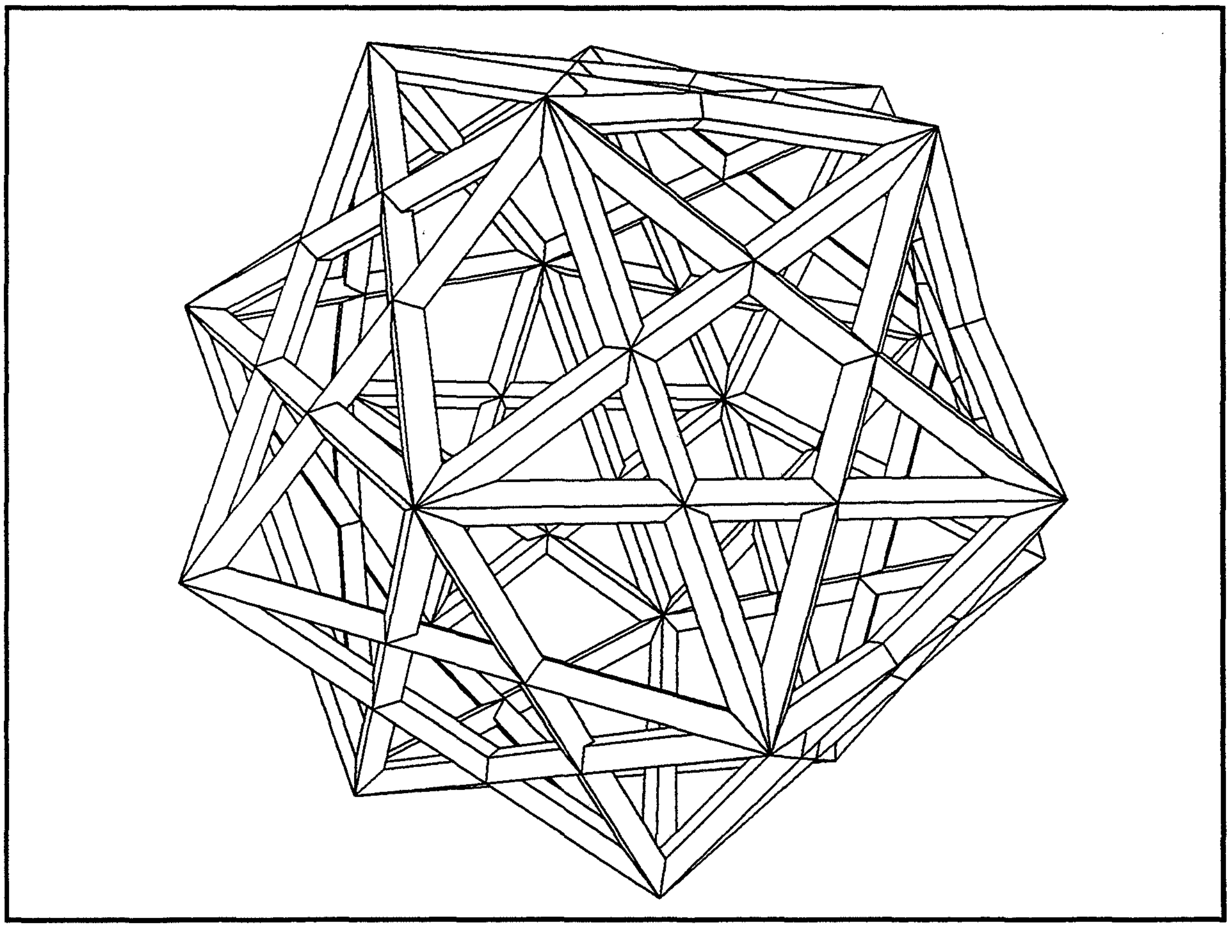


Figure 20.

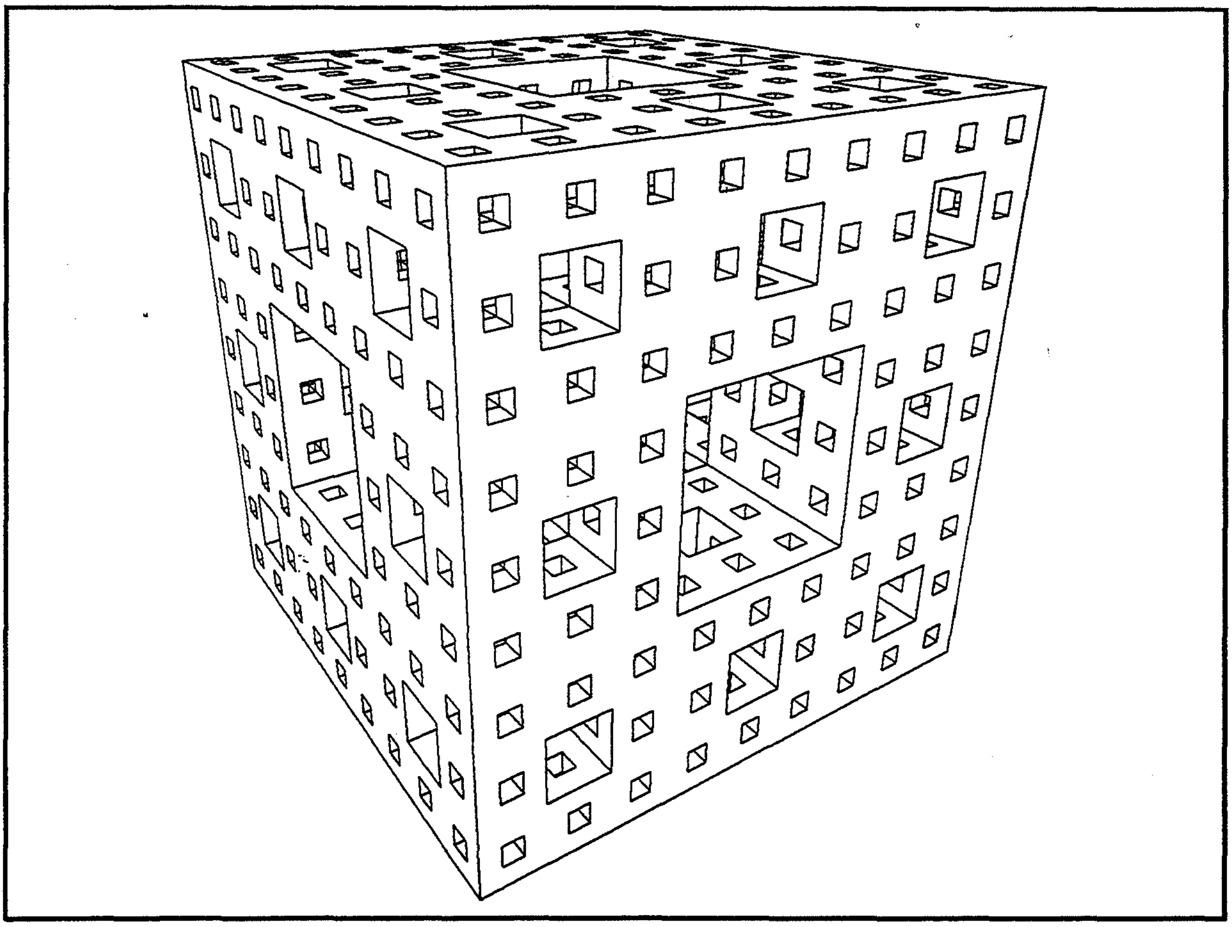


Figure 21.

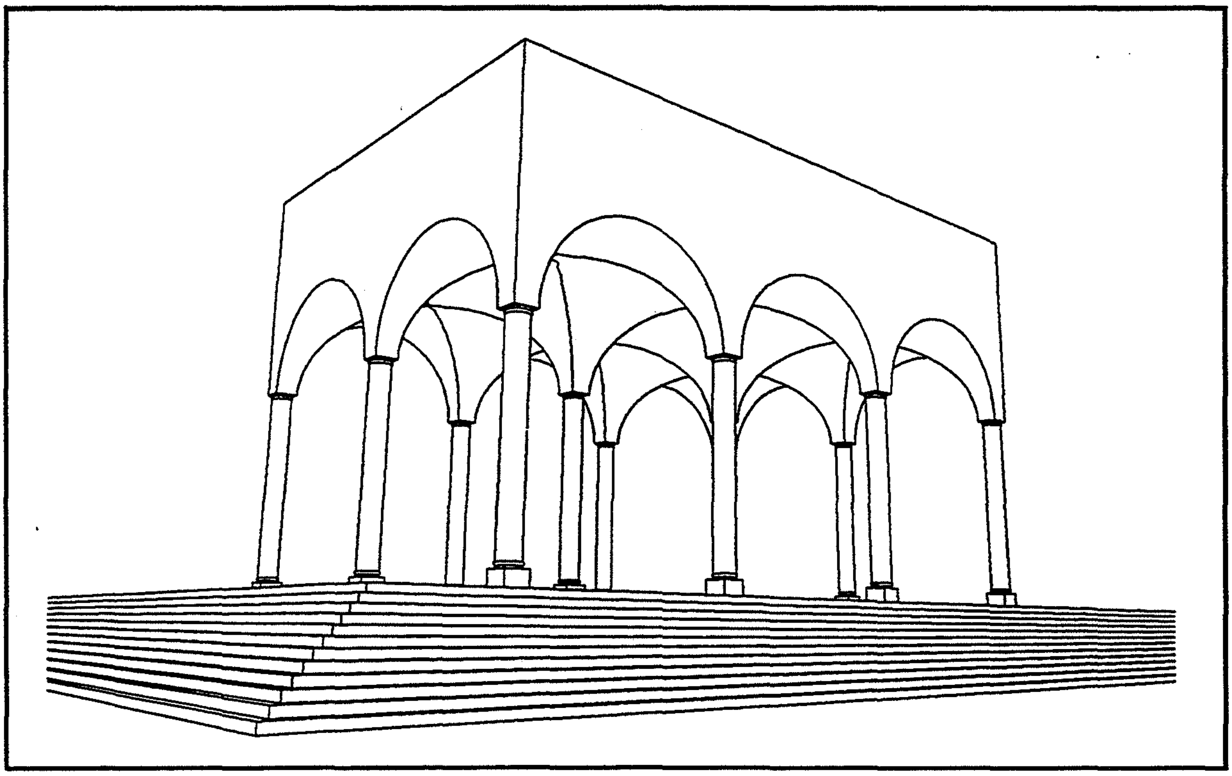


Figure 22.

Chapitre IV

Une optimisation simple des algorithmes d'évaluation d'arbres CSG

I. INTRODUCTION

La frontière d'un objet solide décrit par un arbre CSG est incluse dans l'union des frontières de toutes les primitives. Aussi, les algorithmes d'*évaluation de frontière* [RV-85] utilisent fondamentalement le paradigme "générer et tester" afin de déterminer les sous-ensembles des frontières des primitives qui constituent la frontière ∂S du solide final S.

Une conséquence de ce paradigme est que les algorithmes effectuent certains calculs géométriques (intersections et classifications [Tilo-80]) qui s'avèrent "inutiles" pour la construction de ∂S. Le recours à des techniques d'optimisation simples telles que le test de boîtes englobantes ([LTH-86], [SS-88]) ne permet pas d'éviter systématiquement tous ces calculs inutiles.

En Section II, nous rappelons brièvement le principe général des deux grandes familles d'algorithmes, puis nous décrivons le mécanisme qui conduit aux calculs géométriques inutiles.

En Section III et IV, nous rappelons successivement le principe de deux techniques d'optimisation existantes: la première, due à Cameron [Came-91], est basée sur une classe particulière de boîtes englobantes et la deuxième, due à Rossignac et Voelcker [RV-89], exploite les propriétés algébriques des expressions booléennes.

En Section V, nous présentons une nouvelle méthode qui combine judicieusement ces deux techniques. En Section VI, nous proposon une mise œuvre possible de cette methode.

Nous concluons en VII.

II. EVALUATION D'ARBRES CSG

Traditionnellement, on distingue deux types de méthodes (ou de schémas) d'évaluation d'arbres CSG: la méthode dite "incrémentale" et la méthode dite "non incrémentale".

Cette dénomination, consacrée par l'usage, est passée dans le jargon de la modélisation solide ([RV-85], [Ross-91], [Mill-93]), bien qu'elle prête quelquefois à confusion (à notre sens, les termes de "récursive" et "directe" sont plus adaptés). Pour éviter toute ambiguïté, nous présentons ci-dessous le principe de chacune des deux méthodes.

Sans perte de généralité, nous considérerons des arbres CSG dont les nœuds non terminaux désignent uniquement des opérations booléennes régularisées ($\cap$, $\cup$, $-$): les transformations unaires habituelles sont supposées être appliquées directement aux primitives.

II.1. Méthode incrémentale

Dans la méthode incrémentale, l'évaluation commence aux feuilles de l'arbre CSG et se propage jusqu'à la racine. L'opération booléenne portée par chaque nœud interne est appliquée aux représentations par frontière des deux sous-arbres fils, évaluées récursivement (les primitives sont en général des objets simples dont la frontière est facile à expliciter). Cette méthode se ramène donc à la résolution d'une opération booléenne à la fois, entre deux solides de frontières connues (ou "Boundary Merging" [RV-85]).

On a le schéma d'évaluation suivant :

```
EvaluerArbre (S: CSG): Frontière
début
  si (S est une primitive) alors
      rendre ConstruireFrontièreDePrimitive (S)
  sinon
      G := EvaluerAbre (S->filsGauche) ;
      D := EvaluerAbre (S->filsDroit) ;
      rendre RésoudreOpération (G, S->opération, D)
  fsi
fin.
```

II.2. Méthode non incrémentale

Dans la méthode non incrémentale, la frontière d'un objet CSG *S* est calculée en générant, un à la fois, les éléments *candidats* (sous-ensembles des frontières des primitives), puis en *classifiant* ces éléments par rapport à l'arbre *S* tout entier.

Pour classifier un élément candidat X qui est un sous ensemble de la frontière ∂A d'une primitive A, l'approche standard consiste à décomposer X en trois sous-ensembles: *X-dans-S*, *X-sur-S* et *X-hors-S* qui se trouvent à l'intérieur, sur la frontière, ou à l'extérieur de S, respectivement [Tilo-80].

L'union de tous les sous-ensembles *X-sur-S*, pour tous les éléments X de toutes les primitives de S est égale à la frontière de S.

On a le schéma d'évaluation suivant :

EvaluerArbre (S: CSG) : Frontière
<u>début</u>
    C := Ensemble des éléments candidats ;
    ∂S := $\varnothing$;
    <u>pour</u> chaque $X \in C$ <u>faire</u>
        Classifier (X, S) ;
        ajouter (*X-sur-S*, ∂S)
    <u>fait</u> ;
    <u>rendre</u> ∂S
<u>fin</u>.

II.3. Le problème des calculs géométriques inutiles

Dans la méthode incrémentale, les calculs inutiles sont dus au fait que les objets intermédiaires sont créés explicitement, même si certaines portions des frontières de ces objets n'appartiennent pas à la frontière de l'objet final. Il arrive même que l'évaluation totale de l'arbre CSG conduise à l'objet *vide* lui-même [Tilo-84].

Dans la méthode non incrémentale, les calculs inutiles proviennent du fait que la frontière de S est obtenue à partir des sous-ensembles *X-sur-S* des éléments candidats X (faces, arêtes ou sommets) appartenant aux frontières ∂A des diverse primitives A .

Cepedant, la construction de ∂S n'a pas toujours besoin de *X-sur-S* tout entier, dont certaines parties pourraient être "supprimées" de ∂A (en ajoutant ou en retranchant du matériau à A), sans changer la forme de S. Par exemple, le point p de la Figure 1, bien que situé géométriquement sur la frontière de l'objet $S = (A \cap B) \cup C$, peut être supprimé de ∂A sans affecter ∂S, par exemple en retranchant à A une petite boule fermée centrée en p et de rayon suffisamment petit.

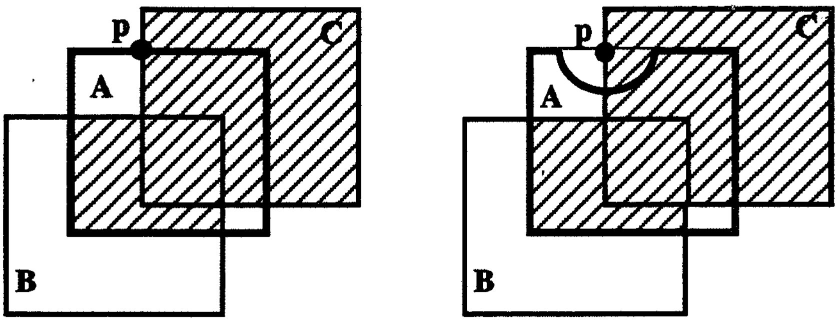


Figure 1. Le point p se trouve sur la frontière de $S = (A \cap B) \cup C$ (à gauche), mais p peut être supprimé de ∂S en retranchant à A une petite boule centrée centrée en p, sans affecter ∂S (à droite).

En général, les algorithmes incrémentaux effectuent moins de calculs inutiles que les algorithmes non incrémentaux, du fait que les premiers exploitent une certaine *cohérence* d'objets. En revanche, la méthode incrémentale nécessite le stockage d'objets intermédiaires, dont la taille peut être très importante (même si le résultat final est un objet vide).

III. BOITES DE CAMERON

III.1. Principe

Cameron [Came-91] associe *explicitement* à chaque nœud A d'un arbre CSG S une *boîte* simple $\beta(A)$, hors de laquelle on peut modifier l'objet A sans affecter l'objet S. Ces boîtes sont obtenues par affinements successifs des boîtes englobantes associées aux primitives. Les boîtes finales sont plus générales que les boîtes englobantes usuelles, en ce sens qu'elles n'englobent pas nécessairement les objets correspondants, mais elles jouent plutôt le rôle de *fenêtres* qui ne laissent voir des objets que les parties qui ont toutes les chances de contribuer au solide final S (Fig. 2).

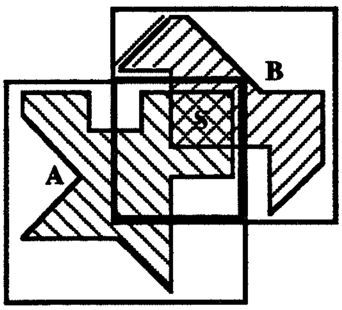


Figure 2. Boîtes de Cameron: soient A et B deux objets bornés (les deux polygones hachurés), pour lesquels on connaît deux boîtes englobantes [A] et [B]. Manifestement, seul le sous-ensemble de ∂A inclus dans la boîte $\beta(A) = [A] \cap [B]$ (en gras) mérite d'être considéré dans le calcul de la frontière ∂S de l'objet $S = A \cap B$.

III.2. Boîtes isothétiques

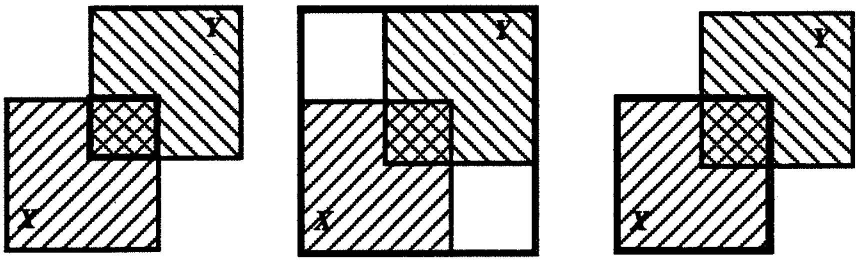
Diverses classes de "boîtes" sont possibles [CY-92], la plus simple et la plus commode étant la classe des parallélépipèdes *isothétiques* (dont les faces sont alignées sur les plans de coordonnées). On distinguera deux boîtes particulières: la boîte *vide* ([]) et la boîte *infinie* (Ω).

Dans cette classe de boîtes, on définit trois opérations de composition internes $\boxed{\cap}$, $\boxed{\cup}$ et $\boxed{\%}$, associées aux opérations booléennes $\cap$, $\cup$ et $-$, respectivement (Fig. 3). Ces opérations sont définies comme suit :

$$X \boxed{\cap} Y = X \cap^* Y\,;$$

$$X \boxed{-} Y = X\,;$$

$$X \boxed{\cup} Y = \textit{plus petite boîte isothétique englobant } (X \cup Y).$$



$X \boxed{\cap} Y$ $\quad$ $X \boxed{\cup} Y$ $\quad$ $X \boxed{-} Y$

Figure 3. Opérations de composition de boîtes isothétiques.
La boîte résultat est montrée en gras.

Notons que pour chaque opération %, on a toujours $(X \% Y) \subseteq (X \boxed{\%} Y)$.

III.3. Affinement des boîtes

Pour chaque nœud A d'un arbre CSG, il s'agit de déterminer la plus "petite" boîte isothétique $\beta(A)$ qui a la propriété de la *fenêtre* pour l'objet représenté par A. Initialement, chaque nœud interne est affecté de la boîte "infinie" Ω et chaque primitive est affectée d'une boîte englobante isothétique. Ensuite, les boîtes sont *affinées* en leur appliquant les deux types de règles définies ci-dessous.

Règle de montée :

En chaque nœud binaire $T = G \% D$, remplacer la boîte $\beta(T)$ par :
$\beta(T) \boxed{\cap} (\beta(G) \boxed{\%} \beta(D))$.

Règle de descente :

En chaque nœud T, de père T_p, remplacer la boîte $\beta(T)$ par:
$\beta(T) \boxed{\cap} \beta(T_p)$.

On vérifie aisément que les règles ci-dessus préservent la propriété de la *fenêtre* pour les objets correspondants. D'autre part, les nouvelles boîtes sont plus "petites" (au sens de l'inclusion) ou de même taille que les boîtes opérandes.

III.4. Convergence des boîtes

Etant données les boîtes initiales, les meilleurs boîtes possibles s'obtiennent comme le *point fixe* d'un processus qui applique itérativement les règles ci-dessus, jusqu'à *stabilisation* de toutes les boîtes. L'ordre naturel d'application des règles correspond à des traversées successives de l'arbre CSG, d'abord des feuilles vers la racine, en appliquant les règles de montée, puis de la racine vers les feuilles, en appliquant les règles de descente.

Le théorème ci-dessous garantit la *convergence* du processus et fournit également un critère simple pour la détection de l'*état stable*.

Théorème :

Si après une phase de montée suivie d'une phase de descente, on n'a pas créé de nouvelle boîte vide, alors le système de boîtes de l'arbre est dans son état stable. Pour un arbre comportant n primitives, l'état stable est atteint après au plus (n-1) paires de phases (montée, descente).

La démonstration de ce théorème a été donnée dans [CY-92] pour des arbres CSG qui ne comportent que des *intersections* ou des *unions*. Nous présentons en Annexe une démonstration qui tient compte également de l'opération *différence*.

IV. LES ZONES ACTIVES DE ROSSIGNAC ET VOELCKER

IV.1. Principe

Rossignac et Voelcker [RV-85] généralisent l'idée de Cameron en associant à chaque nœud A de l'arbre S une région Z de l'espace hors de laquelle on peut modifier A sans affecter S et dans laquelle toute modification de A affecte S. Cette région Z est dite *zone active* de A dans S (Fig. 4). On y reviendra en Section IV.4.

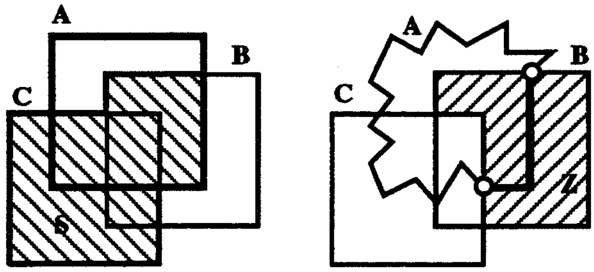


Figure 4. Zones actives: pour S = (A ∩ B) ∪ C (à gauche), toute modification de A dans Z = B-C (à droite) affecte S et aucune modification de A hors de Z n'affecte S.

En d'autres termes, Z est telle que la "compression" de A en $A \cap Z$ ou sa "dilatation" en $A \cup \bar{Z}$ ne modifient pas la forme de S. La notion de zone active relève de l'étude des propriétés *algébriques* des arbres CSG en tant qu'expressions booléennes.

IV.2. Définitions

Forme positive

Tout arbre CSG peut être transformé en un arbre équivalent dans lequel les nœuds internes ne portent que des opérateurs ∩ ou ∪, mais dont les primitives peuvent désigner éventuellement des objets non bornés (définis comme les complémentaires d'objets bornés). Un tel arbre constitue la *forme positive* de l'arbre initial, il s'obtient par une simple application des lois de De Morgan. Sauf indication contraire, les arbres CSG considérés ici sont supposés être sous leur *forme positive.*

Nœuds de branchement

Etant donné un nœud A (terminal ou non) d'un arbre S, on appelle *nœuds de branchement* de A dans S l'ensemble des nœuds de S dont les "pères" se trouvent sur le *chemin* reliant S à A, à l'exception du nœud A lui-même. Un nœud de branchement B est dit *I-nœud* (resp. *U-nœud*) si le nœud père de B porte un opérateur ∩ (resp. ∪).

Zones actives

On appelle *I-zone* de A dans S la région $I_S(A)$ de l'espace définie par l'intersection de tous les *I-nœuds* de A dans S. De même, la *U-zone* de A dans S est la région $U_S(A)$ définie par l'union de tous les *U-nœuds* de A dans S. Enfin, on appelle *zone active* de A dans S la région $Z_S(A)$ définie par $Z_S(A) = I_S(A) - U_S(A)$. En particulier, si $S = A \cap B$ alors $Z_S(A) = B$ et si $S = A \cup B$ alors $Z_S(A) = \bar{B}$.

Notation

Afin d'alléger la notation, nous désignerons toujours par A un nœud générique (terminal ou non) d'un arbre CSG de racine S et nous noterons simplement I, U et Z en lieu et place de $I_S(A)$, $U_S(A)$ et $Z_S(A)$, respectivement.

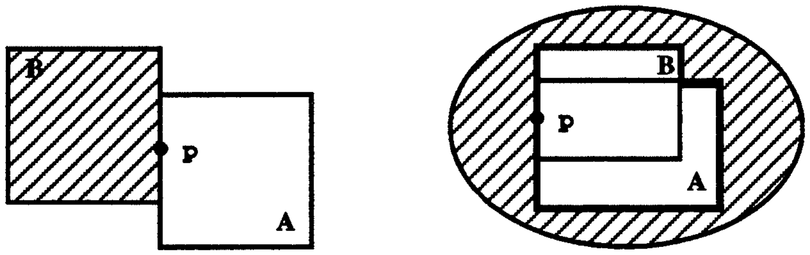
IV.3. Propriété fondamentale des zones actives

Rossignac et Voelcker [RV-89] démontrent (Propositions 10 et 11, page 62) que pour tout nœud A d'un arbre CSG S, l'objet représenté par S s'exprime comme l'union de deux ensembles disjoints ($A \cap Z$) et $S_\varnothing$, où Z est la *zone active* de A dans S et $S_\varnothing$ est un ensemble indépendant de l'objet A ($S_\varnothing$ est l'objet qui résulterait de l'évaluation de l'arbre S si on y remplaçait A par $\varnothing$).

La décomposition ci-dessus montre que toute modification de A *dans* sa zone active Z affecte S et aucune modification de A *hors* de Z n'affecte S. Par conséquent, l'ensemble $A \cap Z$ constitue la *contribution exacte* de A à S. On comprend maintenant l'exemple de la Fig. 4, où $S = (A \cap B) \cup C$ et $Z = B - C$.

Ambiguïté de frontière

En vertu du résultat précédent, tout point p de ∂A intérieur à Z est important pour ∂S (sa suppression de ∂A affecterait ∂S) et tout point p de ∂A intérieur à $\bar{Z}$ est inutile à ∂S (sa suppression de ∂A n'affecterait pas ∂S). Mais qu'en est-il des points p de ∂A qui se trouvent sur la frontière de Z ? Dans ce cas particulier, on ne peut pas conclure comme le montre le contre-exemple de la Figure 5 : dans le cas (a), où $S = A \cap B$, le point p est inutile à ∂S ($= \varnothing$) ; dans le cas (b), où $S = A \cup B$, le point p est indispensable à ∂S car on ne peut le supprimer de ∂A (et de ∂B, par symétrie) sans altérer ∂S.



(a) (b)

Figure 5. a) pour $S = A \cap B$, on a $Z = B$: le point p de ∂A appartient à ∂Z mais il est hors de $A \cap Z$ et $p \notin \partial S$ ($= \varnothing$) ; **b)** pour $S = A \cup B$, on a $Z = \bar{B}$: le point p de ∂A appartient à ∂Z, mais il est hors de $A \cap Z$; cependant, p est important pour ∂S car si on le supprime de ∂A (et de ∂B, par symétrie) ∂S serait incomplet.

L'ambiguïté de frontière ci-dessus est bien connue en modélisation géométrique. Typiquement, elle se résout par une étude de voisinage ([Tilo-80], [RV-85]). Rossignac et Voelcker [RV-89] proposent une méthode qui utilise la *I-zone* et la *U-zone* de A dans S.

Soit ϑ une boule fermée, centrée en p et de rayon suffisamment petit. En définissant les voisinages $\vartheta_+ = \vartheta \cap A$ et $\vartheta_- = \vartheta \cap \bar{A}$, Rossignac et Voelcker démontrent le théorème suivant (Proposition 19, [RV-89], page 67) :

Théorème :

L'ensemble de tous les points p appartenant à l'union des frontières des primitives d'un objet CSG S, qui vérifient la condition ($\vartheta_+ \cap I \neq \varnothing$) *et* ($\vartheta_- \cap \overline{U} \neq \varnothing$) est égal à la frontière de S.

L'intérêt pratique de ce résultat pour les algorithmes d'évaluation d'arbres CSG (qui rappelons le, sont fondés sur le principe "générer et tester") tient au fait qu'il permet de construire la frontière d'un objet CSG S en considérant seulement les *contributions effectives* des frontières des primitives.

En partant de ce résultat, Rossignac et Voelcker [RV-89] proposent une méthode d'évaluation de frontière qui constitue un compromis entre les deux méthodes classiques: la méthode incrémentale et la méthode non incrémentale. Nous rappelons ci-dessous le principe de cette nouvelle méthode.

IV.4. Méthode de Rossignac et Voelcker pour l'évaluation d'arbres CSG

IV.4.1. principe

L'idée est de partir de la méthode *non incrémentale* et de substituer à l'approche standard de *classification* par rapport à S des divers éléments candidats X (appartenant à la frontière d'une primitive A) une approche qui consiste à *découper*[1] X, successivement contre chaque *nœud de branchement* de A dans S, jusqu'à ce que X soit réduit à $\varnothing$ (et donc éliminé de l'évaluation) ou que tous les nœuds de branchement aient été traités.

En d'autres termes, X est découpé contre la zone active Z de A dans S, pour n'en retenir que la *contribution effective* de X à ∂S. La zone active s'exprime

[1] Le découpage d'un élément X contre un ensemble F (une fenêtre) est, ici, l'opération qui consiste à ne garder de X que le sous-ensemble qui se trouve dans F. Le terme anglais employé dans [RV-89] est "Trimming": nous n'avons pas su lui trouver une traduction française satisfaisante.

comme l'intersection de certains nœuds de S ou de leurs complémentaires, de sorte qu'en moyenne X se réduit à $\emptyset$ après seulement quelques opérations de découpage. D'autre part, comme S contient tous les nœuds de Z, l'approche par *découpage* ne requiert jamais plus de calculs géométriques que l'approche par *classification*.

IV.4.2. Limites de cette méthode

Cependant, la méthode ci-dessus a quand même ses limites. Ses performances varient entre deux extrêmes: dans le meilleur des cas, un élément candidat X (supposé entièrement inutile à ∂S) est réduit à $\emptyset$ dès la première opération de découpage; dans le pire des cas, X n'est reconnu inutile qu'après l'avoir découpé contre tous les nœuds de branchement de A dans S.

Dans ce deuxième cas, le découpage aura coûté autant de calculs d'intersections (de X avec les frontières de toutes les primitives) que la méthode standard de classification de X par rapport à l'arbre S.

D'une manière générale, le nombre d'opérations de découpage nécessaire à l'élimination de X dépend a priori de l'*ordre* dans lequel sont considérés les nœuds de branchement de A. Le choix d'un ordre *optimal* ne va pas de soi: Rossignac et Voelcker laissent ouvert ce problème. A l'évidence, cet ordre optimal varie d'un élément X à l'autre (Fig. 6).

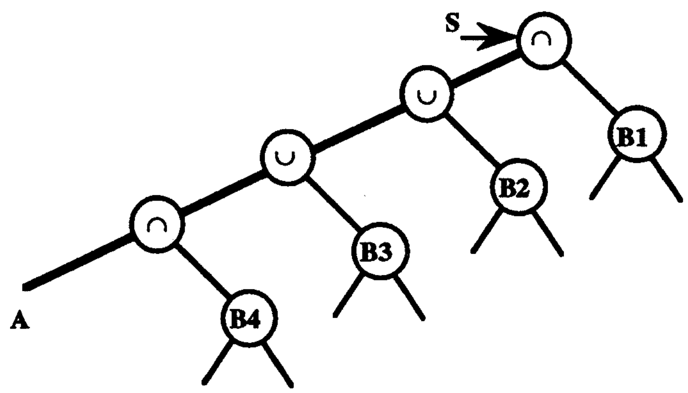


Figure 6. Le nombre d'opérations de découpage nécessaire à l'élimination d'un élément X de ∂A (supposé inutile à ∂S) dépend de l'ordre dans lequel sont considérés les nœuds de branchement (B_i) de A dans S. Dans le meilleur des cas, X est éliminé dès le premier découpage. Dans le pire des cas, tous les noeuds de branchement de A (et donc toutes les primitives de S) sont considérés. En moyenne, X est éliminé après quelques opérations de découpage contre les premiers nœuds de branchement.

A notre sens, la recherche systématique d'un ordre optimal est sans intérêt pratique: elle coûterait au moins autant que l'adoption du plus mauvais ordre de découpage.

V. UNE METHODE COMBINANT ZONES ACTIVES ET BOITES DE CAMERON

Nous proposons maintenant une méthode nouvelle d'évaluation d'arbres CSG, qui combine avantageusement les boîtes de Cameron [Came-91] et les zones actives de Rossignac et Voelcker [RV-89].

V.1. Principe

A l'évidence, pour tout nœud A d'un arbre CSG S, la boîte de Cameron $\beta(A)$ constitue une boîte englobante (au sens usuel) de la contribution *effective* de l'objet A à l'objet S.

Plus précisément, on a : $(A \cap Z(A)) \subset (A \cap \beta(A)) \subset \beta(A)$, comme illustré sur la Figure 7 ci-dessous.

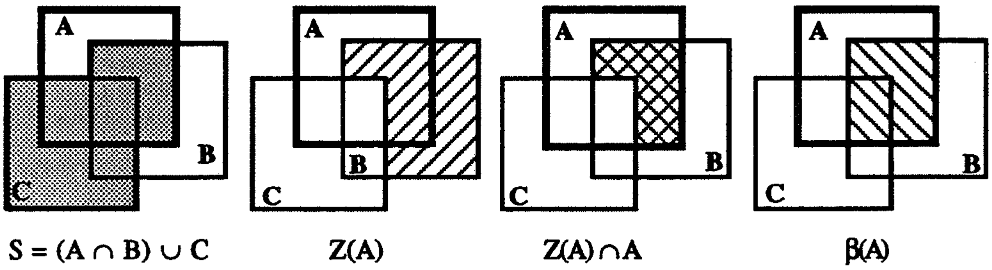


Figure 7. Dans $S = (A \cap B) \cup C$, la primitive A a pour zone active $Z = B-C$.
La boîte $\beta(A)$ englobe la contribution effective de A à S: $(Z(A) \cap A) \subset \beta(A)$.

Du point de vue de l'utilisation, les zones actives requièrent une certaine évaluation de l'arbre CSG, alors que les boîtes de Cameron s'utilisent aussi simplement que les boîtes englobantes usuelles. C'est pourquoi nous avons pensé à *combiner* ces deux notions, d'une manière judicieuse qui évite en pratique de nombreux calculs géométrique inutiles, bien plus que la technique classique des boîtes englobantes.

A notre connaissance, une telle combinaison n'a pas déjà été exploitée pour l'optimisation des algorithmes d'évaluation d'arbres CSG.

V.2. Méthode de base: découpage versus classification

Notre méthode de base s'inspire de la méthode de découpage proposée par Rossignac et Voelcker [RV-89]. Nous verrons en V.3, comment l'utilisation des boîtes de Cameron permet d'optimiser cette méthode.

Soit F un élément de frontière (par exemple, une face) appartenant à une primitive A de S. Tant que $F \neq \emptyset$, on découpe F successivement contre chaque *nœud de branchement* de A dans S. Pour découper F contre un nœud de branchement donné B, on commence par décomposer F en sous-ensembles connexes f, de classification *constante* par rapport à B (i.e. l'intérieur de f est soit entièrement *dans* B, soit entièrement *hors* de B, soit entièrement sur la frontière de B, dans la topologie de f).

Cette décomposition revient à subdiviser F le long de ses intersections 1D (listes de segments, dans le cas polyédrique) avec les faces de toutes les primitives de B. Ensuite, pour chaque sous-ensemble f, on teste si la condition $(\vartheta_+ \cap I \neq \emptyset)$ et $(\vartheta_- \cap \overline{U} \neq \emptyset)$ est vérifiée par un point p quelconque intérieur à f et on élimine les sous-ensembles pour lesquels elle ne l'est pas (voir Sect. IV.3).

A chaque nœud B, l'une seulement des deux inégalités est à vérifier, selon que B est un *I-nœud* (auquel cas, on teste si $\vartheta_+ \cap I \neq \emptyset$) ou un *U-nœud* (auquel cas, on teste si $\vartheta_- \cap \overline{U} \neq \emptyset$). D'autre part, la décomposition de F décrite précédemment garantit qu'à chaque nœud B, les voisinages ϑ_+ et ϑ_- ne peuvent prendre que deux valeurs possibles: $(\vartheta_+ \cap I = \emptyset$ ou $\vartheta_+ \cap I = \vartheta_+)$ si B est un *I-nœud*, $(\vartheta_- \cap \overline{U} = \emptyset$ ou $\vartheta_- \cap \overline{U} = \vartheta_-)$ si B est un *U-nœud*. Par conséquent, ces valeurs sont constantes pour tous les points p intérieurs à un sous-ensemble f donné, et f est éliminé dès que ϑ_+ ou ϑ_- devient *vide*.

Les arêtes appartenant à la frontière de S s'obtiennent à partir des frontières des sous-faces f de F retenues lors du découpage.

Si on note H l'objet B si B est un *I-nœud* ou l'objet $\overline{B}$ si B un *U-nœud*, la procédure ci-dessus peut être résumée comme suit:

1) *classifier* F par rapport à B en utilisant l'approche standard ;

2) *éliminer* celui des deux sous-ensemble *F-hors-B* et *F-dans-B* qui se trouve hors de H ;

3) *éliminer* les parties de *F-sur-B* qui *séparent* l'intérieur de A de l'intérieur de H (Fig. 8).

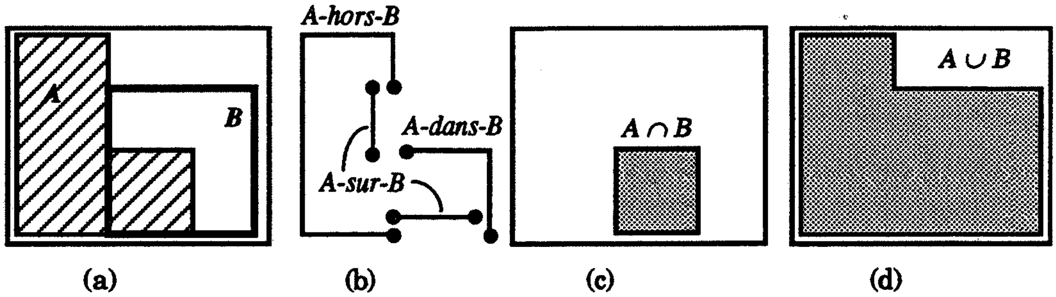


Figure 8. Découpage d'une primitive A contre un nœud de branchement B : **a)** A est la forme hachurée et B est le carré en gras. **b)** Classification de A par rapport à B . **c)** B est un *I-nœud*, alors (comme pour calculer $A \cap B$) on élimine les éléments de ∂A du type *A-hors-B*, ainsi que parmi les éléments du type *A-sur-B*, ceux qui séparent l'intérieur de A de l'intérieur de B. **d)** B est un *U-nœud*, alors (comme pour calculer $A \cup B$) on élimine les éléments de ∂A du type *A-dans-B*, ainsi que parmi les éléments du type *A-sur-B*, ceux qui séparent l'intérieur de A de l'intérieur de $\bar{B}$.

V.3. Exploitation des boîtes de Cameron

Une optimisation simple et efficace consiste à vérifier, avant chaque découpage, si l'élément X de ∂A ne se trouve pas *hors* de la boîte $\beta(B_i)$ associée au nœud de branchement courant B_i. Auquel cas, X peut être éliminé trivialement: il est nécessairement hors de $A \cap Z(A)$ et il devient inutile de poursuivre son découpage contre les nœuds de branchement B_j $(j \geq i)$; autrement, et seulement alors, X est effectivement découpé contre le nœud B_i.

Le test de boîtes de Cameron peut être mis en œuvre de deux manières:

• la première consiste simplement à vérifier si X (ou plus simplement encore, sa boîte englobante) intersecte ou non la boîte $\beta(B)$;

• la deuxième consiste à découper *explicitement* X contre la boîte $\beta(B)$: par exemple en utilisant l'algorithme de fenêtrage polygonal de Liang-Barsky [LB-84].

Cependant, la deuxième solution n'est pas intéressante en pratique: outre son coût, elle a pour effet d'introduire des arêtes "artificielles" provenant des frontières des boîtes de Cameron (et non de celles des primitives). Nous lui préférons la première solution.

V.4. Coût de cette méthode

Dans le meilleur des cas, X est éliminé dès sa comparaison avec la boîte du premier nœud de branchement de A, et aucune opération de découpage n'aura été effectuée. Dans le pire des cas (rarissime), tous les nœuds de branchement sont considérés et aucune des boîtes correspondantes ne permet d'éliminer X trivialement ; dans ce cas, le surcoût dû à la prise en compte des boîtes de Cameron est de toute façon négligeable devant le coût global de découpage de X. Le découpage contre un nœud de branchement B nécessite le calcul des intersections entre X et les frontières de toutes les primitives de B, alors que le coût du test d'intersection entre X et $\beta(B)$ est seulement proportionnel au nombre d'arêtes de X (une face de A).

VI. MISE EN ŒUVRE DE CETTE METHODE

VI.1. Structure de données

Un nœud de l'arbre de construction est supposé représenté par une structure CSG du type: *(filsGauche, filsDroit, père, opération, primitive, boîte)*.

Une primitive est un objet polyédrique quelconque, de frontière connue. La structure PRIMITIVE contient, entre autres, un champ *signe* qui indique si la primitive est bornée (+1) ou non bornée (-1).

Une boîte isothétique bornée et non vide est définie par une structure BOITE du type (*xmin, xmax, ymin, ymax, zmin, zmax*). La boîte vide ([]) et la boîte infinie (Ω) sont représentées par deux adresses (pointeurs) uniques: *laBoîteVide* et *laBoîteInfinie*, respectivement.

Enfin, la structure de données FRONT utilisée pour la représentation de la frontière des objets doit être suffisamment riche et souple pour rendre compte efficacement des découpages successifs des faces et des arêtes. Par exemple, chaque ARETE d'une primitive contient une liste pour toutes les extrémités de *sous-arêtes* qui résultent de l'intersection de cette arête avec les diverses faces des autres primitives (voir Chap. II) et chaque FACE connexe contient une liste de CONTOURS, correctement orientés.

VI.2. Structure des algorithmes

Les pseudo-codes ci-dessous ne sont pas définitifs: leur but est de décrire de manière concise une organisation possible des divers concepts présentés au cours de ce chapitre.

• La procédure *formePositive* ci-dessous transforme l'arbre initial en sa forme positive équivalente. Parallèlement, elle initialise les boîtes de tous les nœuds de l'arbre en tenant compte du *signe* des primitives. L'appel se fait par: *formePositive* (racine, +1).

```
formePositive (nœud : CSG, signe : SIGNE)
début
    si primitive (nœud) alors
        nœud->primitive->signe := signe ;
        nœud->boîte := si (signe = +1)
                         alors enveloppeIsothétique (nœud->primitive)
                         sinon laBoîteInfinie fsi ;
    sinon
        nœud->boîte := laBoîteInfinie ;
        op := nœud->opération ;
        si (signe = +1) alors
            si (op = '-') alors nœud->opération := '∩' fsi
        sinon
            nœud->opération := si (op = '∪') alors '∩' sinon '∪' fsi
        fsi ;
        formePositive (nœud->filsGauche , signe) ;
        signeDroit := si (op = '-') alors (-signe) sinon signe fsi ;
        formePositive (nœud->filsDroit, signeDroit)
    fsi
fin
```

• La fonction *évaluer* calcule la frontière de l'objet correspondant au nœud courant de l'arbre CSG. Les frontières obtenues sont "concaténées" au fur et à mesure des appels récursifs (la fonction *concaténer* fusionne les arêtes inutilement "alignées", les faces inutilement "coplanaires", ...).

```
évaluer (nœud : CSG) : FRONT
début
    si (primitive (nœud)) alors
        front := découperPrimitive (nœud)
    sinon
        front := concaténer (évaluer (nœud->filsGauche),
                             évaluer (nœud->filsDroit))
    fsi ;

    rendre front
fin
```

• La fonction *découperPrimitive* construit la frontière d'une primitive, puis lance son découpage contre la zone active de la primitive. La fonction *construireFrontièreDePrimitive* renvoie une frontière correctement orientée, en tenant compte du signe de la primitive :

```
découperPrimitive (nœud : CSG) : FRONT
début
    front := construireFrontièreDePrimitive (nœud->primitive) ;
    rendre  découperContreZ (front, nœud)
fin
```

• La fonction *découperContreZ* réalise le découpage d'une représentation par frontière, successivement contre chaque nœud de branchement de la primitive en cours de découpage. Le découpage s'arrête dès que la frontière est réduite à l'ensemble *vide*, ou que tous les nœuds de branchement ont été considérés. On suppose que chaque nœud de l'arbre dispose de l'adresse du nœud "père", de sorte qu'il est possible de retrouver le chemin qui relie la primitive à la racine de l'arbre (ici, les nœuds de branchement sont visités de bas en haut).

```
découperContreZ (front: FRONT, nœud: CSG) : FRONT
début
    si vide (front) rendre ∅ fsi ;
    si (nœud->père ≠ nil) rendre front fsi ; { on a traité tous les nœuds de brancht .}
    père := nœud->père ;
    branch := si (père->filsGauche = nœud) alors père->filsDroit
                                             sinon père->filsGauche ;
    inœud := (père->opération ='∩') ;
    nouvelleFront : = découperContreNœud (front, branch, inœud) ;
    rendre  découperContreZ (nouvelleFront, père)
fin
```

• La fonction *découperContreNœud* découpe une représentation par frontière contre un nœud de branchement B de la primitive courante A. La frontière est d'abord *classifiée* par rapport au sous-arbre de racine B ; ensuite, chaque élément de frontière f (*homogène* par rapport à B) est testé contre la boîte de Cameron associée à B (fonction *horsBoîte*) puis, au besoin, f est soumis au test d'élimination présenté en Section V.4.3. Seuls les éléments f ayant "passé" ces deux tests sont retenus dans le découpage (fonction *ajouter*).

découperContreNœud (front: FRONT, *B* : CSG, inœud: Booléen) : FRONT
<u>début</u>
    frontSubdivisée := *classifier* (front, branch) ;
    nouvelleFront := ∅ ;
    <u>pour</u> chaque *f* dans frontSubdivisée <u>faire</u>
        <u>si</u> (*horsBoîte* (*f*, branch->boîte)) <u>rendre</u> ∅ <u>fsi</u> ;
    { soit *C* l'objet représenté par *B* si "inœud" est *vrai*, par $\overline{B}$ sinon}
        <u>si</u> ("*f* sépare l'intérieur de *A* de l'intérieur de *C*") <u>alors</u>
            ajouter (*f*, nouvelleFront)
        <u>fsi</u>
    <u>fait</u> ;
    <u>rendre</u> nouvelleFront
<u>fin</u>

• La fonction *horsBoîte* (*f*, β) réalise le test de boîtes de Cameron. Elle vérifie si une face *f* intersecte ou non la boîte β d'un nœud de branchement. Une variante consiste à découper systématiquement *f* contre β, par exemple au moyen de l'algorithme de Liang-Barsky [LB-84].

• Les trois procédures ci-dessous sont classiques dans les algorithmes de résolution d'arbres CSG. Une présentation générale en est donnée dans [Tilo-80] et dans [RV-85].

classifier (front : FRONT, branch: CSG) : FRONT
<u>début</u>
    frontSubdivisée := ∅ ;
    <u>pour</u> chaque *F* dans front <u>faire</u>
        classif := *classifierFace*(*F*, branch) ;
        ajouter (classif, frontSubdivisée)
    <u>fait</u> ;

    <u>rendre</u> frontSubdivisée
<u>fin</u>

```
classifierFace (F : FACE, nœud: CSG): (Dans, Sur, Hors)
début
    si primitive (nœud) alors
        rendre classifierFaceContrePrimitive (F, nœud->primitive)
    sinon
        rendre Combiner (classifierFace (F, nœud->filsGauche),
                         classifierFace (F, nœud->filsDroit),
                         nœud->opération)
    fsi

fin
```

— o —

```
Combiner (C1, C2: (Dans, Sur, Hors), op : OPERATION): (Dans, Sur, Hors)
début
    { voir "SetMemberShip Classification" [Tilo-80] }
fin
```

• La procédure *classifierFaceContrePrimitive* (*F*, *P*) décompose une face *F* en sous-faces "homogènes" par rapport à une primitive *P*. Typiquement, les primitives sont des polyèdres *convexes*, de sorte que l'on peut appliquer l'algorithme réentrant de Sutherland-Hodgman [SH-74].

Cet algorithme découpe correctement un polygone quelconque contre une *fenêtre convexe* quelconque (conjointement, l'algorithme de Cyrus-Beck [CB-78] sert au calcul efficace des points d'intersection entre les arêtes de *X* et les plans de $\beta(B)$).

• La procédure *StabiliserBoîtes* détermine l'*état stable* de l'ensemble des boîtes de l'arbre (Sect. III.4). La variable booléenne *nouvelleBoîteVide* est utilisée pour détecter la création d'une nouvelle boîte vide au cours d'une phase de montée (procédure *monterBoîtes*) ou de descente (procédure *descendreBoîte*) des boîtes.

```
StabiliserBoîtes (nœud : CSG)
début
 si (non primitive (nœud)) alors
    nouvelleBoîteVide := Faux ;
    répéter
        monterBoîtes (nœud) ;
        descendreBoîtes (nœud) ;
    jusqu'à (non nouvelleBoîteVide)
 fsi
fin
```

• Enfin, la fonction *évaluerArbre* ci-dessous constitue la fonction "principale" qui déclenche l'évaluation de l'arbre CSG.

```
évaluerArbre (arbre : CSG) : FRONT
début
    si (arbre = nil ) rendre Ø fsi ;
    formePositive (arbre) ;
    stabiliserBoîtes (arbre) ;
    rendre évaluer (arbre)
fin
```

VII. CONCLUSION

Dans ce chapitre, nous avons proposé une optimisation possible pour les algorithmes de résolution d'arbres CSG, qui combine deux techniques développées indépendamment: les zones actives de Rossignac et Voelcker [RV-91] et les boîtes de Cameron [Came-91].

La méthode proposée s'inspire du même principe général que l'arithmétique *paresseuse*, présentée au chapitre III. Le but commun est d'éviter d'effectuer des calculs exacts (mais coûteux) partout où des calculs approchés (mais simples) permettent de conclure.

Dans le cas de l'évaluation d'un arbre CSG S, la boîte de Cameron associée à chaque nœud B de S fournit une approximation simple qui englobe la *contribution effective* de ∂B à ∂S. Cette contribution n'a pas besoin d'être calculée exactement lorsque la boîte $\beta(B)$ s'avère suffisante: de ce point de vue, les boîtes de Cameron jouent un rôle analogue à celui des *intervalles* des nombres paresseux.

Cette optimisation est compatible avec les algorithmes d'évaluation de frontières existants et en particulier les algorithmes basés sur la méthode non incrémentale.

L'algorithme incrémental détaillé au chapitre II peut être remanié de façon à exploiter la *paresse* à deux niveaux distincts. D'abord à un niveau *inférieur* (indépendant de l'algorithme), grâce à l'emploi de l'arithmétique *paresseuse*, puis à un niveau *supérieur* (dépendant de l'algorithme), grâce à l'utilisation conjointe des zones actives et des boîtes de Cameron.

L'algorithme résultant a alors un double objectif: 1) éviter au maximum les calculs *géométriques* inutiles et 2) n'effectuer exactement que les calculs *numériques* strictement nécessaires.

Raisonnablement, on peut s'attendre à ce que les performances pratiques d'un tel algorithme soient meilleures que celles des algorithmes classiques.

ANNEXE
BOITES DE CAMERON

Dans cette annexe, nous proposons une démonstration du théorème de convergence des boîtes de Cameron, tel qu'énoncé en III.4. La preuve donnée dans [CY-92] ne tient pas compte des opérateurs *différence* et utilise quelques résultats qui réclament une justification théorique.

A.1. Démonstration du théorème de convergence

Etant donné un ensemble de boîtes *initiales*, il existe toujours un *état stable* vers lequel *convergent* toutes les boîtes de l'arbre considéré. En effet, si n est le nombre total de *primitives*, alors chacune des six bornes définissant une boîte ne peut prendre qu'au plus n valeurs possibles, du moment que la composition des boîtes ne crée jamais de nouvelles bornes. Par conséquent, le nombre de boîtes distinctes susceptibles d'être créées est fini. D'autre part, les boîtes successives d'un même nœud A forment une suite décroissante (au sens de l'inclusion), minorée par la boîte vide.

Plus, précisément, nous démontrons maintenant le théorème suivant :

Théorème :

Si l'état courant des boîtes d'un arbre CSG S est tel que l'on ne crée pas de nouvelle boîte vide au cours d'une phase de montée suivie d'une descente, alors le système de boîtes de S est dans son état stable.

Nous commençons par établir le lemme suivant :

Lemme :

Une condition suffisante pour que le système de boîtes d'un arbre CSG S soit dans son état stable est qu'il vérifie les trois conditions suivantes :

C1) si $T = G \cap D$ alors $\beta(T) = \beta(G) = \beta(D)$;

C2) si $T = G \cup D$ alors $\beta(T) = \beta(G) \boxed{\cup} \beta(D)$;

C3) si $T = G - D$ alors $\beta(G) = \beta(T)$ et $\beta(D) \subseteq \beta(T)$.

Preuve du Lemme :

Il suffit de montrer que les boîtes restent invariantes par application d'une règle de *montée* ou de *descente*. Pour cela, on utilisera les propriétés ci-dessous, évidentes pour la composition de boîtes *isothétiques*. Pour toutes boîtes X et Y, on a:

$X \boxed{\cap} Y = Y \boxed{\cap} X$ et $X \boxed{\cup} Y = Y \boxed{\cup} X$ (*commutativité*) ;

$X \boxed{\cap} X = X$ et $X \boxed{\cup} X = X$ (*idempotence*) ;

$(X \boxed{\cap} Y) \subseteq X$ et $(X \boxed{\cap} Y) \subseteq Y$ (l'*intersection* "réduit" les boîtes);

$X \subseteq (X \boxed{\cup} Y)$ et $Y \subseteq (X \boxed{\cup} Y)$ (l'*union* "grossit" les boîtes).

- Pour tout nœud $T = G \cap D$ vérifiant (C1), on a:

$\beta'(T) = \beta(T) \boxed{\cap} (\beta(G) \boxed{\cap} \beta(D))$, après une *montée* ;

$= \beta(T) \boxed{\cap} (\beta(T) \boxed{\cap} \beta(T))$, car $\beta(T) = \beta(G) = \beta(D)$;

$= \beta(T)$, par *idempotence*.

$\beta'(G) = \beta(T) \boxed{\cap} \beta(G)$, après une *descente* ;

$= \beta(G) \boxed{\cap} \beta(G)$, car $\beta(T) = \beta(G)$;

$= \beta(G)$, par *idempotence* ;

$\beta'(D) = \beta(D)$, par symétrie.

- Pour tout nœud $T = G \cup D$ vérifiant (C2), on a:

$\beta'(T) = \beta(T) \boxed{\cap} (\beta(G) \boxed{\cup} \beta(D))$, après une *montée* ;

$= \beta(T) \boxed{\cap} \beta(T)$, car $\beta(G) \boxed{\cup} \beta(D) = \beta(T)$;

$= \beta(T)$, par *idempotence* ;

$\beta'(G) = \beta(T) \boxed{\cap} \beta(G)$, après une descente ;

$= \beta(\mathrm{G})$, du fait que $\beta(G) \subseteq (\beta(G) \boxed{\cup} \beta(D)) = \beta(T)$.

$\beta'(D) = \beta(D)$, par symétrie.

• Enfin, pour tout nœud $T = G - D$ vérifiant (C3), on a:

$\beta'(T) = \beta(T) \boxed{\cap} \beta(G)$, après une *montée* ;

$= \beta(T) \boxed{\cap} \beta(T)$, car $\beta(G) = \beta(T)$;

$= \beta(T)$, par *idempotence* ;

$\beta'(G) = \beta(T) \boxed{\cap} \beta(G)$, après une *descente* ;

$= \beta(G) \boxed{\cap} \beta(G)$, car $\beta(T) = \beta(G)$;

$= \beta(G)$, par *idempotence* ;

$\beta'(D) = \beta(T) \boxed{\cap} \beta(D)$, après une *descente* ;

$= \beta(D)$, car $\beta(D) \subseteq \beta(T)$. []

Preuve du théorème :

Soient β l'état courant du système de boîtes et β' le nouvel état qui résulterait d'une phase de *montée* suivie d'une phase de *descente* (Fig. 9). En supposant qu'aucune *nouvelle* boîte *vide* n'est créée pour un nœud interne *T*, nous montrons que le système β' vérifie les conditions du lemme, ce qui impliquera sa stabilité.

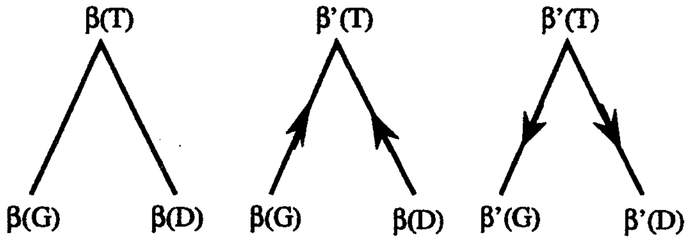


Figure 9. A gauche, l'état initial des boîtes en un nœud T donné. Au centre, l'état des boîtes après une *montée*: $\beta'(T) = \beta(T) \boxed{\cap} (\beta(G) \boxed{\%} \beta(D))$, où % désigne l'opération portée par T. A droite, l'état final des boîtes après une phase supplémentaire de *descente*: $\beta'(G) = \beta'(T) \boxed{\cap} \beta(G)$ et $\beta'(D) = \beta'(T) \boxed{\cap} \beta(D)$.

Notons d'abord que si un nœud *T* a déjà une boîte*vide* dans l'état initial β, alors tous ses descendants ont nécessairement des boîtes vides ; autrement, la phase de *descente* les forcera à des boîtes *vides*, ce qui contredirait l'hypothèse que l'on n'a pas créé de *nouvelles* boîtes *vides*. Les conditions du lemme se vérifient trivialement pour chaque nœud dont la boîte est *vide*.

Considérons maintenant le cas où toutes les boîtes résultantes sont *non vides*. En s'aidant de la Figure 9, les boîtes β' obtenues à l'issue d'une étape montée/descente vérifient :

• pour tout nœud $T = G \cap D$

$$\beta'(T) = \beta(T) \boxed{\cap} (\beta(G) \cap \beta(D)), \text{ après la } \textit{montée} \text{ ;}$$

$$\beta'(G) = \beta'(T) \boxed{\cap} \beta(G) = \beta'(T), \text{ car } \beta'(T) \subseteq \beta(G) \text{ ;}$$

$$\beta'(D) = \beta'(T) \boxed{\cap} \beta(D) = \beta'(T), \text{ car } \beta'(T) \subseteq \beta(D) \text{ ;}$$

donc, $\beta'(G) = \beta'(D) = \beta'(T)$;

• pour tout nœud $T = G - D$

$$\beta'(T) = \beta(T) \boxed{\cap} \beta(G)), \text{ après la } \textit{montée} \text{ ;}$$

$$\beta'(G) = \beta'(T) \boxed{\cap} \beta(G) = \beta'(T), \text{ car } \beta'(T) \subseteq \beta(G) \text{ ;}$$

$$\beta'(D) = \beta'(T) \boxed{\cap} \beta(D) \subseteq \beta'(T) \text{ ;}$$

donc, $\beta'(T) = \beta'(G)$ et $\beta'(D) \subseteq \beta'(T)$;

• enfin, pour tout nœud $T = G \cup D$

$$\beta'(T) = \beta(T) \boxed{\cap} (\beta(G) \boxed{\cup} \beta(D)), \text{ après la } \textit{montée} \text{ ;}$$

$$\beta'(G) \cup \beta'(D) = (\beta'(T) \boxed{\cap} \beta(G)) \boxed{\cup} (\beta'(T) \cap \beta(D)) \text{ ;}$$

$$= \beta'(T) \boxed{\cap} (\beta(G)) \boxed{\cup} \beta(D)), \text{ par } \textit{distributivité} \text{ ;}$$

$$= \beta'(T), \text{ car } \beta'(T) \subseteq (\beta(G) \boxed{\cup} \beta(D)) \text{ ;}$$

donc, $\beta'(G) \boxed{\cup} \beta'(D) = \beta'(T)$. []

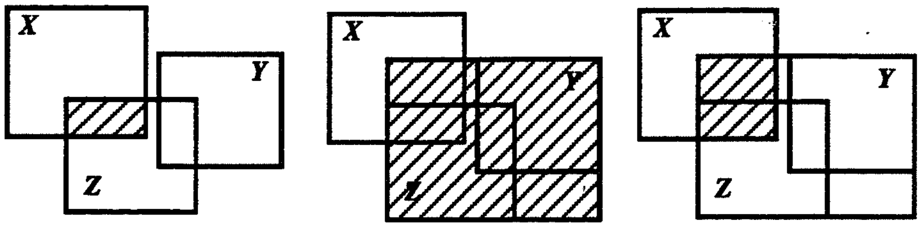
Remarque

Il convient de justifier la propriété de *distributivité* (D) ci-dessous, exploitée précédemment pour la démonstration du théorème de convergence :

$$(\beta'(T) \boxed{\cap} \beta(G)) \boxed{\cup} (\beta'(T) \boxed{\cap} \beta(D)) = \beta'(T) \boxed{\cap} (\beta(G)) \boxed{\cup} \beta(D)) \quad \textbf{(D)}.$$

Cette propriété est en général *fausse* dans $\Re^n$ ($n \geq 2$), comme en témoigne le contre-exemple de la Figure 10. Formellement, l'ensemble des boîtes isothétiques, muni des opérations $\boxed{\cap}$ et $\boxed{\cup}$ n'est pas une algèbre booléenne.

Cependant, nous allons montrer que cette propriété s'applique effectivement sous l'hypothèse du théorème de convergence, selon laquelle toutes les boîtes résultantes sont *non vides*.



$(X \boxed{\cap} Y) \boxed{\cup} (X \boxed{\cap} Z)$ $\quad$ $Y \boxed{\cup} Z$ $\quad$ $X \boxed{\cap} (Y \boxed{\cup} Z)$

Figure 10. L'opération $\boxed{\cap}$ n'est pas distributive par rapport à $\boxed{\cup}$.

Commençons par montrer que la propriété de distributivité est toujours vraie dans le cas des intervalles de $\Re^1$. Soient X, Y et Z trois intervalles *non vides*, définis par $X = [x_1, x_2]$, $Y = [y_1, y_2]$ et $Z = [z_1, z_2]$, respectivement.

En partant des règles de composition d'intervalles

$$X \boxed{\cap} Y = [\mathrm{Max}\,(x_1, y_1)\,,\ \mathrm{Min}\,(x_2, y_2)]\,,$$
$$X \boxed{\cup} Y = [\mathrm{Min}\,(x_1, y_1)\,,\ \mathrm{Max}\,(x_2, y_2)]$$

et des deux identités

$$\mathrm{Max}\,(a\,, \mathrm{Min}\,(b\,, c)) = \mathrm{Min}\,(\mathrm{Max}\,(a\,, b), \mathrm{Max}\,(a\,, c))\,,$$
$$\mathrm{Min}\,(a\,, \mathrm{Max}\,(b\,, c)) = \mathrm{Max}\,(\mathrm{Min}\,(a\,, b), \mathrm{Min}\,(a\,, c))\,,$$

on obtient :

$$\begin{aligned}
X \boxed{\cap} (Y \boxed{\cup} Z) &= [x_1\,, x_2] \boxed{\cap} (\,[y_1\,, y_2] \boxed{\cup}\ [z_1\,, z_2]) \\
&= [x_1, x_2] \boxed{\cap} [\mathrm{Min}\,(y_1\,, z_0)\,, \mathrm{Max}\,(y_2, z_2)] \\
&= [\,\mathrm{Max}\,(x_1\,, \mathrm{Min}\,(y_1\,, z_1))\,, \mathrm{Min}\,(x_2\,, \mathrm{Max}\,(y_2\,, z_2))] \\
&= [\mathrm{Min}\,(\mathrm{Max}\,(x_1\,, y_1)\,, \mathrm{Max}\,(x_1\,, z_1))\,, \mathrm{Max}\,(\mathrm{Min}\,(x_2\,, y_2)\,, \mathrm{Min}\,(x_2\,, z_2))] \\
&= [\mathrm{Max}\,(x_1, y_1)\,, \mathrm{Min}\,(x_2\,, y_2)] \boxed{\cup} [\mathrm{Max}\,(x_1\,, z_1)\,, \mathrm{Min}\,(x_2\,, z_2)] \\
&= (\,[x_1\,, x_2] \boxed{\cap} [y_1\,, y_2]\,) \boxed{\cup} (\,[x_1, x_2] \boxed{\cap} [z_1\,, z_2]) \\
&= (X \boxed{\cap} Y) \cup (X \boxed{\cap} Z).
\end{aligned}$$

Une boîte isothétique X de $\Re^n$ ($n \geq 2$) s'exprime comme un *produit cartésien* de n intervalles X_i, qui préserve bien la *n-dimensionnalité* des boîtes: un produit cartésien est *non vide* si, et seulement si, aucun des ses *facteurs* n'est *vide*. Or, par hypothèse du théorème de convergence, toutes les boîtes résultantes sont *non vides*. Dans ces conditions, la propriété de distributivité établie dans le cas 1D s'applique à chacune des n dimensions, séparément. []

A.2 Une borne supérieure pour le théorème de convergence

Le théorème de convergence implique le corollaire suivant :

Corollaire :

L'ensemble des boîtes d'un arbre CSG binaire à n primitives atteint son état stable en au plus (n-1) étapes (montée, descente).

Preuve :

Dans le pire des cas, l'arbre considéré correspond à un objet *vide* et l'on ne crée qu'une seule boîte*vide* pour chacun des (n-1) nœuds binaires, au cours d'une étape montée/descente. []

Exemple où la borne supérieure est atteinte

L'exemple ci-dessous montre que le nombre d'étapes nécessaires à la convergence peut être égal à (n-1).

Considérons un arbre CSG *binaire* défini comme une *intersection* entre une *union* de (n/2) barres verticales et une *union* de (n/2) barres horizontales, disposées comme sur la Figure 11. On constate qu'après chaque phase de *descente*, on produit une boîte *vide* pour la barre la plus haute ou la plus à droite, alternativement. Il faut donc exactement (n-1) étapes (*montée, descente*) pour atteindre la convergence.

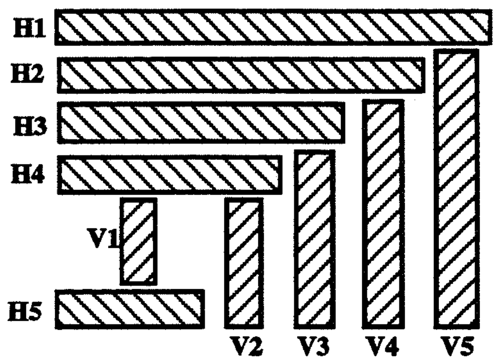


Figure 11. S = ((((H1 ∪ H2) ∪ H3) ∪ H4) ∪ H5) ∩ ((((V1 ∪ V2) ∪ V3) ∪ V4) ∪V5).

Remarque

Dans le domaine de la *modélisation constructive* d'objets solides, il est rare de créer une boîte*vide* car cela supposerait que le nœud correspondant dans l'arbre CSG est totalement inutile à la description CSG de l'objet. Aussi, dans ce domaine, une seule étape montée/descente est suffisante pour obtenir l'état stable.

En revanche, dans le domaine de la *détection d'interférences*, les boîtes vides peuvent apparaître du fait de la nature du problème. Cameron [Came-91] observe tiutefois qu'il suffit de *deux* ou *trois* étapes pour atteindre la convergence, en pratique. Nous avons vérifié ce fait dans le cas des intervalles, sur des arbres CSG ayant jusqu'à 99 nœuds (dont 50 intervalles primitifs, générés au hasard).

CONCLUSION

Dans cette thèse, nous avons abordé les principaux problèmes auxquels restent confrontés les algorithmes de résolution d'opérations booléennes sur les solides représentés par leurs frontières.

Nous avons proposé une solution qui tient compte d'une part de l'explosion des cas particuliers et d'autre part des effets désastreux des imprécisions numériques sur la cohérence des objets. La réalisation pratique d'un module d'évaluation d'arbres CSG nous a permis de mieux comprendre ces problèmes.

Une arithmétique rationnelle optimisée, dite paresseuse, a été présentée. Elle résout tous les inconvénients des arithmétiques rationnelles classiques. Nous pensons que cette arithmétique offre une solution radicale au problème des imprécisions numériques, à un coût très raisonnable.

L'arithmétique paresseuse est accessible à tous les algorithmes géométriques. Nous avons mis en évidence le concept de programmation "avertie" qui conduit à des programmes relativement robustes en arithmétique flottante, et totalement robustes et efficaces en arithmétique paresseuse. En particulier, ce concept s'est révélé très efficace pour l'implantation de l'algorithme d'évaluation d'arbres CSG.

Une autre méthode d'optimisation a été présentée. Elle consiste à réduire le volume des calculs géométriques nécessaires dans les divers algorithmes d'évaluation de frontières. Par cette méthode, nous avons voulu illustrer une généralisation simple du principe fondamental de la "paresse".

Enfin, il serait intéressant d'étudier dans quelle mesure l'arithmétique rationnelle paresseuse peut s'étendre au domaine algébrique. Cette extension permettrait d'envisager, entre autres, la résolution robuste d'opérations booléennes sur les solides non polyédriques.

BIBLIOGRAPHIE

[ABBN 91] D. Ayala, F. Batlle, P. Brunet, and I. Navazo. *Boolean Operations between Extended Octrees.* Research Report LSI-91-31, Dept. of Languages and Computer Science Systems, University of Catalonia (Spain), 1991.

[Ala 91] S. R. Ala. *Design Methodology of Boundary Data Structures.* Proc. Symp. on Solid Modeling Foundations and CAD/CAM Appl., June 1991, Austin (Texas), ACM Press, pp.13-23.

[Ala 92] S. R. Ala. *Perfomance Anomalies in Boundary Data Structures.* IEEE CG & A, March 1992, pp.49-58.

[Allen 84] G. Allen. *An Introduction to Solid Modeling.* Computer & Graphics (1984), Vol. 8, No. 4, pp. 439-447.

[Ather 83] P. Atherton. *A Scan-Line Hidden Surface Removal Procedure for Constructive Solid Geometry.* SIGGRAPH'83, ACM Computer Graphics, July 1983, Vol. 17, No. 3, pp.73-82.

[Arbab 90] F. Arbab. *Set Models and Boolean Operations for Solids and Assemblies.* IEEE CG & A, Nov. 1990, pp. 76-86.

[AFF 85] S. Ansaldi, L. de Floriani, and B. Falcidieno. *Geometric Modeling of Solid Objects by Using a Face Adjacency Graph Representation.* SIGGRAPH'85, ACM Computer Graphics, July 1985, Vol. 19, No. 3, pp. 131-139.

[Barr 84] A. H. Barr. *Global and Local Deformations of Solid Primitives.* SIGGRAPH'84, ACM Computer Graphics, July 1984, Vol. 18, No. 3, pp. 21-31.

[Baum 72] B. G. Baumgart. *Winged-Edge Polyhedron Representation.* Report No. CS-320, Stanford A.I. Lab., Stanford University, Oct. 1972.

[Baum 75] B. G. Baumgart. *A polyhedron Representation for Computer Vision.* AFIPS National Computer Conference Proc. 44, 1975, pp. 589-596.

[Beig 88] M. Beigbeder. *Un développement pour la modélisation et la visualisation en synthèse d'images.* Thèse de Doctorat en Informatique, Ecole Nationale Supérieure des Mines de Saint-Etienne, Avril 1988.

[BH 88] D. Badouel et G.Hegron. *Opérations booléennes sur les polyédres: évaluation d'arbres CSG.* Rapport INRIA-839, Avril 1988.

[BHS 80] I. C. Braid, R.C. Hillyard, and I. A. Stroud. *Stepwise Construction of Polyhedra in Geometric Modeling.* In "Mathematical Methods in Computer Graphics and Design", Academic Press, London (1980), pp. 123-141.

[BJMM 93a] M.O. Benouamer, P. Jaillon, D. Michelucci, and J.M. Moreau. *A Lazy Exact Arithmetic Library*.11th. IEEE Symposium on Computer Arithmetic, 30 June-2-July (1993), Windsor, Ontario, Canada.

[BJMM 93b] M.O. Benouamer, P. Jaillon, D. Michelucci, and J.M. Moreau. *A Lazy Solution to Imprecision in Computational Geometry*. 5th Canadian Conference on Computational Geometry. July 8-10 (1993), Waterloo, Ontario, Canada.

[BJMM 93c] M.O. Benouamer, P. Jaillon, D. Michelucci, and J.M. Moreau. *Hashing Lazy Numbers*. SCAN'93, September 1993, Vienna, Austria.

[BMP 93] M.O. Benouamer, D. Michelucci, and B. Péroche. *Error-Free Boundary Evaluation Using A Lazy Rational Arithmetic: A Detailed Implementation*. Proc. of the 2nd. ACM/IEEE Symposium on Solid Modeling and Applications, Montreal (Canada), May 19-21, 1993, pp. 115-126.

[BO 79] J. L. Bentley and T. Ottman. *Algorithms for Reporting and Counting Geometric Intersections*. IEEE Trans. on Computers, C-28(9), 1979, pp. 643-647.

[BP 90] M. Beigbeder et B. Péroche. *Un système de Synthèse D'Images 3D: Illumines*. Rapport interne No. 90.2, Ecole Nationale Supérieure des mines de Saint-Etienne, Fevrier 1990.

[BR 82] J. W. Boyse and J. M. Rosen. *GMsolid: Interactive Modeling for Design and Analysis of Solids*. IEEE CG&A 2(2), pp. 27-40, March 1982.

[CB 78] M. Cyrus and J. Beck. *Generalized Two- and Three-Dimensional Clipping*. Computers & Graphics, Vol. 3, pp. 23-28, 1978.

[CR 91] G. A. Crocker and W. F. Reinke. *An Editable Nonmanifold Boundary Representation*. IEEE CG & A, March 1991, pp. 39-51.

[Came 91] S. Cameron. *Efficient Bounds in Constructive Solid Geometry*. IEEE CG & A, May 1991, pp. 68-74.

[Casa 87] M. S. Casale. *Free-Form Solid Modeling with Trimmed Surface Patches*. IEEE CG & A, Jan. 1987, pp. 33-43.

[Chaz 80] B.M. Chazelle. *Computational Geometry and Convexity*. Dept. of Computer Science, Carnegie Melon University, Pittsburgh (Pensylvania), 1980.

[CK 83] H. Chiyokura and F. Kimura. *Design of Solids with Free-Form Surfaces*. SIGGRAPH'83, ACM Computer Graphics, July 1983, Vol. 17, No. 3, pp. 289-298.

[CY 92] S. Cameron and Chee-Keng Yap. *Refinement Methods for Geometric Bounds in Constructive Solid Geometry*. ACM Trans. on Graphics, Jan. 1992, Vol. 11, No. 1, pp. 12-39.

[DGHS 88] D. Dobkin, L. Guibas, J. Hershbeger, and J. Snoeyink. *An Efficient Algorithm for Finding the CSG Representation of a Simple Polygon*. SIGGRAPH'88, ACM Computer Graphics, Aug. 1988, Vol. 22, No. 4, pp. 31-40.

[DS 92] H. Desaulniers and N. F. Stewart. *An Extension of Manifold Boundary Representations to r-sets*. ACM Trans. on Graphics, Jan. 1992, Vol. 11, No. 1, pp. 40-60.

[EH 77] C. M. Eastman and M. Henrion. *GLIDE: A language for Design Information Systems*. ACM Comput. Graphics 11(2), pp. 24-33, July 1977.

[EM 88] H. Edelsbrunner and E. P. Mücke. *Simulation of Simplicity: A technique to Cope with Degenerate Cases in Geometric Algorithms*. Proc. of the 4th ACM Symp. on Computational Geometry, 1988, pp. 118-133.

[Emm 91] M. J. G. M. Van Emmerik. *Interactive Design of 3D Models with Geometric Constraints*. The Visual Computer, 1991, No. 7, pp. 309-325.

[EOS 86] H. Edelsbrunner, J. O'Rourke, and R. Seidel. *Constructing Arrangements of Lines and Hyperplanes with Applications*. SIAM Journal of Computing, 1986, Vol. 15, pp. 341-363.

[FP 91] V. Ferrucci and A. Paoluzzi. *Extrusion and Boudary Evaluation for Multidimensional Polyhedra*. CAD, Jan./Feb. 1991,Vol. 23, No. 1, pp. 40-50.

[FR 88] J. Flaquer and J. R. RODIL. *Boolean operations Based on the Planar Polyhedral Representation*. Comput. & Graphics, Vol. 12, No. 1, 1988, pp. 59-64.

[GCP 91] E. L. Gursoz, Y. Choi, and F. B. Prinz. *Boolean Set Operations on Non-manifold Boundary Representation Objects*. CAD, Jan./Feb. 1991, vol. 23, No. 1, pp. 33-39.

[GM 79] R. Goldstein, and L. Malin. *3D modeling with the Synthavision System*. Proc. 1st Annual Conf. on Computer Graphics in CAD/CAM Systems. April 1979, pp. 244-247.

[Gold 91] D. Goldberg. *What Every Computer Scientist should Know About Floating-Point Arithmetic*. ACM Computing Surveys, Vol. 23, No. 1, March 1991, pp. 5-48.

[GSS 89] L. Guibas, D. Salesin, and J. Stolfi. *Epsilon Geometry: Building Robust Algorithms From Imprecise Computations*. Proc. of the 5th ACM Symp. on Computational Geometry, 1989, pp. 208-217.

[GT 91] M. Gangnet and J.M. Van Thong. *Robust Boolean Operations on 2D Paths*. Proc. COMPUGRAPHICS'91, Sesimbra (Portugal), 1991, Vol. 2, pp. 434-443.

[GY 86] D. H. Greene and F. Yao. *Finite-resolution Computational Geometry*. Proc. of the 27th. Annual Symposium of the Foundations of Computer Science, 1986, pp. 143-152.

[HW 60] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, 4th Ed., 1960.

[HHK 88] C. M. Hoffmann, J. E. Hopcroft, and M. S. Karasick.*Towards Implementing Robust Geometric Computations*. Proc. of the 4th ACM Symp. on Computational Geometry, 1988, pp. 106-118.

[HHK 89] C. M. Hoffmann, J. E. Hopcroft, and M. S. Karasick. *Robust Set Operations on Polyhedral Solids*. IEEE CG & A, Nov. 1989, Vol. 9, No. 6, pp. 50-59.

[HMMN 84] S. Hertel, M. Mantyla, K. Mehlhorn, and J. Nievergelt. *Space Sweep Solves Intersection of Convex Polyhedra*. Acta Informatica **21**, Springer-Verlag (1984), pp. 501-519.

[Jaill 93] Ph. Jaillon. Thèse de Doctorat, en préparation à l'Ecole Nationale Supérieure des Mines de St-Etienne, 1993.

[Juan 88] R. Juan. *Boundary to Constructive solid Geometry: A Step towards 3D Conversion*. Proc. EUROGRAPHICS'88, Elsevier Science Publishers B. V. (North-Holland), 1988, pp. 129-139.

[KLN 89] M. Karasick, D. Lieber, and L. R. Nackman. *Efficient Delaunay Triangulation Using Rational Arithmetic*. IBM Research Report RC-14455, Aug. 1989.

[Kara 89] M. Karasick. *On the Representation and Manipulation of Rigid Solids*. Ph. D. Thesis, TR 89-976, Dept. of Computer Science, Cornell University, Ithaca, NY 14853-7501, March 1989.

[KM 81] U.W. Kulisch and W.L. Milanker. *Computer Arithmetic in Theory and Practice*, Academic Press, New York, 1981. Voir aussi par les mêmes auteurs : *A New Approach to Scientific Computation*. Academic Press, New York, 1982.

[Knut 81] D. E. Knuth. *Seminumerical Algorithms*. Addison-Wesley, Readings, Mass., 1981.

[LB 84] Y. D. Liang and B. Barsky. *A new Concept and Method Line Clipping*. ACM Trans. on Graphics, Vol. 3, pp. 1-22, 1984.

[LTH 86] D.H. Laidlaw, W.B. Trumbore, and J.F. Hugues. *Constructive Solid Geometry for Polyhedral Objects*. Proc. SIGGRAPH'86, ACM Computer Graphics, Aug. 1986, Vol. 20, No. 4, pp. 161-180.

[LST 90] X. Li, J. Sun, and Z. Tang. *An Approach to Improve the Reliability of Boolean Operations on a Pair of Polyhedra*. EUROGRAPHICS'90, Elsevier Science Publishers, North-Holland, 1990.

[MS 82] M. Mantyla and R. Sulonen. *GWB: A Solid Modeler with Euler Operators*. IEEE CG & A, Sept. 1982, Vol. 2, No. 7, pp. 18-31.

[MT 83] M. Mantyla and M. Tamminen. *Localised Set Operations for Solid Modeling*. Proc. SIGGRAPH'83, July 1983, ACM Computer Graphics, Vol. 17, No. 3, pp. 279-288.

[Mant 86] M. Mäntylä. *Boolean Operations of 2-Manifolds through Vertex Neighborhood Classification*. ACM Trans. on Graphics, Jan. 1986, Vol. 5, No. 1, pp. 1-29.

[Mant 88] M. Mantyla. *Solid Modeling*. Computer Science Press, 1988.

[Masu 92] H. Masuda. *Form-feature Representation based on Non-manifold Geometric Modeling*. Proc. MICAD'92, Paris, Fevrier 1992, Ed. Hermes,Vol. 1, pp. 17-35.

[Mehl 84] K. Mehlhorn. *Data Structures and Algorithms 3: Multidimensional Searching and Computational Geometry*. Springer-Verlag, Berlin, 1984.

[Mich 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, France, Nov. 1987.

[Mile 88] V. J. Milenkovic. *Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic*. Ph.D. Dissertation, CMU-CS-88-168, Carnegie-Mellon, July 1988.

[Mill 88] J. R. Miller. *Analyse of Quadric-Surfaces Based Solid Models.* IEEE CG & A, Jan. 1988, pp. 28-42.

[Mill 89] J. R. Miller. *Architectural Issues in Solid Modelers.* IEEE CG & A, Sept. 1989, pp. 72-87.

[Mill 93] J. R. Miller. *Incremental Boundary Evaluation Using Inference of Edge Classifications.* IEEE CG & A, Jan. 1993, pp. 71-78.

[ML 89] V. J. Milenkovic and Z. Li. *Constructing Strongly Convex Hulls Using Exact or Rounded Arithmetic.* Aiken Computational Lab., Cambridge, MA 02138, July 1989.

[MM 87] P. Martin et D. Martin. *Les difficultés et les erreurs possibles dans les algorithmes de clacul de l'intersection de solides définis par le bord.* MARI 87, Mai 1987, pp. 48-56.

[MN 90] V. J. Milenkovic and L. R. Nackman. *Finding Compact Coordinate Representations for Polygons and Polyhedra.* IBM Journal of Research and Development, Vol. 34, No. 5, Sept. 1990, pp. 753-769.

[Moore 66] R. E. Moore. *Interval Analysis.* Printice Hall, Englewood Cliffs, N.J., 1966.

[More 90] J. M. Moreau. *Hiérarchisation et facetisation de la représentation par segments d'un graphe planaire dans le cadre d'une arithmétique mixte.* Thèse de Doctorat en Informatique, Ecole Nationale Supérieure des Mines de Saint-Etienne, France, Oct. 1990.

[MP 78] D. E. Muller and F. P. Preparata. *Finding the Intersection of two Convex Polyhedra.* Theoretical Computer Science Pub., North-Holland, July 1978, pp. 217-236.

[Mull 89] J.M. Muller. *Arithmétique des Ordinateurs.* Ed. Masson, 1989.

[NAB 86] I. Navazo, D. Ayala, and P. Brunet. *A Geometric Modeler Based on the Exact Octtree Representation of Polyhedra.* Computer Graphics Forum 5, May 1986, pp. 91-104.

[NAT-90] B. Naylor, J. Amanatides, and W. Thibault. *Merging BSP Trees Yields Boolean Set Operations.* SIGGRAPH'90, ACM Computer Graphics, Aug. 1990, Vol. 24, No. 4, pp. 115-124.

[Nie 91] Zhigang Nie. *Utilisation des déformations pour la modélisation des solides de forme libre en sy,nthèse d'images.* Thèse de Doctorat en Informatique, Ecole Nationale Supérieure des Mines de Saint-Etienne, Oct. 1991.

[OTU 87] T. Ottmann, G. Thiemt, and C. Ullrich. *Numerical Stability of Geometric Algorithms.* Proc. of the 3rd. ACM Symp. on Computational Geometry, 1987, pp. 119-125.

[PBCF 93] A. Paoluzzi, F. Bernardini, C. Cattani, and V. Ferrucci. *Dimension-Independent Modeling with Simplicial Complexes.* ACM Trans. on Graphics 12, Jan. 1993, pp. 56-102.

[Peter 86] D. P. Peterson. *Boundary to Constructive Solid Geometry Mappings: A Focus on 2D Issues.* CAD Vol. 18, No. 1, Jan. / Feb. 1986, pp. 3-14.

[Pero 87] B. Péroche, J. Argence, D. Ghazanfarpour, et D. Michelucci. *La synthèse d'images.* Hermes 1987.

[PRS 89] A. Paoluzzi, M. Ramella, and A. Santarelli. *Booleán Algebra Over Linear Polyhedra.* CAD Vol. 21, No. 8, Oct. 1989, pp. 474-484.

[PS 85] F. P. Preparata and M. I. Shamos. *Computanional Geometry: An Introduction.* Springer-Verlag, New York, NY, 1985.

[PSB 91] R. M. Persiano, M. Salim, and L. P. Bueno. *Boundary Evaluation of CSG Solids by Simplicial Subdivision.* Proc. COMPUGRAPHICS'91, 1991, pp. 220-229.

[Requ 80] A. A. G. Requicha. *Representation for Rigid Solids: Theory, Methods and Systems.* ACM Computing surveys, Dec. 1980, Vol. 12, No. 4, pp. 437-464.

[Ross 86] J. R. Rossignac. *Constraints in Construtcive Solid Geometry.* Proc. of Workshop on Interactive 3D Graphics, Oct. 1986, Chapel Hill, ACM Press, pp. 93-110.

[Ross 90] J. R. Rossignac. *Issues on Feature-Based Editing and Interrogation of Solid Models.* Computer & Graphics (1990), Vol. 14, No. 2, pp. 149-172.

[Ross 91] J. R. Rossignac. *Through the Cracks of the Solid Modeling Milestone.* EUROGRAPHICS'91, State of The Art Report on Solid modeling, 1991, pp. 23-109.

[Roth 82] S. Roth. *Ray Casting for Modeling Solids.* Computer Graphics and Image Processing 18, 1982, pp. 109-144.

[RR 86] J. R. Rossignac and A. A. G. Requicha. *Depth-Buffering Display Techniques for Constructive Solid Geometry.* IEEE CG & A, Sept. 1986, pp. 29-39.

[RR 91] J. R. Rossignac and A. A. G. Requicha. *Constructive Non-Regularized Geometry.* CAD Vol. 23, No. 1, Jan. / Feb. 1991, pp. 21-32.

[RR 92] A. A. G. Requicha and J. R. Rossignac. *Solid Modeling and Beyond.* IEEE CG & A, Sept. 1992, pp. 31-44.

[RV 83] A. A. G. Requicha and H. B. Voelcker. *Solid Modeling: Current Status and Research Directions.* IEEE CG & A, Oct. 1983, pp. 25-37.

[RV 85] A. A. G. Requicha and H. B. Voelcker. *Boolean Operations in Solid Modeling: Boundary Evaluation and Merging Algorithms.* Proc. IEEE, Jan. 1985, Vol. 73, No. 1, pp. 30-44.

[RV 89] J. R. Rossignac and H. B. Voelcker. *Active Zones in CSG for Accelarating Boundary Evaluation, Redundancy Elimination, Interference Detection, and Shading Algorithms.* ACM Trans. on Graphics, Jan. 1989, Vol. 8, No. 1, pp. 51-87.

[SA 85] T. W. Sederberg and D. Anderson. *Steiner Surface Patches.* IEEE CG & A, 1985, Vol. 5, No. 1, pp. 23-36.

[Sega 90] M. Segal. *Using Tolerances to Guarantee Valid Polyhedral Modeling Results.* SIGGRAPH'90, Aug.1990, ACM Computer Graphics, Vol. 24, No. 4, pp. 105-114.

[SH 74] I. E. Sutherland and G. W. Hodgman. *Reentrant Polygon Clipping.* CACM, Vol. 17, pp. 32-42, 1974.

[SI 89] K. Sugihara and M. Iri. *A Solid Modelling System Free From Topological Inconsistency*. Journal of Information Processing, Vol. 12, No. 4, 1989, pp. 380-393.

[Sugi 89] K. Sugihara. *On Finite-Precision Representations of Geometric Objects*. Journal of Computer and Syst. Sciences, Vol. 39, No. 2, Oct. 1989, pp.236-247.

[SP 86] T. W. Sederberg and S. R. Parry. *Free-Form Deformation of Solid Geometric Models*. SIGGRAPH'86, Aug. 1986, ACM Computer Graphics, Vol. 20, No. 4, pp. 151-160.

[SS 85] M. Segal and C. H. Sequin. *Consistent Calculations for Solid Modeling*. Proc. of the First ACM Symp. on Computational Geometry, 1985, pp. 29-38.

[SS 88] M. Segal and C.H. Sequin. *Partitioning Polyhedral Objects into Non Intersecting Parts*. IEEE CG & A, Jan. 1988, pp. 53-67.

[SV 91] V. Shapiro and D. L. Vossler. *Construction and Optimization of CSG Representations*. ACM Trans. on Graphics, Jan. 1993, Vol. 12, No. 1, pp. 35-55.

[SV 93] V. Shapiro and D. L. Vossler. *Separation for Boundary to CSG Conversion*. CAD, Jan. / Feb. 1991, Vol. 23, No. 1, pp. 4-20.

[Szil 84] M. Szilvazi-Nagy. *An Algorithm for Determining the Intersection of Two Simple Polyhedra*. Computer Graphics Forum, March 1984, pp. 219-225.

[Tawf 91] M. S. Tawfik. *An Efficient Algorithm for CSG to B-Rep Conversion*. Proc. Solid Modeling Foundations and CAD/CAM Appl., June 1991, Texas, pp. 99-108.

[Tilo 80] R. B. Tilove. *Set Membership Classification: A Unified Approach to Geometric Intersection Problems*. IEEE Trans. on Computers, Oct. 1980, Vol. C-29, No. 10, pp. 874-884.

[Tilo 84] R. B. Tilove. *A Null-Object Detection Algotithm for Constructive Solid Geometry*. Comm. ACM **27**, July 1984, pp. 684-694.

[TN 87] W. C. Thibault and B. F. Naylor. *Set Operations on Polyhedra Using Binary Space Partitioning Trees*. SIGGRAPH'87, July 1987, ACM Computer Graphics, Vol. 21, No. 4, pp. 153-162.

[TSUC 86] H. Toriya, T. Satoh, K. Ueda, and H. Chiyokura. *UNDO and REDO Operations for Solid Modeling*. IEEE&CGA, April 1986, pp. 35-42.

[Verr 90] A. Verroust. *Etude de problèmes liés à la définition, la visualisation et l'animation d'objets complexes en Informaique graphique*. Thèse de Doctorat d'Etat, Université de Paris-Sud, Centre Orsay, Déc. 1990.

[Walk 89] M. Walker. *Boolean Operations with Enriched Octree Structures*. Computer & Graphics, 1989, Vol. 13, No. 4, pp. 487-495.

[WBLR 85] F. Winkler, B. Buchberger, F. Lichtenberger, and H. Rolletschek. *An Algorithm for Constructing Canonical Bases of Polynomial Ideals*. ACM Trans. on Math. Software, Vol. 11, No. 1, March 1985, pp. 66-78.

[Weil 85] K. Weiler. *Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments*. IEEE CG & A, Jan. 1985, pp. 21-40.

[Weil 86] K. Weiler. *Topological Structures for Geometric Modeling*. Ph.D. Dissertation, Rensselaer Polytechnic Institute, Troy, NY, 1986.

[WLLLG 80] M. A. Wesley, T. Lozano-Perez, L. I. Lieberman, M. A. Lavin, and D. D. Grossman. *A geometric modeling system for Automated Mechanical Assembly*. IBM J. Res. & Develop. 24(1), pp. 64-74, Jan. 1980.

[Wu 92] Shin-Ting Wu. *Non-manifold Data Models: Implementational Issues*. Actes MICAD'92, Paris, Fevrier 1992, Vol. 1, Ed. Hermes, pp. 37-56.

[Yap 88] C. K. Yap. *A Geometric Consistency Theorem for a Symbolic Perturbation Scheme*. Proc. of the 4th ACM Symp. on Computational Geometry, 1988, pp. 134-142.

[YT 84] F. Yamaguchi and T. Tokieda. *A Unified Algorithm for Boolean Shape Operations*. IEEE CG & A, June 1984, pp. 24-37.

Résumé :

Les progrès enregistrés en modélisation solide ont beaucoup contribué à l'essor des diverses applications de la CAO/FAO, de la robotique et de la synthèse d'images. Les systèmes de modélisation solide contemporains combinent souvent la représentation par *arbre de construction* et la représentation par *frontière* afin de mieux répondre aux besoins des applications. Dans cette thèse nous proposons une nouvelle méthode de calcul de la frontière d'un objet polyédrique décrit par un arbre de construction, qui traite uniformément les nombreux cas particuliers et qui résout le problème crucial des imprécisions numériques inhérentes à l'arithmétique flottante. Une implantation utilisant une arithmétique rationnelle optimisée est présentée ainsi que des résultats de tests.

Mots clés :

Modélisation solide, Représentation par frontière, Opérations booléennes, Imprécision numérique, Cohérence topologique, Arithmétique rationnelle, Arithmétique paresseuse.