Les représentations par les frontières: quelques constructions ; difficultés rencontrées

Dominique Michelucci

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

N° D'ordre 7 ID

THESE

présentée par

Dominique MICHELUCCI

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, Image, Intelligence artificielle et algorithmique)

LES REPRESENTATIONS PAR LES FRONTIERES : QUELQUES CONSTRUCTIONS ; DIFFICULTES RENCONTREES

soutenue à SAINT-ETIENNE le 30 novembre 1987

composition du jury :

Monsieur	F. MARTINEZ	Président
Madame	Y. AHRONOVITZ	
Messieurs	C. PUECH	
	M. GANGNET	Examinateurs
	J.P. GOURE	
	B. PEROCHE	
	P. QUINTRAND	

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

N° D'ordre 7 ID

THESE

présentée par

Dominique MICHELUCCI

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, Image, Intelligence artificielle et algorithmique)

LES REPRESENTATIONS PAR LES FRONTIERES : QUELQUES CONSTRUCTIONS ; DIFFICULTES RENCONTREES

soutenue à SAINT-ETIENNE le 30 novembre 1987

composition du jury :

Monsieur	F. MARTINEZ	Président
Madame	Y. AHRONOVITZ	
Messieurs	C. PUECH	
	M. GANGNET	Examinateurs
	J.P. GOURE	
	B. PEROCHE	
	P. QUINTRAND	

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT ETIENNE

Directeur : M. Philippe SAINT RAYMOND
Directeur des Etudes et de la formation : M. Jean CHEVALIER
Directeur des Recherches : M. MUDRY
Secrétaire Général : M. J. Claude PIATEK

PROFESSEURS DE 1ère CATEGORIE

MM.	COINDE	Alexandre	Gestion
	FORMERY	Philippe	Mathématiques Appliquées
	GOUX	Claude	Métallurgie
	LE COZE	Jean	Matériaux
	LOWYS	Jean-Pierre	Physique
	MATHON	Albert	Gestion
	PERRIN	Michel	Géologie
	PEROCHE	Bernard	Informatique
	RIEU	Jean	Mécanique - Résistance des Matériaux
	SOUSTELLE	Michel	Chimie
	VERCHERY	Georges	Matériaux

PROFESSEURS DE 2ème CATEGORIE

MM.	LADET	Pierre	Entreprise et Travaux
	PLA	Jean Marie	Mathématiques
	TOUCHARD	Bernard	Physique Industrielle

DIRECTEUR DE RECHERCHE

M.	LESBATS	Pierre	Métallurgie

MAITRES DE RECHERCHE

MM.	BISCONDI	Michel	Métallurgie
	CONRAD	Francis	Informatique
	DAVOINE	Philippe	Géologie
	DRIVER	Julian	Matériaux
Mle	FOURDEUX	Angeline	Métallurgie
MM.	GIRARDOT	Jean Jacques	Informatique
	GUILHOT	Bernard	Chimie
	KOBYLANSKI	André	Métallurgie
	LALAUZE	René	Chimie
	LANCELOT	Francis	Chimie
	MONTHEILLET	Franck	Matériaux
	TRAN MINH	Canh	Chimie

PERSONNALITES HABILITEES A DIRIGER LES TRAVAUX DE RECHERCHE

MM.	COURNIL	Michel	Chimie
	MAGNIN	Thierry	Matériaux
	THOMAS	Gérard	Chimie

PROFESSEUR A L'U.E.R. DE SCIENCES DE SAINT ETIENNE

M.	VERGNAUD	Jean Marie	Chimie des Matériaux

REMERCIEMENTS

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

Monsieur Francis MARTINEZ, professeur à l'Institut National Polytechnique de GRENOBLE, rapporteur, qui me fait l'honneur de présider le jury ;

Monsieur Claude PUECH, directeur du Laboratoire d'Informatique de l'Ecole Normale Supérieure, rapporteur, dont la pertinence des remarques a beaucoup apporté à cette thèse ;

Madame Yollande AHRONOVITZ, maître de conférence à l'Université de Saint Etienne, dont les conseils sur le dernier chapitre m'ont été très utiles ;

Monsieur Michel GANGNET, qui m'a permis d'entamer cette thèse, m'a initié à l'informatique et à la recherche, et a continué de suivre mes travaux après son départ de l'Ecole des Mines de Saint Etienne ;

Monsieur Jean Pierre GOURE, directeur du Laboratoire de Traitement du Signal et de l'Information à l'Université de Saint Etienne, qui a bien voulu se joindre au jury ;

Monsieur Bernard PEROCHE, directeur du Département Informatique Appliquée de l'Ecole des Mines de Saint Etienne, qui a consacré énormément de temps à encadrer ce travail, et a sacrifié stoïquement quelques soirées et week-ends pour corriger ce rapport ;

Monsieur Paul QUINTRAND, directeur du Groupe pour l'Application des Méthodes Scientifiques à l'Architecture et à l'Urbanisme, que je remercie pour sa patience, et qui m'a fait l'amitié de participer à ce jury.

Je remercie Messieurs Jean Claude HERVE, Jean Manuel VAN THONG et Michel GANGNET, dont les commentaires sur certains de mes logiciels m'ont indéniablement enrichi ; je remercie Monsieur Franck CHOPIN, pour son obstination qui a produit les perspectives au trait du chapitre II.

Je remercie aussi pour leur amitié et leur support moral (ils m'ont supporté en effet) les membres du Département Informatique de l'Ecole des Mines de Saint Etienne. Plus particulièrement : Jean Pierre TAVERNIER pour la maintenance du matériel informatique et quelques sauvegardes qui furent les bienvenues ; Jean Jacques GIRARDOT pour son logiciel de formatage de texte : Groff[1], et pour ses leçons d'orthografe (Entre parenthèses, je lui suis très reconnaissant de ses cours et de son livre sur Lisp) ; et tous les membres, ou anciens membres, de l'équipe

"Communications visuelles" : Jacqueline ARGENCE, Michel BEIGBEDER, Sabine COQUILLART, Gilles FERTEY, Djamchid GHAZANFARPOUR KHOLENDJANY, Jean Claude MOISSINAC, Jean Michel MOREAU, Didier TALLOT et Jacqueline TALLOT.

Je remercie enfin les membres du service de reprographie de l'Ecole des Mines de Saint Etienne.

1 GROFF : "Girardot's Ridiculous Output File Formater"

LES REPRESENTATIONS PAR LES FRONTIERES : QUELQUES CONSTRUCTIONS ; LES DIFFICULTES RENCONTREES .

INTRODUCTION

Cette thèse traite de la construction de quelques représentations par les frontières, ou BREP (pour "*Boundary REPresentation*"), et des difficultés rencontrées. Aussi cette introduction rappelle-t-elle très brièvement ce que sont les BREP; on se reportera à [PERO 87] pour un exposé plus détaillé et pour d'autres références.

La synthèse d'images et la CAO utilisent diverses modèlisations des solides pour les visualiser et calculer leurs propriétés géométriques et mécaniques : masse, aire de l'enveloppe, centre de gravité, axes principaux d'inertie, etc.

Plusieurs modèles informatiques des solides ont été proposés :

La **représentation filaire**, la première utilisée, permet surtout d'automatiser le travail traditionnel sur table à dessins. Avec cette représentation, un objet est seulement connu par la liste de ses arêtes et les coordonnées de ses sommets; cette représentation permet des sorties graphiques au trait, sans élimination des parties cachées. Aujourd'hui, cette représentation n'a plus guère qu'une importance historique, encore qu'elle soit parfois utilisée pour la production en temps réel de séquences d'animation lors de la mise au point ("*line-test*"). Cette représentation est fortement ambiguë : des objets différents peuvent avoir des représentations filaires identiques. Le plein n'est pas distingué du vide.

La modélisation suivante, historiquement parlant, est la **représentation surfacique**; elle décrit le contour de l'objet par une liste de ses surfaces; ces surfaces peuvent être des faces planes, des surfaces de révolution, des quadriques [LEVI 79], des surfaces libres, ou "*patches*" comme les surfaces de COONS, les surfaces de BEZIER, les Bsplines rationnelles [BEZI 86] [CAST 85]. Cette représentation permet la production d'images couleur ("*shaded*"), et l'usinage des surfaces. Cependant, sa cohérence doit être gérée manuellement par l'opérateur : toute liste de surfaces ne délimite pas forcément un intérieur et un extérieur; cette modélisation ne fait donc pas partie des représentations dites volumiques, contrairement aux

trois représentations suivantes :

Les **BREP**, ou **représentations par les frontières**, sont le prolongement des représentations surfaciques. Les BREP décrivent explicitement les contours de l'objet, et, contrairement aux représentations surfaciques, assurent la cohérence du contour et de l'objet. Une des premières BREP est la "*Winged Edge Structure*" introduite par BAUMGART [BAUM 75] en infographie, pour représenter les polyèdres. Les BREP ont toutes les fonctionnalités des représentations surfaciques; elles permettent de plus un calcul précis des caractéristiques géométriques ou mécaniques des objets décrits.

On distingue en général deux types d'informations dans les BREP : les informations de nature géométrique : coodonnées des sommets, représentation mathématique de la forme des arêtes ou des surfaces...; il s'agit en bref d'une représentation surfacique, vérifiant certaines conditions d'intégrité. Et les informations de nature topologique : les diverses relations d'incidence entre les surfaces et les arêtes, entre les arêtes et les sommets de l'enveloppe du solide.

Une autre distinction classique s'attache au degré des surfaces admises dans l'enveloppe des solides :

Le cas le plus simple est celui où les surfaces du contour sont forcément des faces planes. On remarquera que tout solide peut être approximé d'aussi près que souhaité par un polyèdre, d'où le nom de "BREP approximative". Quand toutes ces faces sont convexes, voire triangulaires, on parle parfois de "*tessellations*"; des architectures matérielles spécialisées peuvent manipuler efficacement ce type de données [YAMA 84].

Depuis le travail de LEVIN [LEVI 79] [LEVI 80], certaines BREP admettent les surfaces quadriques dans le contour des objets [MILL 86].

Enfin, certaines BREP acceptent des "*patches*", assez souvent des carreaux de surfaces bicubiques polynômiales ou rationnelles. Il est nécessaire de savoir calculer et modéliser l'intersection entre deux "*patches*"; ce problème est, semble-t-il, toujours résolu de façon approximative, par exemple par une facettisation des "*patches*" : [CARL 82], [CASA 87]; le chapitre IV propose une alternative (certes pas forcément efficace...), qui résoud ce problème de façon exacte.

L'emploi des BREP présente quelques difficultés :

Première difficulté : comment assurer la cohérence de la représentation ? Ce problème est bien connu, et réglé par exemple par l'emploi des opérateurs d'EULER [MANT 83] pour le cas des polyèdres; cette thèse proposera une solution nouvelle à ce problème, dans le chapitre III.

Deuxième inconvénient des BREP : il n'existe pas de moyen interactif évident pour saisir les BREP d'objets complexes. D'ailleurs les BREP sont assez souvent obtenues à partir d'un autre type de modélisation, le plus souvent la modélisation CSG ("*Constructive Solid Geometry*") dont le principe est rappelé ultérieurement.

Troisième difficulté, déjà signalée : il est nécessaire de savoir calculer l'intersection entre deux surfaces de l'enveloppe; dans le cas des "*patches*", il semble que les solutions utilisées soient toutes approximatives.

Un quatrième problème, amplement discuté dans cette thèse, est lié aux imprécisions, et aux incohérences qu'elles peuvent causer. Ce problème est connu, mais sa difficulté et son importance sont, me semble-t-il, sous-estimées : cette opinion est partagée par [TAKA 86] et par [GANG 87]. Les conséquences de l'imprécision sur les BREP et sur les algorithmes de construction des BREP sont mises en évidence par deux exemples détaillés dans le chapitre 1.

La **représentation CSG** ("*Constructive Solid Geometry*") ne décrit plus l'objet par ses frontières, mais par un processus dont l'objet est le résultat. Classiquement, un objet est représenté par un arbre, dont les noeuds portent des opérations booléennes (intersection, union, différence) ou des transformations affines, et dont les feuilles portent des solides élémentaires; dans les définitions premières, il s'agit essentiellement des quadriques.

Plusieurs généralisations ont été proposées; BARR [BARR 84], et SEDERBERG et PARRY [SEDE 86] ont proposé de remplacer les transformations affines par des opérations plus générales, des "déformations" de R^3; il est aussi possible d'admettre des solides élémentaires plus complexes que les quadriques aux feuilles, comme des tores, ou des "*patches*" de STEINER dont la forme implicite, de degré bas, décrit bien un solide. Enfin, il est possible de prendre en compte les chanfreins, congés et biseaux, et les surfaces de mélange [MIDD 85]. L'arbre CSG semble donc bien adapté à la description des objets de la mécanique ou du design industriel. Sa définition se prête à une saisie interactive, et des langages de modélisation emploient l'arbre CSG.

Contrairement aux BREP, un arbre CSG syntaxiquement correct est toujours cohérent (Bien sûr, il ne décrit peut-être pas l'objet souhaité par l'opérateur mais c'est là un autre problème).

La représentation CSG n'effectue pas les opérations (transformations géométriques et opérations booléennes) : elle ne fait que les décrire; en ce sens, la représentation CSG est une représentation implicite; par exemple, elle ne permet pas de savoir directement si l'objet décrit est ou non le solide vide, s'il est connexe... Une BREP le permet.

Visualiser ou calculer les caractéristiques mécaniques de l'objet décrit suppose d'effectuer les opérations portées par l'arbre CSG, au moins partiellement.

Une procédure récursive simple et efficace détermine si un point donné se trouve à l'intérieur, à l'extérieur ou sur le contour (à un epsilon près) de l'objet décrit; de même, il est facile de calculer la liste des intervalles d'intersection entre une droite et l'objet décrit par l'arbre CSG. Le lancer de rayons emploie cette propriété pour visualiser l'objet [ROTH 81]. Il est aussi possible d'estimer le volume de l'objet (et les autres caractéristiques géométriques) par une méthode de ce type : on intègre la longueur des intervalles sur un faisceau de droites parallèles. Vu le grand nombre de droites nécessaires pour obtenir une bonne précision, cette approche exige une puissance de calculs non négligeable, selon les critères actuels.

Des applications de la CAO nécessitent une connaissance explicite des surfaces de contour de l'objet décrit par un arbre CSG. D'où le problème classique, appelé "*Boundary Evaluation*", du passage d'un arbre CSG à une BREP équivalente (le plus souvent une BREP polyèdrique approximative). Même dans le cas de polyèdres, ce problème est plus difficile qu'il ne paraît; il est abordé aux chapitres III et IV.

Le troisième modèle est celui de l'**énumération spatiale**. Ce modèle renonce à la continuité de l'espace, qui est supposé discret : l'espace est seulement une trame cubique 3D, l'analogue d'une matrice de pixels. Dans la définition première, un cube élémentaire, ou voxel, ne peut prendre que deux valeurs : plein, ou vide. L'espace est donc un tableau 3D de bits; ce tableau peut informatiquement être codé et compacté sous forme d'octree, ou de bintree [SAME 84]. Avec ce codage, la place mémoire nécessaire est proportionnelle à l'aire de la surface de l'objet : la complexité de l'objet (mesuré par exemple par le nombre et le degré des surfaces du contour) n'importe pas. Tout solide peut être approximé d'aussi près que l'on veut par cette représentation.

La représentation par un tableau de booléens permet d'effectuer trivialement un calcul approché des caractéristiques géométriques et mécaniques de l'objet représenté, les transformations géométriques et les opérations booléennes. Le codage sous forme d'octree ou de bintree ne complique pas fondamentalement ces opérations. Ces procédures prennent un temps proportionnel à la taille des octrees ou bintrees, ou des tableaux de voxels, selon le codage.

Bien sûr, comme les arbres CSG, un tableau 3D de bits est toujours cohérent. De même pour l'octree qui code ce tableau de booléens. Les seuls problèmes posés sont, fondamentalement, des problèmes d'aliassage, perceptibles quand sont effectuées des rotations ou des affinités.

L'objet représenté ainsi peut être visualisé de deux façons; d'avant en arrière, par l'algorithme du lancer de rayons : les rayons sont suivis de cellule en cellule jusqu'à l'impact; et d'arrière en avant : les voxels sont affichés successivement, d'arrière en avant, par l'algorithme du peintre.

Cette représentation, la plus simple, est aussi la dernière qui soit apparue, pour des raisons évidentes d'encombrement mémoire et de temps machine. Des implantations matérielles efficaces sont possibles.

Cette représentation a aussi ses inconvénients : elle est approximative, ce qui peut être un défaut; elle perd la continuité et la normale des surfaces de contour, nécessaires pour l'usinage des surfaces ou le rendu graphique : ces informations doivent être reconstituées approximativement par une étude locale du voisinage des voxels. Cette représentation perd aussi les informations de forme : comment reconnaître par algorithme qu'un tableau de voxels ou un octree "représente" un cylindre ? C'est là un problème de reconnaissance des formes, mal résolu à l'heure actuelle.

Chaque représentation volumique a ses avantages et ses inconvénients, et les modeleurs de la CAO emploient en règle générale plusieurs types de modélisation, ce qui pose le problème des transitions et des compatibilités entre les différentes représentations.

Ces brefs rappels situent les BREP relativement aux autres représentations des solides. Pour simplifier, cette introduction ignore les nombreuses modélisations hybrides : [AYAL 85] [CHAZ 80] [GLAS 84] [KUNI 85] [SAME 86]...

PLAN DE LA THESE

Cette thèse discute la construction de quelques représentations par les frontières, ou BREP (pour "*Boundary REPresentation*"), et les difficultés rencontrées.

Chapitre I :

Le premier chapitre présente la construction d'une BREP, dite carte planaire, qui décrit les scènes 2D polygonales. Ce chapitre met en évidence les méfaits de l'imprécision numérique : un exemple dû à [GANG 87] montre que certaines configurations d'arêtes ne peuvent être représentées de façon cohérente avec des nombres flottants; d'autres exemples détaillent les conséquences de l'imprécision sur le déroulement d'un algorithme dû à BENTLEY-OTTMAN [BENT 79]; ces conséquences sont comparées avec celles de l'imprécision sur d'autres algorithmes classiques de la synthèse d'images. Une solution est proposée : l'arithmétique rationnelle, qui permet d'éviter toute erreur de calcul. Le chapitre I propose en outre deux algorithmes nouveaux : détermination de la topologie d'une carte planaire, et affichage antialiassé de cartes planaires par sur-échantillonnage local.

Chapitre II :

Le deuxième chapitre décrit la construction d'une BREP représentant une image en perspective d'une scène polyèdrique décrite par un arbre de construction ("*CSG tree*") : les algorithmes publiés précédemment résolvaient le problème à la résolution d'affichage. Des problèmes d'imprécision 3D surgissent, et des solutions sont proposées et évaluées. Une autre difficulté jusqu'ici méconnue est mise en évidence : elle est liée à la projection sur un plan des arêtes de la scène : trop souvent, le nombre de points d'intersection entre les arêtes projetées croît comme le carré du nombre d'arêtes, d'après un constat empirique; ce qui risque fort de nuire à l'efficacité des algorithmes travaillant en projection.

Chapitre III :

Le troisième chapitre traite des opérations booléennes sur les polyèdres : en plus des difficultés dues à l'imprécision, le foisonnement des cas particuliers pose problème, comme l'a bien perçu MARTIN [MART 87]. Ce chapitre propose une solution nouvelle, "naïve", mais qui résoud simplement tous les cas particuliers. Une autre difficulté est le contrôle de la cohérence topologique des BREP produites; elle est en général surmontée par l'emploi des opérateurs d'EULER [MANT 83], qui sont d'une manipulation lourde et délicate. Le chapitre III montre qu'ils sont inutiles : résoudre les problèmes d'imprécision numérique règle du même coup les problèmes d'incohérence. Cette liaison entre incohérence topologique et imprécision numérique est un constat relativement nouveau : il n'a été effectué auparavant que par TAKALA [TAKA 86] et GANGNET [GANG 87] (non publié). TAKALA propose une solution tout à fait différente de la nôtre : il renonce à la cohérence topologique des BREP. La dernière difficulté est la minimisation du nombre nécessaire de tests d'intersection entre les faces. Une optimisation nouvelle est proposée; elle est inspirée directement du "*Multi-Dimensionnal Searching*"; incidemment, ce chapitre démontre aussi une borne inférieure pour le "*Static Range Searching Problem*".

Chapitre IV :

Le chapitre IV est davantage prospectif. Il montre qu'il est théoriquement possible d'obtenir une BREP exacte du solide décrit par un arbre de construction dont les feuilles sont des inéquations algébriques de degré quelconque : un algorithme sert de preuve constructive. Les techniques des chapitres précédents permettent de réduire ce problème à une liste de sous problèmes mathématiques déjà résolus (pas forcément de façon efficace...) par le calcul algébrique symbolique; les méfaits de l'imprécision numérique peuvent être prévenus par l'emploi d'une arithmétique algébrique. Les cas particuliers sont cependant ignorés.

Chapitre 1

LES CARTES PLANAIRES : CONSTRUCTION, PROBLEMES D'IMPRECISION, AFFICHAGE.

1 INTRODUCTION

Le terme de "carte planaire" est emprunté à la théorie des graphes [BERG 70] : S étant un ensemble de sommets et A une partie de SxS, le graphe G=(S,A) est dit planaire s'il est possible d'en trouver une représentation dans le plan telle que les images de deux arêtes distinctes n'aient d'autres points communs que des représentations des sommets; une carte planaire est une représentation plane d'un graphe planaire. Les cartes planaires sont aussi appelées "Planar Straight-Line Graphs", ou PSLG [MULL 78]. Il existe aussi des cartes planaires dont les arêtes sont des arcs de courbe, mais ce chapitre ne traite que des PSLG.

La section 2 présente d'abord une structure de données décrivant les cartes planaires. Par abus de langage, cette structure de données est aussi appelée "carte planaire". Elle est constituée de deux parties : une carte planaire locale, ou CP-locale, qui s'inspire de la *"Winged-Edge Structure"* de BAUMGART [BAUM 75] et de la *"Doubly-Connected-Edge List"* de MULLER et PREPARATA [MULL 78]; et une carte planaire globale, ou CP-globale.

Puis la section 3 expose la construction de cette structure de données, à partir d'un ensemble non structuré de N segments de droite $[(x_i\ y_i)\ (x_i'\ y_i')]$ éventuellement intersectants. Cette construction s'effectue en trois étapes; la première est une simple initialisation. La deuxième étape utilise l'algorithme de BENTLEY-OTTMAN [BENT 79] pour déterminer les points d'intersection entre les segments; la troisième étape construit la CP-globale par un parcours de la CP-locale : cet algorithme, dû à GANGNET et MICHELUCCI, n'a été publié que dans [GANG 84].

La programmation de l'algorithme de BENTLEY-OTTMAN a révélé quelques difficultés inattendues, liées à l'imprécision numérique; dans la section 4, deux contre-exemples précis démontrent que, au moins pour certaines configurations de segments, l'imprécision numérique perturbe gravement la méthode de BENTLEY-OTTMAN : elle peut oublier des points d'intersection, voire s'enliser

dans une situation imprévue; il semble que les conséquences de l'imprécision numérique sur la méthode de BENTLEY-OTTMAN aient été sous-estimées jusqu'ici. Le premier contre-exemple montre, en outre, que les coordonnées des sommets de certaines cartes planaires ne peuvent pas être représentées par des nombres flottants de façon cohérente; à ma connaissance, ce phénomène est largement méconnu, sauf par [GANG 87] et peut-être [TAKA 86]. Une solution à ces problèmes est proposée, et discutée.

Ce chapitre traite ensuite en section 5 de l'affichage des cartes planaires sur des mémoires de trame; la structure de données permet des affichages efficaces. Elle permet aussi d'éviter les phénomènes d'aliassage inhérents à la résolution limitée de ce type d'affichage. Deux méthodes de filtrage sont présentées : un suréchantillonnage global, qui est une application naturelle sur les cartes planaires d'une méthode classique de filtrage; et un suréchantillonnage local. Cette dernière méthode n'a fait l'objet d'aucune publication antérieure.

Enfin, ce chapitre se conclut en 6 sur quelques problèmes ouverts, comme la construction incrémentale de cartes planaires.

Tous les algorithmes et structures de données discutés ici ont fait l'objet de programmes en langage PASCAL ou C en 1984 ou 1985. Le logiciel a été utilisé pour des applications diverses; en effet, les cartes planaires décrivent explicitement la métrique et la topologie de toutes les scènes 2D :

MOISSINAC a utilisé les cartes planaires pour des logiciels d'animation [MOIS 84]; COQUILLART les a utilisées pour une optimisation du lancer de rayons [COQU 84]; la société TANGRAM a développé un logiciel d'illustration : CADIX, où les dessins sont modélisés par des cartes planaires [GANG 87]. ROMMEL a modélisé les cartes géologiques par des cartes planaires [ROMM 87]. Les cartes planaires, ou des variantes, sont aussi utilisées dans les chapitres suivants. Initialement, le logiciel de cartes planaires ne servait pourtant qu'à des instrumentations diverses (recalages, désignations, affectations) sur des croquis d'architecture, saisis à main levée; ce travail est présenté en annexe; il a été réalisé en collaboration avec GANGNET, et a fait l'objet d'un article [MICH 84] et d'un rapport interne [GANG 84], dont s'inspire partiellement ce chapitre.

2 LA STRUCTURE DE DONNES DE CARTE PLANAIRE

Le but est de construire une structure de données : CP, décrivant une carte planaire. Cette construction s'effectue en trois étapes : dans un premier temps, une structure de données locale à chaque sommet initialement connu est d'abord construite; cette structure, CP-locale, est une variante de la structure WE ("*Winged-Edge Structure*") introduite par BAUMGART [BAUM 75] en infographie, et de la DCEL ("*Doubly-Connected-Edge List*") de MULLER et PREPARATA [MULL 78]. La

CP-locale contient les informations relatives à chaque sommet, et aux arêtes incidentes. Au début, la CP-locale n'est pas cohérente, car elle ignore les intersections entre les arêtes. Aussi, dans un deuxième temps, les points d'intersection entre les segments sont-ils déterminés, et la CP-locale est mise à jour au fur et à mesure : addition de sommets, division des segments intersectés. Enfin, dans un troisième temps, un parcours de la CP-locale crée la deuxième partie de la structure, dite CP-globale; la CP-globale indique explicitement pour chaque arête à quels contours elle appartient, et elle rend compte des adjacences et des inclusions entre les divers contours de la carte planaire.

2.1 CP-LOCALE

Un ordre lexicographique sur les points du plan est d'abord défini : l'xy-ordre, qui classe les sommets par abscisses croissantes; à abscisses égales, les sommets sont classés par ordonnées croissantes. L'xy-ordre est total; les sommets de la carte appartiennent à une liste doublement chaînée, représentant l'xy-ordre. Il permet de différencier les deux sommets d'une même arête : le plus petit, qui se trouve "à gauche", et le plus grand qui se trouve "à droite". L'xy-ordre résoud simplement le cas particulier des arêtes verticales : tout se passe comme si leur point le plus haut se trouvait à droite du point le plus bas : toutes les arêtes sont obliques.

Une arête "naît" en S quand S est son premier sommet; elle "meurt" en S quand S est son deuxième sommet.

Une arête est d'abord décrite par un triplet (a,b,c), $ax + by + c = 0$ étant l'équation de la droite support de l'arête. On reviendra ultérieurement sur la représentation du triplet (a,b,c). Une arête est aussi décrite par deux "brins" (intuitivement : deux "bouts"). Le premier brin a pour sommet le premier sommet de l'arête; le deuxième brin a pour sommet le deuxième sommet de l'arête. Cette distinction entre un brin et une adresse de sommet est nécessaire, car d'autres données sont attachées à un brin.

Il est possible de définir, sur les brins de même sommet S, un ordre total; tout brin, ou plutôt la demi-droite qui le porte, fait avec la verticale descendant de S un angle $\alpha \in]0, 2\pi]$; les brins sont ordonnés par leur angle; cet ordre est total, et est appelé α-ordre.

Fondamentalement, la CP-locale représente l'α-ordre, et ce de la façon suivante : sur chaque brin est noté le brin suivant selon l'α-ordre; ce brin est aussi appelé brin voisin; de plus le dernier brin d'un sommet pointe vers le premier brin du même sommet. Dans une implémentation de type PASCAL, où toute arête est un tableau [1..2] de brins, l'adresse d'un brin est en fait l'adresse de son arête, plus un indice, un ou deux. L'α-ordre est donc représenté par une liste bouclée de brins. Le nombre d'arêtes incidentes en un sommet n'est bien sûr pas limité, mais fini.

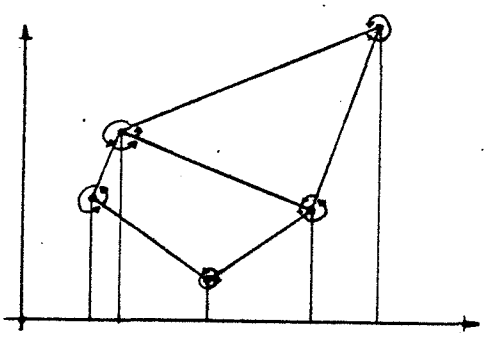


Fig. 1 : Une carte planaire locale

La CP-locale permet de retrouver le sommet de tout brin; elle permet aussi de retrouver les brins de chaque sommet : sur chaque sommet, est notée l'adresse de son premier brin selon l'α-ordre; or ce brin est la tête de la liste de l'α-ordre. De plus, les enregistrements de type sommet contiennent aussi les coordonnées cartésiennes du point. La représentation de ces coordonnées est précisée plus loin.

La CP-locale permet de tourner autour d'un sommet; quand elle est cohérente, c'est à dire quand aucune intersection n'est ignorée, la CP-locale permet aussi de parcourir les contours. Soit un brin; la CP-locale fournit le brin voisin; si on prend non pas ce brin voisin, mais l'autre brin de la même arête, et si on réitère cette opération, on va parcourir un contour par un algorithme main-gauche : on suit le même chemin qu'un petit bonhomme virtuel qui considère les arêtes comme des murs, et avance en posant sa main gauche sur les arêtes. Ces chemins ou contours sont appelés "bords" de la carte planaire. Dans la CP-locale, les bords ne sont représentés qu'implicitement, par un parcours dans la structure de données; ils sont décrits explicitement dans la CP-globale.

Parcourir un bord se fait par une procédure duale de celle qui permet de tourner autour d'un sommet; la structure de données présente en effet des propriétés de dualité analogues à celles des graphes planaires.

2.2 INITIALISATION DE LA CP-LOCALE

La première étape de construction d'une CP initialise la CP-locale, à partir des N segments $[(x_i\ y_i)\ (x'_i\ y'_i)]$. L'initialisation ne tient pas compte des éventuelles intersections entre les N arêtes. Elle peut se faire de deux façons : une première méthode utilise une table de hachage pour ne créer qu'une fois chaque sommet, et doit classer les brins incidents en un même sommet; elle est donc en O(NlogD), D étant le degré maximum des sommets. Mais, de toutes façons, l'étape suivante de calcul des intersections doit bien prendre en compte le cas particulier des points confondus : en effet un sommet peut être confondu avec un point d'intersection, initialement inconnu; une initialisation plus brutale est donc possible : elle crée deux

sommets et une arête pour chaque segment initial; un sommet est créé plusieurs fois, quand plusieurs arêtes sont incidentes en ce point; cette initialisation est en $\Theta(N)$. Ces deux méthodes sont d'une programmation triviale. Expérimentalement, elles se sont avérées sans incidence sur les performances globales de l'algorithme construisant la CP.

2.3 CP-GLOBALE

Certains bords définis en 2.1 sont parcourus dans le sens trigonométrique direct : le petit bonhomme tourne autour d'un contour; ces bords sont dits externes. D'autres bords sont parcourus dans le sens trigonométrique indirect : le petit bonhomme tourne à l'intérieur d'un contour; ces bords sont dits internes. Nous verrons ultérieurement comment l'algorithme peut distinguer les bords internes des bords externes.

Tout bord est une suite de brins, et la CP-locale permet de les parcourir. Si la CP-locale permet de retrouver le sommet de tout brin, elle ne permet pas d'en retrouver le bord. Pour combler cette lacune, on ajoute aux brins l'adresse de leur bord. Cette information est considérée comme faisant partie de la CP-globale, car elle ne peut être établie que lorsque la CP-locale est parfaitement connue, et cohérente.

Comment représenter un bord ? Pour parcourir ses brins, il suffit de pouvoir en retrouver un et la CP-locale permet le parcours. A tout bord on attache donc son premier brin : c'est celui qui a le sommet le plus petit selon l'xy-ordre. Ce sommet est le sommet de naissance du bord, mais plusieurs brins de ce bord peuvent y naître; dans ce cas, le premier brin est le plus petit des brins du bord, selon l'α-ordre.

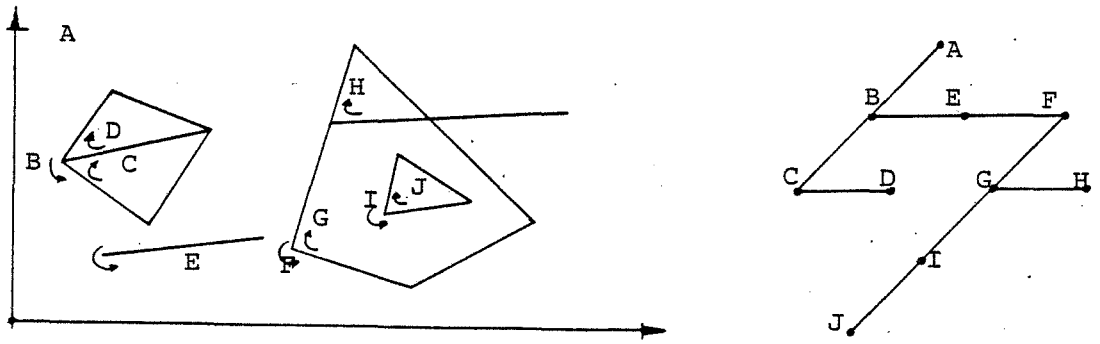


Fig. 2 : Une carte planaire et son arbre des bords

Il faut aussi représenter l'ordre naturel induit par les inclusions entre les bords. Remarquons qu'un bord interne est forcément contenu dans un bord externe; un bord externe, s'il est contenu dans un autre bord, est forcément contenu dans un bord interne. Par commodité, on considère qu'un bord externe est toujours au moins

contenu dans un bord interne virtuel, sans aucun brin : il peut être assimilé à la droite de l'infini du plan.

Une arête pendante est une arête dont les deux bords sont égaux; un bord ne contenant que des arêtes pendantes est un bord externe et ne peut contenir aucun bord.

L'inclusion entre les bords est représentée par un arbre généalogique binarisé : chaque bord pointe sur son bord père, sur son bord fils ainé, et sur son bord frère cadet. Le nombre de bords fils peut donc être quelconque. Il est bienvenu de définir un ordre sur les bords frères, inclus dans le même bord père. Cet ordre est l'xy-ordre des sommets de naissance des bords frères; dans le cas où deux bords frères ont le même sommet de naissance, ils sont ordonnés sur l'α-ordre de leurs brins de naissance..., ou plutôt sur l'α-ordre du brin voisin de leur brin de naissance. Cette subtilité est sans conséquence sur l'ordre des bords frères; mais elle assure que tout bord externe est immédiatement inférieur à son premier bord fils, s'il en a, et elle permet de généraliser l'ordre des bords frères à tous les bords. Avec cette définition, l'ordre des bords est leur ordre d'apparition dans l'algorithme de construction de la CP-globale à partir d'une CP-locale cohérente (ce qui explique en partie l'efficacité de cet algorithme).

Il reste à expliciter la nature d'une donnée supplémentaire attachée aux sommets et utilisée pour la localisation rapide d'une arête ou d'un bord. Cette donnée ne concerne que les sommets de naissance; elle est appelée l'arête "de butée", ou "butée", d'un sommet de naissance. Cette arête est la première arête rencontrée par la verticale descendant du sommet de naissance. C'est un sous produit de l'algorithme de BENTLEY-OTTMAN (voir plus loin).

```
type Côté = (1 , 2);
    Booléen = (vrai, faux);
    Sensbord = (interne, externe);
    Sommet = enregistrement
            { Sommet *x_précédent, *x_suivant;
              Arête  *incidente, *butée;
              Coté coté_incidente;
              Coordonnées xy;
              Booléen naissance;
            };
    Brin = enregistrement
            { Sommet *point;
              Arête *voisine;
              Coté coté_voisine;
              Bord *bord;
            };
    Arête = enregistrement
            { Equation abc;
              Brin demi[1..2];
            };
    Bord  = enregistrement
            { Sensbord sens;
              Bord *père, *fils_ainé, *frère;
              Arête *départ; /* coté du brin de départ = 1 */
            };
    /* Les types Equation et Coordonnées sont décrits ultérieurement */
```

Fig. 3 : Une description en pseudo langage C des cartes planaires

La figure 3 décrit une implantation en pseudo C de la structure de données de carte planaire. Il est possible d'attacher aux sommets, arêtes et bords d'autres informations relatives à l'application envisagée, par exemple une couleur ou un numéro de texture à chaque bord, une épaisseur à chaque arête, etc. Une liste de couples (attribut, valeur) est la solution la plus souple et la plus générale; on reconnait là le concept LISP des listes de propriétés. Ces listes de propriétés sont utilisées par les différents logiciels d'application des cartes planaires [MOIS 84] [COQU 84] [GANG 87] [ROMM 87]...

Les tailles des enregistrements des arêtes, des sommets, et des bords sont très comparables; la place mémoire occupée par la structure de données est manifestement proportionnelle à la somme du nombre final de sommets (y compris donc les points d'intersection), du nombre final d'arêtes, et du nombre final de bords. Vue la relation d'EULER, l'espace mémoire utilisé est proportionnel au nombre de sommets (pour la démonstration, considérez un graphe planaire totalement triangulé).

3 CONSTRUCTION DE LA CARTE PLANAIRE

La structure CP a été décrite. Sa construction se fait en trois étapes : l'initialisation, déjà traitée; la recherche des points d'intersection entre les segments initiaux, qui permet d'obtenir une CP-locale cohérente; et enfin la construction de la CP-globale. Ces deux dernières étapes sont maintenant présentées :

3.1 RECHERCHE DES POINTS D'INTERSECTION

3.1.1 Algorithme naïf

Pour trouver les points d'intersection entre les N arêtes, la méthode la plus simple teste l'intersection de toutes les paires d'arêtes. Cette méthode effectue donc N(N-1)/2 tests, et est en $O(N^2)$. Le calcul de l'intersection entre deux arêtes peut être accéléré par un test préliminaire de superposition des boites englobantes des arêtes, mais cette optimisation classique n'améliore en rien les performances théoriques de cet algorithme.

Une autre optimisation de cet algorithme ne compare que les segments superposés en projection sur un axe, Ox par exemple; pour ce faire, les N segments sont d'abord triés sur leur plus petite abscisse; l'ordre de deux segments de même abscisse gauche est indifférent. Soit segment[1..N], le tableau des segments ainsi trié; le k ième segment a pour sommet gauche le point (segment[k].x segment[k].y) et pour sommet droit le point (segment[k].x' segment[k].y'). La méthode procède alors comme suit :

```
pour i de 1 à N-1 faire
{  j ← i+1;
   /* segment[i].x ≤ segment[j].x */
   tant que (j≤N et segment[j].x ≤ segment[i].x')
           {   comparer-les-segments i et j;
               j ← j+1;
           }
}
```

Cette optimisation améliore beaucoup les performances pratiques de l'algorithme; celui-ci est en O(NlogN+T), T étant le nombre de superpositions de segments sur Ox; $T=O(N^2)$. Par la suite ces deux méthodes sont confondues et appelées : algorithme naïf.

3.1.2 Algorithme de BENTLEY-OTTMAN

Les points d'intersection entre les arêtes sont déterminés efficacement par l'algorithme de BENTLEY et OTTMAN [BENT 79]. Dans le cas général, pour chaque nouveau point d'intersection détecté, un nouveau sommet est créé, et les deux arêtes intersectantes sont coupées en deux. La méthode de BENTLEY et OTTMAN exploite et gère deux ordres : l'xy-ordre, déjà défini, et l'y-ordre.

L'y-ordre est défini sur un sous ensemble des arêtes, dites "actives", qui coupent une droite parallèle à l'axe des y; cette droite est indifféremment appelée "droite de balayage", ou "barre de balayage"; l'y-ordre classe les arêtes actives sur l'ordonnée croissante de leur point d'intersection avec la droite de balayage; ces points sont tous distincts pour les droites de balayage ne passant ni par un sommet, ni par un point d'intersection. Bien sûr, l'ensemble des arêtes actives, et l'y-ordre, dépendent de la position de la barre de balayage.

La méthode de BENTLEY-OTTMAN exploite les deux remarques suivantes :

L'y-ordre dépend de l'abscisse de la droite de balayage, mais est en grande partie héréditaire. Il est modifié localement, lorsque la droite de balayage franchit un sommet ou un point d'intersection. Les modifications de l'y-ordre sont toujours les mêmes : zéro, une ou plusieurs arêtes meurent en un sommet, et zéro, une ou plusieurs arêtes naissent en un sommet.

Deux arêtes ne peuvent se couper que si et seulement si elles sont contiguës selon l'y-ordre pour une barre de balayage donnée (le cas particulier de plusieurs arêtes se coupant au même point ne pose pas vraiment de difficultés). On est donc assuré, si l'on teste toutes les arêtes adjacentes dans un y-ordre, de trouver toutes les intersections. Le nombre de ces adjacences est dans les cas les plus courants bien inférieur au nombre de tests de l'algorithme naïf, qui est en $O(N^2)$.

L'algorithme exploitant ces remarques est maintenant décrit de façon informelle :

Les sommets initialement connus sont d'abord classés selon l'xy-ordre.

La barre de balayage est initialisée comme vide, puis elle est déplacée de la gauche vers la droite; elle balaie successivement tous les sommets et tous les points d'intersection, qui sont trouvés au fur et à mesure; aucun n'est oublié, et ils sont toujours trouvés à droite de la position courante de la barre de balayage.

Au cours du balayage, l'y-ordre est géré ainsi : en chaque sommet (initialement connu, ou point d'intersection), les arêtes mourantes sont enlevées, et les arêtes naissantes sont insérées en bonne place. Lors de cette modification, certaines

arêtes deviennent contiguës dans l'y-ordre; cette adjacence entre deux arêtes est une condition nécessaire pour qu'elles s'intersectent; il suffit donc, pour chaque nouvelle adjacence, de tester les arêtes contiguës : on est assuré de n'oublier aucune intersection.

Si deux arêtes deviennent adjacentes et convergent vers la gauche, peut être s'intersectent-elles. Mais alors le point d'intersection est déjà connu : ou bien le point était un sommet initial, ou bien il a été détecté lorsque la barre de balayage avait une abscisse inférieure à celle du point. Ainsi, quand deux arêtes contiguës convergent à gauche de la barre de balayage, il est inutile de tester leur intersection.

Si deux arêtes deviennent adjacentes, convergent vers la droite, et n'ont pas leur sommet droit en commun, alors le point d'intersection des deux droites support est calculé : si ce point appartient bien aux deux arêtes, une nouvelle intersection a été détectée. Dans le cas général d'intersection, le sommet correspondant i est créé et les deux arêtes intersectantes, disons $[s_1,s_2]$ et $[t_1,t_2]$, sont respectivement coupées en $[s_1,i]$, $[i,s_2]$ d'une part, et en $[t_1,i]$, $[i,t_2]$ d'autre part; dans la structure de données de CP-locale, i est un sommet comme les autres, qui aura pour arêtes naissantes $[i,s_2]$ et $[i,t_2]$, et pour arêtes mourantes $[s_1,i]$ et $[t_1,i]$; i est inséré en bonne place dans l'xy-ordre. La mise à jour de la CP-locale ne pose pas de difficultés particulières dans le cas particulier où i est le sommet gauche ou droit d'une ou des deux arêtes intersectantes.

Quels sont les cas où deux arêtes deviennent adjacentes en un sommet S ? S peut être un "point de mort", un "point de naissance", ou un "point de vie". S est un point de mort quand toutes les arêtes incidentes en S meurent en S; S est un point de naissance quand toutes les arêtes incidentes en S naissent en S; autrement, S est un point de vie (les sommets sans aucune arête n'existent pas).

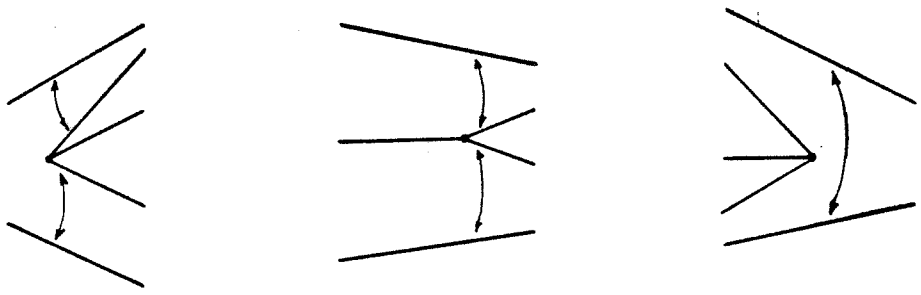


Fig. 4 : Les cas où deux arêtes actives deviennent contiguës

Si S est un point de mort, alors l'arête immédiatement au dessous (dite active basse) et l'arête immédiatement au dessus (dite active haute) deviennent contiguës, quand elles existent : si elles convergent à droite, leur intersection est testée.

Si S est un point de vie ou de naissance, alors l'arête naissante la plus basse et l'active basse d'une part, l'arête naissante la plus haute et l'active haute d'autre part, deviennent adjacentes. Quand ces arêtes contiguës convergent à droite, leur intersection est testée. Ce sont là les seuls cas où des contiguïtés nouvelles apparaissent dans l'y-ordre.

Maintenant, l'algorithme de BENTLEY-OTTMAN peut être formulé plus précisément :

```
BENTLEY-OTTMAN =
{ insérer dans xy-ordre tous les sommets connus, ordonnés;
  initialiser y-ordre comme vide;
  pour chaque sommet S, selon l'xy-ordre, faire
  {
    si S est :
      - un point de mort :
          enlever les arêtes mortes de y-ordre;
          comparer (active-basse, active-haute);
      -un point de naissance :
          insérer les arêtes naissantes dans y-ordre;
          comparer (la plus basse des nouvelles-nées, active-basse);
          comparer (la plus haute des nouvelles-nées, active-haute);
      -un point de vie :
          enlever les arêtes mortes de y-ordre;
          les remplacer dans y-ordre par les arêtes naissantes;
          comparer (la plus basse des nouvelles-nées, active-basse);
          comparer (la plus haute des nouvelles-nées, active-haute);
  }
}


comparer (a, b: arêtes) =
{ si (a et b existent)
  et (a et b convergent vers la droite)
  et (leurs sommets droits sont distincts)
  et (a et b se recouvrent en y)
  alors
       { i ← point d'intersection de a et b;
         si (i existe)
         alors { mettre à jour les structures de données de CP;
                 insérer i dans xy-ordre;
               }
       }
}
```

Cette formulation ne prend pas en compte certains cas particuliers : sommets confondus, arêtes confondues, point d'intersection égal à un sommet déjà connu. Les difficultés ne proviennent pas vraiment des cas particuliers, mais des imprécisions numériques, qui rendent incertaines la reconnaissance des cas particuliers. L'imprécision fait aussi s'enliser l'algorithme dans certains cas. Le paragraphe 4 est consacré à ces problèmes.

3.1.3 Structures de données pour la méthode de BENTLEY-OTTMAN

L'y-ordre :

La structure de données représentant l'y-ordre doit permettre les opérations suivantes [BENT 79] :

- l'insertion d'une nouvelle arête active;
- l'extraction d'une arête qui a cessé d'être active;
- la recherche de l'arête immédiatement inférieure ou supérieure à une arête donnée, selon l'y-ordre.

La structure de données la plus simple est une liste doublement chainée. Il est possible, pour une arête donnée, de connaître immédiatement sa position dans la liste de l'y-ordre : par exemple, c'est l'enregistrement de type arête qui contient un pointeur vers l'arête précédente et un pointeur vers l'arête suivante selon l'y-ordre.

Dans cette structure de données, la suppression d'une active et la recherche de la voisine se font en O(1), ce qui est manifestement optimal. Par contre l'insertion d'une arête naissante n'est pas toujours aussi rapide; deux cas sont possibles :

Le cas le plus favorable est celui d'un sommet de vie, S; au moins une arête meurt en S. Lorsque la barre de balayage franchit S, il est possible d'extraire chaque arête morte en O(1), et dans le même temps de déterminer l'arête "active basse" de S; or les arêtes naissant en S doivent être insérées juste après l'active basse. L'insertion d'une naissante se fait donc en O(1) dans ce cas.

Le deuxième cas est moins favorable : la nouvelle arête active est la première incidente (pour l'α-ordre) en un sommet de naissance S; alors, on ne connaît pas d'avance la position correcte de l'active dans l'y-ordre; la seule solution consiste à parcourir la liste, jusqu'à trouver la bonne position; les autres naissantes en S pourront ensuite être insérées juste après en O(1). Dans le pire des cas, tous les éléments de la liste doivent être testés pour insérer une arête naissante; or cette liste compte N éléments quand tous les segments sont actifs : l'insertion est donc en O(N). Multiplié par le nombre N de segments, la méthode aurait une complexité en $O(N^2)$, comme l'algorithme naïf. C'est cette mauvaise performance qui condamne la liste d'un point de vue théorique; un arbre équilibré lui est préféré [BENT 79].

Un arbre équilibré, contenant N éléments, permet l'insertion, la suppression, et la recherche en O(logN). Les arbres équilibrés les plus courants sont l'AVL, le B-tree, le 2-3-tree [MELH 84][HORO 84].

L'y-ordre est donc représenté par un arbre équilibré. L'insertion et l'extraction se font en O(logN). De même pour la recherche de l'active précédente ou suivante; une recherche plus rapide n'améliore pas les performances théoriques de l'algorithme; en pratique, il est pourtant préférable de superposer une liste doublement chainée à l'arbre : ainsi la recherche de l'active voisine peut se faire en O(1); l'insertion et la suppression se font toujours en O(logN). La suppression ne se fait plus en O(1), mais en O(logN) à cause du nécessaire rééquilibrage.

L'xy-ordre :

Dans l'article original [BENT 79], l'xy-ordre ne contient que les sommets que l'algorithme n'a pas encore balayé; tout sommet balayé est en effet enlevé de l'xy-ordre, et la structure de données représentant l'xy-ordre doit alors permettre les opérations suivantes :

- la recherche du premier élément;
- l'extraction du premier élément;
- l'insertion d'un nouveau sommet.

Un *"heap"* , ou tas, ou puisier, est logiquement proposé dans [BENT 79]. Les deux opérations s'effectuent en O(log|xy-ordre|) = O(logN); le nombre de sommets et de points d'intersection est en effet en $O(N^2)$ et $O(\log N^2) = O(\log N)$.

En fait, tout autre arbre équilibré convient aussi bien que le *"heap"* , mais a des capacités inemployées : ainsi la recherche et l'extraction de n'importe quel élément se font en O(logN). C'est par économie que l'article original préconise un "*heap*"; "employer la structure de données juste suffisante" : première application du principe d'OCCAM.

Sans entrer dans les détails, l'emploi d'un arbre équilibré, et non d'un "*heap*", peut faciliter la prise en compte des cas particuliers. Par exemple, l'insertion d'un sommet dans un "*heap*" ne permet pas de se rendre compte qu'un sommet confondu y figure déjà. Enfin, un arbre équilibré est de toutes façons nécessaire pour représenter l'y-ordre; pourquoi employer deux types différents de structures de données, quand une seule suffit ? Deuxième application du principe d'OCCAM : "ne pas multiplier inutilement les objets".

3.1.4 Complexité de l'algorithme, et comparaisons

L'analyse en complexité de l'algorithme de BENTLEY-OTTMAN est simple [BENT 79]. Soient N le nombre de segments initiaux, K le nombre d'intersections; au pire $K = N(N-1)/2 = O(N^2)$, quand tous les segments s'intersectent : c'est le cas si les segments sont tous tangents à un même cercle. Le nombre de sommets initiaux est manifestement borné par 2N.

Les N+K sommets sont insérés dans l'xy-ordre, ce qui s'effectue en $O(N+K)\log(N+K)$. Comme $K = O(N^2)$, $O((N+K)\log(N+K)) = O(2(N+K)\log N) = O((N+K)\log N)$.

Avec les K intersections, les N segments initiaux donnent le jour à $O(N+2K) = O(N+K)$ arêtes; toutes sont insérées puis extraites de l'y-ordre, ce qui s'effectue en $O((N+K)\log(N+K)) = O((N+K)\log N)$.

Enfin, pour chacun des N+K sommets, l'intersection des arêtes qui deviennent contiguës est testée; le calcul de l'intersection est en O(1); la détermination des actives haute et basse exige O(logN) opérations. Donc $O((N+K)\log N)$.

La complexité totale de l'algorithme de BENTLEY-OTTMAN est donc $O((N+K)\log N)$; on rappelle que l'algorithme naïf est en $O(N^2)$. La méthode de BENTLEY-OTTMAN n'est en $O(N^2)$ que si le nombre de points d'intersection, K, est en $O(N^2)$.

Un autre algorithme calculant les points d'intersection de N segments dans le plan a été proposé par CHAZELLE [CHAZ 83]; cette méthode n'utilise pas le balayage du plan, mais une approche hiérarchique. Elle s'exécute en $O(K+N\log^2 N/\log\log N)$; elle ne classe pas les points d'intersection situés sur une même arête, ce qui est nécessaire pour le logiciel de carte planaire. Sa programmation semble plus délicate que celle de la méthode de BENTLEY-OTTMAN.

MAIRSON et STOLFI [MAIR 83] ont proposé un algorithme qui, étant donnés deux ensembles de segments non intersectants A et B, avec $N = |A| + |B|$, déterminent les K points d'intersection entre A et B en $O(K+N\log N)$.

3.2 CONSTRUCTION DE LA CP-GLOBALE

A ce stade, les points d'intersection ont été déterminés, et la structure de CP-locale est maintenant cohérente. L'algorithme qui construit la CP-globale est présenté de façon informelle pour commencer; il doit créer les bords, les insérer dans l'arbre d'inclusion, et marquer sur chaque brin le bord qui l'emprunte.

L'algorithme procède en un balayage unique. Il considère successivement chaque sommet, du premier au dernier, selon l'xy-ordre :

En chacun des sommets, S, l'algorithme considère les arêtes naissantes en ce sommet, de la première à la dernière, selon l'α-ordre. Pour chaque arête naissante N, on envisage les cas suivants :

Si le brin numéro 2 de cette arête (le "dessous" de l'arête) a déjà été marqué comme appartenant à un bord, alors l'algorithme passe à la naissante suivante s'il en existe, au sommet suivant sinon.

Si le brin numéro 2 de l'arête n'est pas marqué, l'algorithme vient de rencontrer un nouveau bord, B. Ce bord est créé; il est parcouru grâce à la CP-locale, en marquant au passage les brins empruntés comme appartenant à ce bord. Le brin de naissance de ce bord est le dernier brin marqué (un brin de naissance est toujours un brin numéro un) : cette information est stockée dans le bord.

Le bord B créé, et ses brins marqués, il faut maintenant situer B dans la carte. B est externe si et seulement si S est un sommet de naissance et si l'arête naissante N est sa première arête incidente, selon l'α-ordre en S. Autrement, B est un bord interne.

Si B est un bord interne, l'algorithme considère le bord B' empruntant le brin deux (le "dessous") de l'arête du brin de naissance de B. B' existe déjà puisqu'il a été obligatoirement parcouru avant B. Soit B' est un bord externe, et alors il est le père de B. Soit B' est un bord interne, et alors B' est un frère de B, donc le père de B, connu, est celui de B'.

Si B est un bord externe, l'algorithme considère l'arête de butée A du sommet S. Si A n'existe pas, alors B n'est contenu dans aucun bord réel et il a pour père le bord interne virtuel ancêtre de tous les bords. Si A existe, il faut considérer le bord situé "au dessus" de A, c'est à dire le bord B' empruntant le brin un de A. Comme l'arête A naît nécessairement avant S, les deux bords empruntant les brins de A ont été créés et parcourus avant B. Si B' est un bord interne, alors B' est le père de B, sinon B' est un bord externe, et un frère de B; dans ce cas le père de B' est connu et c'est aussi le père de B.

Le père de B étant connu, il ne reste plus qu'à ajouter B à la fin de la liste de ses frères; ceci s'effectue en O(1) si on prend la précaution de tenir à jour un pointeur sur le benjamin des fils de chaque bord existant.

L'algorithme considère ensuite la naissante suivante, ou le sommet suivant. Quand la liste des sommets a été épuisée, la CP-globale est parfaitement connue.

La complexité de l'algorithme s'établit ainsi; chaque arête est testée une fois, à sa naissance, pour savoir si elle a été ou non empruntée, et elle est parcourue et marquée deux fois, une fois sur chaque brin. Le nombre de bords créés est inférieur ou égal au nombre d'arêtes. Soit A le nombre d'arêtes; O(A)=O(N+K); la méthode est en O(A) : A tests, 2A marquages de brins et au plus A créations de bords sont nécessaires; situer un bord dans l'arbre d'inclusion requiert un nombre constant d'opérations. Le pseudocode suivant décrit l'algorithme :

```
CP-locale vers CP-globale =
{
 pour chaque sommet S, pris dans l'xy-ordre, faire
  pour chaque arête naissante en S, selon l'α-ordre, faire
   si (le bord du brin 2 de l'arête n'existe pas encore)
   alors
   {
     créer le bord : B;
     parcourir les brins de B en les marquant comme empruntés par B;
     affecter le brin de naissance de B;
     si (S est une naissance) et (l'arête est la première incidente de S)
     alors
        {
         B est externe;
         si (la butée de S n'existe pas)
         alors B' ← bord virtuel de la carte
         sinon B' ← bord du brin 1 de la butée de S;
         si (B' est interne)
         alors le père de B est B'
         sinon le père de B est le père de B'
        }
     sinon
        {
         B est interne;
         B' ← bord du brin 2 de l'arête de naissance de B;
         si (B' est externe)
         alors le père de B est B'
         sinon le père de B est le père de B'
        }
     ajouter B à la fin de la liste des fils du père de B;
   }
}
```

4 LES EFFETS DE L'IMPRECISION NUMERIQUE

L'imprécision numérique est inhérente à l'utilisation des nombres flottants; son effet sur le calcul des points d'intersection est étudié sur deux exemples. Le comportement de la méthode de BENTLEY-OTTMAN sur ces configurations est étudié en 4.1 et 4.2, et comparé avec celui de l'algorithme naïf : la méthode de BENTLEY-OTTMAN donne des résultats encore plus erronés que la méthode naïve; les erreurs sont propagées sur des configurations correctement traitées par

l'algorithme naïf; surtout, une erreur fatale peut interrompre le déroulement des programmes utilisant la méthode de BENTLEY-OTTMAN, alors que l'algorithme naïf arrive toujours à son terme.

Les conséquences de l'imprécision sur le comportement des algorithmes géométriques les plus utilisés en synthèse d'images sont rappelées en 4.3; le cas de l'algorithme d'ATHERTON est détaillé. La comparaison montre que l'imprécision numérique a des conséquences bien plus graves pour l'algorithme de BENTLEY-OTTMAN que pour les méthodes classiques de la synthèse d'images qui rendent toujours un résultat approché; les causes de ce phénomène sont récapitulées en 4.4.

Une solution à l'imprécision numérique est ensuite présentée en 4.5 : l'algorithme de BENTLEY-OTTMAN ainsi modifié fonctionne correctement, dans tous les cas de figure. Cette solution est ensuite évaluée.

4.1 PREMIER CONTRE-EXEMPLE

4.1.1 le calcul de l'intersection de deux segments de droite

GANGNET [GANG 87] étudie l'effet de l'imprécision numérique sur le calcul de l'intersection de deux segments de droite non parallèles. Par commodité, il suppose employée une arithmétique flottante en base B=10, avec P=4 chiffres significatifs; les conclusions peuvent cependant être généralisées à toute arithmétique flottante. On note x^* la meilleure valeur flottante représentant le nombre réel x.

Un nombre réel est représenté par le nombre flottant :
0.0 10^0, aussi noté 0.0e0
ou par $\pm m\ 10^k$, aussi noté ±mek,
avec $1/B \leq m < 1$, ou $0.1 \leq m \leq 0.9999$
et k entier relatif, tel que $-E \leq k \leq E$
(On suppose E assez grand pour ce qui suit)

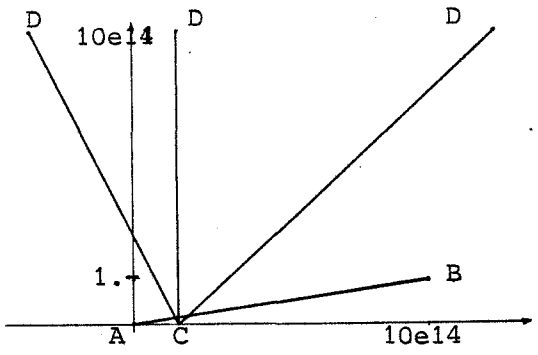


Fig. 5 : La configuration de [GANG 87]

Soient les points A(0.0e0, 0.0e0) et B(0.1e15, 0.1e1)
La droite (AB) a pour équation :
(AB) : x(0.1e1 - 0.0e0) + y(0.0e0 - 0.1e15) = 0
(AB) : x(0.1e1) - y(0.1e15) = 0

Soient les points C(0.1e1, 0.0e0) et D_u(0.1e1 + u, 0.1e15)
On étudie l'effet du paramètre u sur l'intersection de (AB) et (CD) :

(CD_u) a pour équation :
(CD_u) : x(0.1e15 - 0.0e0) - y(0.1e1 + u - 0.1e1) + (0.0e0)(0.1e1 + u) - (0.1e1)(0.1e15) = 0.
(CD_u) : x(0.1e15) - y(u) - 0.1e15 = 0

Calculons le point d'intersection I_u de (AB) et (CD_u) par :
ax + by + c = 0
a'x + b'y + c' = 0
Δ = ab' - a'b = -(0.1e1)(u) + (0.1e15)(0.1e15) = -0.1e1 u + 0.1e29
Δ^*= 0.1e29 est la valeur flottante la plus proche de Δ, pour u tel que :
0.1e29 = (0.1e29 - u)*, soit |u| < 0.0001e29,
soit : u $\in$]-0.1e(30-P), 0.1e(30-P)[

Δx = -cb' + c'b = -(0.0e0)u + (0.1e15)(0.1e15) = 0.01e30 = 0.1e29 = Δx^*
Δy = -ac' + a'c = (0.1e1)(0.1e15) - (0.1e15)(0.0e0) = 0.01e16 = 0.1e15 = Δy^*

x_I = (0.1e29)/(0.1e29 - 0.1e1 u) et y_I = (0.1e15)/(0.1e29 - 0.1e1 u)
donc x_I^* = 0.1e1 et y_I^* = 0.1e-13 pour u $\in$]-0.1e(30-P), 0.1e(30-P) [

Ainsi, pour toutes les valeurs de u $\in$]-0.1e(30-P), 0.1e(30-P)[=]-0.1e26, 0.1e26[, toutes les droites (CD_u) ont le même point d'intersection avec la droite (AB) ! Les coordonnées du point d'intersection ne sont pas seulement entachées d'imprécision, ce que l'on savait déjà; elles sont aussi en contradiction avec les propriétés du plan euclidien. Cette incohérence ne peut passer inaperçue lors de la modélisation d'une scène 2D contenant le segment [A,B] et au moins deux segments [C,D_i] et [C,D_j] distincts, qui auront pourtant la même intersection avec [A,B].

Les coordonnées des sommets de ces cartes planaires ne peuvent pas être représentées de façon cohérente avec notre arithmétique flottante. L'exemple et le constat se généralisent trivialement aux scènes 3D constituées de polyèdres : nous y reviendrons au chapitre III.

Cette incohérence est propre aux configurations et à l'arithmétique flottante, et est indépendante de l'algorithme de BENTLEY-OTTMAN. En fait, avec ces configurations, *tous* les algorithmes déterminant les intersections entre segments (algorithme naïf, [BENT 79] [OTTM 82] [OTTM 82b] [OTTM 85] [CHAZ 83] [MAIR 83] [SZIL 84]...) et utilisant cette arithmétique donneront, au mieux, des résultats géométriquement aberrants; au pire, ils ne donneront pas de résultats du tout, comme la méthode de BENTLEY-OTTMAN (cf 4.1.2.1, 4.1.2.2).

♦ **Remarque** On objectera, non sans raison, qu'il est d'une certaine façon normal que $((CD_u) \cap (AB))^* = (CD_0) \cap (AB)$ pour toutes les valeurs de u telles que $D_u^* = D_0$, soit pour u tel que $(0.1e1 + u)^* = 0.1e1$, donc pour $u \in$]-0.1e-2, 0.1e-2[. Certes : mais cet intervalle est strictement inclus dans]-0.1e26,0.1e26[.

GANGNET conclut en considèrant le cas où le vecteur directeur des droites (AB) et (CD_u) est unitaire ($a^2 + b^2 = 1$) : l'intersection de (AB) et (CD_u) reste pathologique, non euclidienne.

Les valeurs numériques de l'exemple précédent ont peut-être le tort de suggérer que de "grands nombres" (0.1e15 dans l'exemple) sont nécessaires, et que ce type de configuration est donc très improbable. Il n'en est rien; remplaçons 0.1e15 par 0.1eN dans l'exemple de GANGNET :

A = (0.0e0, 0.0e0) B = (0.1eN, 0.1e1)
C = (0.1e1, 0.0e0) D_u = (0.1e1 + u, 0.1eN)
(AB) : x(0.1e1) - y(0.1eN) = 0
(CD) : x(0.1eN) - y(u) - 0.1eN = 0
Δ = -0.1e1(u) + 0.1e(2N-1)
Δ^* = 0.1e(2N-1) quand |u| < 0.0001e(2N-1), soit $u \in$]-0.1e(2N-P), 0.1e(2N-P)[

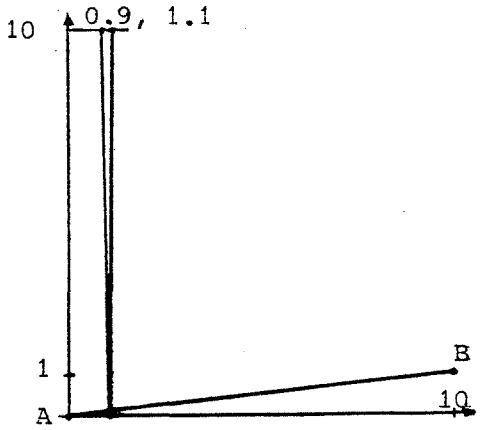


Fig. 6 : La configuration de [GANG 87], sans grands nombres

Par exemple, pour N=2, on a : A(0, 0), B(10, 1), C(10, 0), D(1+u, 10); pour u $\in$]-0.1, 0.1[, donc pour D_u entre (0.9, 10) et (1.1, 10), tous les points I_u^* sont égaux; or 0.1e2 n'est certainement pas un "grand nombre". On remarquera que les droites (CD_u) sont presque parallèles.

Quelle que soit la taille de N, il existe donc toujours un intervalle de mesure non nulle, tel que les points d'intersection calculés de (AB) avec (CD_i) et (CD_j) soient incohérents.

4.1.2 Calcul de l'intersection de [A,B] et [D,C]

Etudions l'effet de l'imprécision sur le test de l'intersection des segments [A,B] et $[D_u,C]$ de l'exemple de GANGNET : on choisit u dans l'intervalle d'incohérence, et tel que la pente de la droite (D_uC) soit négative : $u < 0$.

On a vu que le point d'intersection de (AB) et (D_uC) est le point $I_u^*=I=(0.1e1, 0.1e-13)$; en admettant que I_u^* se trouve exactement sur les droites (AB) et (D_uC), comment un algorithme peut-il décider si I_u appartient bien aux segments [A,B] et $[D_u,C]$?

Un premier test possible est : $I_u \in [D_u,C]$ si et seulement si $D_u \leq I_u \leq C$, selon l'xy-ordre. Bien sûr, l'algorithme ne connait les coordonnées que par leurs représentations flottantes : selon ce test, $D_u \leq C \leq I_u^*$; donc I_u n'appartient pas à $[D_u,C]$.

Un deuxième test possible est : $I_u \in [D_u,C]$ si et seulement si $x_D \leq x_I \leq x_C$ et si $y_C \leq y_I \leq y_D$. Selon ce test, mathématiquement équivalent au premier quand on ne tient pas compte des imprécisions numériques, I_u appartient à $[D_u,C]$. Ainsi, deux tests mathématiquement équivalents donnent des résultats contradictoires.

Si la méthode de BENTLEY-OTTMAN emploie le premier test, ou tout autre test donnant le même résultat, elle oublie le point d'intersection I_u; elle n'oublie pas seulement ce point d'intersection "pathologique", mais aussi des points d'intersection que trouverait l'algorithme naïf; plus probablement, la méthode de BENTLEY-OTTMAN peut aussi ne pas arriver normalement à son terme; ceci est montré en 4.1.2.1.

Si la méthode de BENTLEY-OTTMAN utilise le deuxième test, une erreur fatale met fin au déroulement du programme; ceci est montré en 4.1.2.2.

4.1.2.1 Si [A,B] et [C,D] ne s'intersectent pas

Le programme a donc oublié le point d'intersection I_u; cette erreur peut certes être tolérée dans certaines applications; mais il va commettre ensuite des erreurs bien plus graves. Suivons le déroulement de l'algorithme sur le cas de la figure 7.

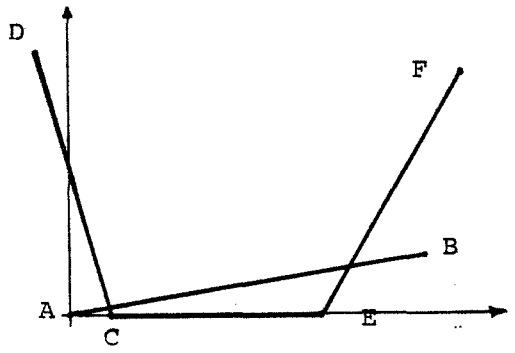


Fig. 7 : Le premier contre-exemple

Juste avant C, l'y-ordre est : [A,B], [D,C]. En C, [D,C] est remplacée par [C,E] dans l'y-ordre, qui devient donc : [A,B], [C,E]; l'ordre correct est [C,E], [A,B]. [A,B] et [C,E] sont comparées : leur intersection est vide. En E, [C,E] est remplacée par [E,F]. L'y-ordre devient : [A,B], [E,F]; l'ordre correct serait : [E,F], [A,B]. [C,E] et [E,F] sont comparées : pour l'algorithme, elles divergent : la plus haute, [E,F], a une pente supérieure à celle de la plus basse, [A,B]. Le point d'intersection de [A,B] et [E,F] est donc oublié par l'algorithme de BENTLEY-OTTMAN; il serait pourtant détecté par l'algorithme naïf.

Ce qui précède suppose que la gestion de l'y-ordre n'est pas perturbée par les erreurs de classement. Mais ces erreurs de classement peuvent être fatales à certaines implantations de l'y-ordre. Supposons que l'y-ordre soit représenté par un arbre binaire (équilibré ou non). Supposons aussi que l'arbre soit le seul accès aux actives : il faut descendre dans l'arbre pour enlever une active morte (c'est la solution initiale de [BENT 79]).

Par définition, un arbre binaire doit vérifier certaines contraintes; ainsi, les éléments rangés dans le fils gauche (resp. droit) d'un noeud doivent être inférieurs (resp. supérieurs) à l'elément porté par le noeud. Les algorithmes classiques de recherche dichotomique et de mise à jour exploitent bien sûr ces propriétés.

Considérons l'arbre binaire de la figure 8 représentant un y-ordre partiellement faux; l'y-ordre correct est a_1, a_2, a_3; pour fixer les idées, les ordonnées des points d'intersection entre les actives a_1, a_2, a_3 et la position actuelle de la droite de

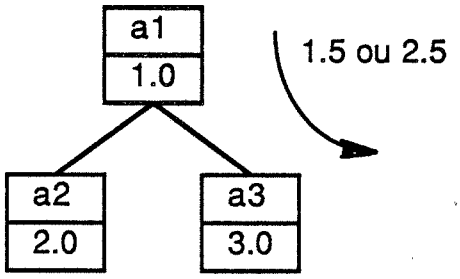


Fig. 8 : Un arbre binaire représentant un y-ordre partiellement faux

balayage sont respectivement 1.0, 2.0, et 3.0. L'y-ordre représenté est a_2, a_1, a_3. Si une arête a naît entre a_1 et a_2, ou entre a_2 et a_3, la consultation de l'arbre situe dans les deux cas la naissance de a entre a_1 et a_3, ce qui est faux (et, bien sûr, des comparaisons nécessaires entre arêtes vraiment contiguës ne seront pas effectuées par la méthode de BENTLEY-OTTMAN, et d'autres points d'intersection seront oubliés).

Toutes ces erreurs ne sont pas forcément fatales dans l'immédiat; tout dépend de l'implantation. Par contre, comme l'algorithme classique de recherche dichotomique ne peut même pas trouver l'arête a_2 dans l'arbre, il ne peut donc pas l'enlever de l'y-ordre quand elle meurt; même si le programme a pu se dérouler cahin-caha jusque là, cette erreur lui portera le coup de grâce.

On ne peut exclure que certaines implantations de l'y-ordre puissent résister à ces incohérences. Mais, dans ce cas, l'algorithme rend des résultats bien plus erronés que ceux de l'algorithme naïf, en oubliant des points d'intersection ordinaires tels [A,B] ∩ [E,F].

4.1.2.2 Si [A,B] et [D,C] s'intersectent

En A, l'y-ordre est : [A,B]. En B, le point suivant A dans l'xy-ordre, l'y-ordre devient : [A,B], [D,C]. Les deux arêtes sont comparées; le test employé trouve que [A,B] ∩ [D,C] = I. I est inséré dans l'xy-ordre, tout de suite après C (ce qui est bien sûr absurde). [A,B] et [D,C] sont coupées en [A,I][I,B], et [D,I][I,C]. L'y-ordre devient [A,I],[D,I].

En C, le point suivant B dans l'xy-ordre, l'algorithme tente d'enlever l'arête morte [I,C] de l'y-ordre. Or cette arête n'est pas active ! Cette situation, théoriquement impossible, n'est pas gérée par l'algorithme, et l'erreur est fatale.

4.2 DEUXIEME CONTRE-EXEMPLE

Le deuxième contre-exemple est le cas de la figure 9. Les segments [A,B] et [C,D] ont été choisis de telle sorte que [A,B] ∩ [C,D] = (2/3, 0) = I. Si l'arithmétique flottante de l'exemple précédent est employée, ce point d'intersection aura au mieux pour représentation flottante I^* = (0.6667e1, 0.0e0). On peut prendre par

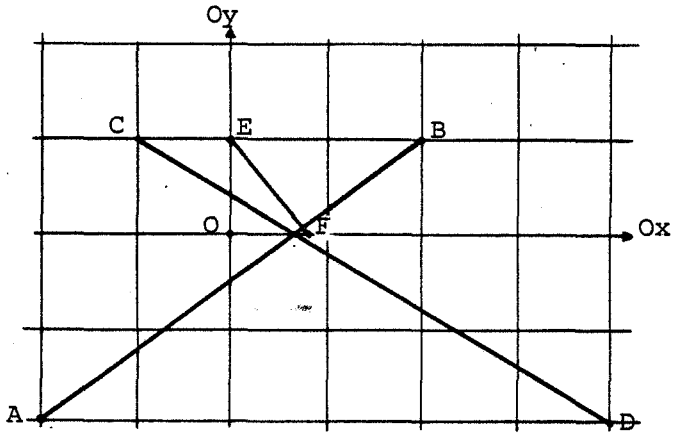


Fig. 9 : Le deuxième contre-exemple

exemple $A = A^* = (-2, -2)$, $B = B^* = (2, 1)$, $C = C^* = (-1, 1)$, et $D = D^* = (4, -2)$.

Le point $F = F^* = (0.6667e1, -0.1e\text{-}10)$ a été choisi de telle sorte que $I < F=F^* < I^*$; la représentation flottante inverse donc l'ordre de I et F. Les coordonnées du point E n'ont qu'une importance secondaire; il suffit que $E < F$ et que [E,F] intersecte [A,B] sans intersecter [C,D].

L'algorithme de BENTLEY-OTTMAN ne fonctionne pas correctement sur cette configuration :

En A, le premier sommet selon l'xy-ordre, l'y-ordre est : [A,B]. En C, [C,D] devient active et est insérée dans l'y-ordre; les deux actives sont comparées; elles s'intersectent en I^*; ce point est inséré dans l'xy-ordre, juste après F (alors que I est inférieur à F); [A,B] est coupée en $[A,I^*]$, $[I^*,B]$, et [C,D] est coupée en $[C,I^*]$,$[I^*,D]$; l'y-ordre est devenu : $[A,I^*]$, $[C,I^*]$.

La barre de balayage franchit ensuite le point E : [E,F] est insérée dans l'y-ordre qui devient : $[A,I^*]$, $[C,I^*]$, [E,F]. Ces deux dernières arêtes deviennent contiguës et sont donc comparées; leur intersection est vide.

En F, [E,F] meurt et est enlevée de l'y-ordre. L'algorithme de BENTLEY-OTTMAN ne pourra donc pas trouver le point d'intersection $[I,B] \cap [E,F] = J$; J est pourtant un point d'intersection tout à fait ordinaire, qui serait détecté par l'algorithme naïf.

Cette erreur n'est pas fatale à l'algorithme. Mais il suffit d'ajouter une arête [F,G] horizontale, avec F < G. En F, [F,G] remplace [E,F] dans l'y-ordre, qui devient : [A,I*], [C,I*], [F,G]; ces deux dernières arêtes sont comparées et n'intersectent pas. En I*, l'y-ordre devient : [I*,D], [I*,B], [F,G]; cet ordre est faux, et va se propager; le paragraphe précédent a montré quelles pouvaient être les conséquences fatales de ce type d'erreur.

Dans cet exemple, il n'existe aucune abscisse flottante possible entre les abscisses de I et de I*; mais en fait, l'écart entre I et I* est souvent bien plus grand, à cause du cumul des erreurs d'arrondi (considérées comme nulles par l'exemple) sur le calcul des coefficients de l'équation des droites supports des segments.

Les deux configurations détaillées ne sont évidemment pas les seules à perturber la méthode de BENTLEY-OTTMAN : elles ont été choisies pour leur simplicité; un seul contre-exemple aurait d'ailleurs suffi à notre démonstration.

Bien sûr, l'imprécision ne se manifeste pas uniquement dans le cas de l'algorithme de BENTLEY-OTTMAN :

4.3 L'IMPRECISION SUR D'AUTRES ALGORITHMES

En synthèse d'images, les algorithmes comme le tampon de profondeur, le lancer de rayons, la méthode de WATKINS ... déterminent les surfaces visibles d'une scène pour chaque pixel du terminal d'affichage. On va rappeller les effets de l'imprécision sur ces méthodes; ils sont qualitativement très différents des effets de l'imprécision sur l'algorithme de BENTLEY-OTTMAN.

Le tampon de profondeur et le lancer de rayons, les méthodes les plus simples, réagissent comme l'algorithme naïf à l'imprécision numérique : les erreurs restent locales, et elles sont souvent indiscernables des phénomènes d'aliassage. Le pire cas est celui des scènes comportant des surfaces quasi parallèles et très proches : elles peuvent alors être mal classées sur de nombreux pixels [ROSS 86]. L'emploi d'un epsilon ne résoud que partiellement ce problème. Mais, même dans ce cas, ces algorithmes arrivent normalement à leur terme.

Bien sûr, les algorithmes géométriques de la synthèse d'image ne sont pas systématiquement aussi "naïfs" que le lancer de rayons ou le tampon de profondeur; certains utilisent aussi le paradigme du balayage, tels l'algorithme de WATKINS ou celui, plus général et plus récent, d'ATHERTON [ATHE 83]. Qu'advient-il de l'imprécision sur ce dernier algorithme ?

4.3.1 Rappels sur la méthode d'ATHERTON

La méthode d'ATHERTON [ATHE 83] visualise sur des mémoires de trame les scènes facettisées modélisées par des arbres de construction (ou arbres CSG). Comme l'algorithme maintenant classique de WATKINS, cette méthode réduit le problème initial 3D à une suite de problèmes 2D, un pour chaque ligne de l'image calculée.

Le problème 2D s'énonce ainsi : soient un ensemble de polygones dans le plan Oxy, et un ensemble d'opérations booléennes régularisées [TILO 80] à effectuer sur ces polygones; soit un oeil Ω situé à moins l'infini sur l'axe Oy; quels sont les segments visibles pour Ω, en tenant compte des opérations booléennes ?

Comme WATKINS, ATHERTON réduit ce problème 2D à l'étude des intervalles [xg,xd] "homogènes" : aucun des sommets des segments des contours des polygones n'a une abscisse appartenant à]xg,xd[.

4.3.2 Etude d'un intervalle [xg, xd]

Comment déterminer le(s) segment(s) visible(s) dans l'intervalle [xg,xd], où sont actifs les segments s_i ? Ces segments sont connus par l'équation de leur droite de support : y=ax+b; cette forme est la plus commode car la profondeur y(x) des segments est très souvent calculée.

La méthode trie les segments par profondeur (distance à l'oeil) croissante en xg, et en xd. L'ordre des segments n'est donc pas géré par propagation, comme dans l'algorithme de BENTLEY-OTTMAN. Soit (sg_1, sg_2...) l'ordre des segments en xg, et soit (sd_1, sd_2...) l'ordre des segments en xd. Bien sûr, quand xg=xd, $(sg_i) = (sd_i)$.

Si les deux ordres sont semblables : $(sg_i) = (sd_i)$, alors le même segment est visible sur tout l'intervalle [xg,xd], et il est déterminé par une variante de la méthode que ROTH [ROTH 82] a proposée pour le lancer de rayons.

Sinon, l'intervalle est partitionné en deux intervalles plus petits : soit k, le plus petit indice tel que $sg_k \neq sd_k$; sg_k et sd_k s'intersectent forcément entre xg et xd; soit (x_k, y_k) le point d'intersection entre sg_k et sd_k; le traitement de [xg,xd] est récursivement réduit à celui de [xg, tronc(x_k)], et de [tronc(x_k)+1, xd].

Lors de la programmation de cette méthode, TALLOT et moi-même nous sommes heurtés à un problème inattendu, dû à l'imprécision numérique : quand les ordres sg_i et sd_i commencent à différer en i=k, il arrive que pour certaines valeurs numériques, x_k n'appartienne pas à [xg, xd]. Cette situation est absurde et contredit entre autres le théorème de ROLLE; mais elle est possible. Par exemple, $x_k > xd$,

avec tronc(x_k)=xd; l'intervalle [xg,xd] est alors coupé en [xg,xd], [xd+1, xd] : pour traiter [xg,xd], la fonction calcul_intervalle([xg,xd], s_i) appelle donc récursivement la fonction calcul_intervalle([xg,xd], s_i), à l'infini, ou plutôt jusqu'à saturation de la pile.

A titre d'exemple, sur SPS9, sous ROS, en flottant double précision, cette mésaventure survient avec les valeurs numériques suivantes :

xg=725, xd=783,

le segment sg_k a pour extrêmités :

x_1=579.37763500000010

y_1=0.8231469999999998

x_2=1447.7350419261313

y_2=0.7967479767846342

le segment sd_k a pour extrêmités :

x_1=680.41827873007446

y_1=0.82007525640606697

x_2=782.80916197758779

y_2=0.81696246012848607

le point d'intersection calculé a pour abscisse :

x_k=723.98901817231012 < xg

Ce cas authentique est sans doute moins parlant que cet exemple artificiel :

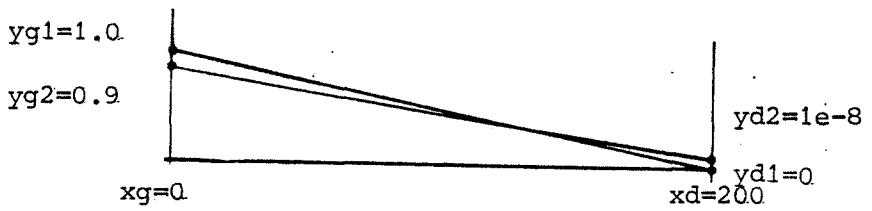


Fig. 10 : L'imprécision et l'algorithme d'ATHERTON

Sur SPS9, sous ROS, en flottant simple précision, on considère les deux segments s_1 = [(xg=0, yg_1=1), (xd=200, yd_1=0)] et s_2 = [(xg=0, yg_2=0.9), (xd=200, yd_2=1e-8)]. s_1 et s_2 s'intersectent manifestement entre xg et xd. s_1 a pour équation $y=a_1x+b_1$, avec $a_1=(yd_1-yg_1)/(xd-xg)$, et $b_1=yg_1-a_1*xg$; s_2 a pour équation $y=a_2x+b_2$, avec $a_2=(yd_2-yg_2)/(xd-xg)$, et $b_2=yg_2-a_2*xg$; le point d'intersection a pour abscisse $x_k=(b_1-b_2)/(a_2-a_1)$; le calcul flottant donne x_k=200.000015259 : x_k n'appartient pas à [xg,xd].

4.3.3 La solution

La difficulté se surmonte ici trivialement, en remarquant que x_k ne sert qu'à couper l'intervalle [xg, xd]. Si x_k > xd, alors il suffit de couper [xg, xd] en [xg, xd-1] et [xd, xd]; si x_k < xg, il suffit de couper [xg, xd] en [xg, xg] et [xg+1, xd]. Le programme ainsi corrigé fonctionne correctement, mis à part le cas classique des scènes comportant des surfaces visibles très proches; mais, même dans ce cas, l'algorithme s'achève normalement.

La méthode d'ATHERTON propage beaucoup moins de résultats que celle de BENTLEY-OTTMAN : ainsi toutes les arêtes actives sont retriées pour chaque borne d'intervalle. Aussi les erreurs n'ont-elles que des conséquences très locales : elles peuvent donc être tolérées ! Hélas, cette solution simple au problème de l'imprécision ne peut pas être utilisée pour l'algorithme de BENTLEY-OTTMAN.

4.4 RECAPITULATION

Pour certaines configurations de segments, le calcul en arithmétique flottante des points d'intersection entre droites fournit des coordonnées aberrantes : il existe des cartes planaires dont les coordonnées des sommets ne peuvent être représentées correctement.

L'algorithme naïf donne alors des résultats non compatibles avec les propriétés du plan euclidien. Ainsi le graphe produit n'est pas forcément planaire. Cependant, l'algorithme naïf arrive toujours normalement à son terme, et les erreurs ne sont pas propagées sur les autres points d'intersection. Certaines applications peuvent tolérer ces quelques menues erreurs (ce n'est certes pas le cas du logiciel de carte planaire).

L'algorithme de BENTLEY-OTTMAN est beaucoup plus perturbé. Des points d'intersection peuvent être oubliés, alors qu'ils seraient détectés par l'algorithme naïf. Surtout, une erreur fatale peut interrompre le déroulement de l'algorithme. Le détail des perturbations subies dépend cependant beaucoup de l'implantation, des implantations théoriquement équivalentes ayant des comportements très différents.

Pourquoi l'algorithme naïf résiste-t-il beaucoup mieux à l'imprécision numérique que la méthode de BENTLEY-OTTMAN ?

Quels que soient ces résultats partiels, l'algorithme naïf effectue toutes les comparaisons entre arêtes : une erreur sur un test ne se propage pas.

Au contraire, l'algorithme de BENTLEY-OTTMAN propage les résultats courants; par exemple, comme beaucoup d'algorithmes efficaces, à commencer par les algorithmes de tri, il exploite les propriétés des relations d'ordre, comme la tran-

sitivité : si les informations $a < b$, et $b \leq c$ ont été établies, alors un algorithme efficace en déduit sans calcul supplémentaire que $a < c$. La méthode de BENTLEY-OTTMAN procède ainsi pour l'y-ordre sur les actives et pour l'xy-ordre sur les coordonnées.

Malheureusement, les comparaisons sont le résultat de calculs effectués en flottant; l'imprécision numérique peut donc fausser un de ces tests. Il se peut fort bien que $a > b$, mais que $a^* < b^*$; pour l'algorithme, qui n'a accès qu'à la représentation flottante des nombres : $a < b$. Initialement, l'erreur est certes minime, dans les limites de l'imprécision. Mais l'algorithme la propage : s'il a établi par ailleurs que $b \leq c$, il va déduire l'information $a < c$ illégitimement.

Les informations incohérentes (comme un arbre binaire partiellement faux) ne sont évidemment ni prévues ni gérées par l'algorithme; en conséquence, on ne peut plus répondre de rien, et le programme peut fort bien commettre une erreur fatale.

Insistons une dernière fois sur la profonde différence de nature entre les effets uniquement locaux de l'imprécision sur l'algorithme naïf ou les algorithmes classiques de la synthèse d'images, et les conséquences catastrophiques de l'imprécision sur la méthode de BENTLEY-OTTMAN. D'après mon expérience, dans la plupart des cas *réels, non aléatoires* de 4000 arêtes ou plus, la méthode de BENTLEY-OTTMAN n'arrive pas à son terme quand est employée une arithmétique flottante sur 64 bits.

4.5 UNE SOLUTION AUX INCOHERENCES NUMERIQUES

4.5.1 Quelques approches classiques

Comment résoudre le problème des incohérences numériques ? Une idée classique est l'emploi d'un epsilon : si deux nombres a^* et b^* diffèrent de moins de epsilon (différence absolue ou relative selon les cas), alors a et b sont considérés comme égaux. Le choix d'un epsilon correct est un art délicat : la valeur correcte de l'epsilon est très dépendante de l'application et de l'ordre de grandeur des nombres les plus fréquemment utilisés. Cette technique améliore les résultats des algorithmes naïfs, mais ne résoud que partiellement notre problème. Il existe en effet plusieurs façons mathématiquement équivalentes de calculer deux nombres a^*, b^*; il se peut que pour une premier mode de calcul $(a^* - b^*)^* < \varepsilon$, et que pour un deuxième $(a^* - b^*)^* > \varepsilon$. Ces deux résultats ne peuvent être tous les deux corrects; comment décider alors de façon sûre de l'ordre de a et b ?

La méthode "Permutation-Perturbation" exploite justement ce phénomène; une valeur est calculée de plusieurs façons différentes, mathématiquement équivalentes, par *permutation* de termes; de plus, une *perturbation* aléatoire modifie le dernier bit, pour compenser le cumul des troncatures. L'ensemble des valeurs obtenues est

considérée comme une population statistique, sur laquelle on peut calculer une moyenne, un écart-type, etc. D'après un constat empirique, trois valeurs constituent un échantillon significatif dans la plupart des cas; ce constat a ensuite été confirmé par une démonstration rigoureuse. Cette méthode permet d'obtenir des encadrements fiables $[u_0^*, u_1^*]$ pour les valeurs calculées de u. Malheureusement elle s'avère insuffisante pour l'algorithmne de BENTLEY-OTTMAN : soit u = a - b; si $0 \in [u_0^*, u_1^*]$, l'ordre de a et b ne peut être déterminé de façon sûre, les intervalles $[u_0^*, u_1^*]$ n'étant pas de mesure nulle dans le cas général.

♦ **Remarque** Ironiquement, dans la grande majorité des applications ([GREE 86] est la seule exception que je connaisse), la valeur parfaitement exacte des nombres, comme les coordonnées des sommets ou les paramètres des équations de droite, ne nous intéressent guère; à la limite, l'imprécision serait tout à fait supportable si elle ne modifiait pas l'ordre entre les nombres : ce sont en effet les comparaisons numériques fausses qui rendent instable l'algorithme de BENTLEY-OTTMAN.

4.5.2 L'arithmétique rationnelle

La seule solution que j'ai trouvée est maintenant présentée; elle est d'une programation simple, et supprime tous les effets de l'imprécision. Elle supprime l'imprécision elle-même.

Les sommets initiaux sont supposés recalés sur une très grande grille carrée entière, $[0..G]^2$; a priori, G=20000 est largement suffisant pour la plupart des applications, mais rien n'empêche de choisir un G encore plus grand.

La droite $((x_0\ y_0)(x_1\ y_1))$ a alors pour équation : ax + by + c=0, avec par exemple $a= y_0-y_1$, $b= x_1-x_0$, $c= x_0y_1 - x_1y_0$. a, b et c sont entiers, et $|a|, |b| \leq G$, et $|c| \leq 2G^2$. Bien sûr ces entiers sont fidèlement représentés en machine.

Le point d'intersection M entre deux de ces droites "entières" vérifie ax+by+c=a'x+b'y+c'=0; soient Δ, Δx, Δy les déterminants du système; supposons $\Delta \neq 0$; alors ce point M a pour coordonnées $(\Delta x/\Delta\ \Delta y/\Delta)$; il est donc "rationnel". $|\Delta x|$ et $|\Delta y| \leq 4G^3$, et $|\Delta| \leq 2G^2$. En machine, ce point peut être représenté fidèlement, sans aucune imprécision; par exemple il est stocké en coordonnées homogènes entières $(\Delta x\ \Delta y\ \Delta)$, ou bien sous une forme euclidienne : x_e, y_e, x_r, y_r, Δ, avec $\Delta x= x_e\Delta + x_r$, $0 \leq x_r < \Delta$ et $\Delta y = y_e\Delta + y_r$, $0 \leq y_r < \Delta$; comme les points d'intersection hors de $[0,G]^2$ sont écartés, $0 \leq x_e, y_e \leq G$, et $0 \leq x_r, y_r < \Delta \leq 2G^2$; la forme euclidienne semble la plus commode, car la plus grande valeur absolue entière à stocker est $2G^2$. A titre d'exemple, sur une machine 32 bits, le plus grand entier fidèlement représenté est 2^{31}-1 et il faut donc que $2G^2 < 2^{31}$; d'où G < 32768. Il est bien sûr possible d'utiliser des valeurs supérieures pour G, mais les entiers disponibles sur la machine ne suffiront pas pour mémoriser certains des nombres nécessaires.

Il se peut que certaines valeurs, provisoirement nécessaires, excèdent les possibilités de la machine, par exemple, des valeurs pour Δx ou Δy. Diverses techniques permettent de passer outre, comme l'utilisation des flottants double précision pour représenter les entiers dans les limites de la mantisse; ceci permet de faire l'économie d'une librairie arithmétique manipulant des entiers de longueur quelconque; la librairie est en effet plus lente (appel de procédures, passage de paramètres, empilements de variables locales...).

Une autre technique peut être utilisée lors de la comparaison de deux nombres rationnels a/b et c/d, avec $0 \leq a < b \leq 2G^2$ et $0 \leq c < d \leq 2G^2$. La première idée exploite le fait que l'ordre de a/b et de c/b est l'ordre de ad et de bc; mais alors il faut représenter des nombres inférieurs ou égaux à $2G^2(2G^2-1)$; une autre technique évite ces trop grands nombres, en faisant appel au développement en fractions continues; plus précisément : ordre(a/b, c/d) = ordre(d/c, b/a) = ordre ([d/c]+(d%c)/c, [b/a]+(b%a)/a) ([d/c] est le quotient euclidien de d par c; d%c est le reste de d dans la division euclidienne par c). Si [d/c] ≠ [b/a], alors ordre(a/b, c/d) = ordre ([d/c], [b/a]) : la comparaison de deux entiers est triviale; sinon ordre(a/b, c/d) = ordre((d%c)/c,(b%a)/a); or c < d et a < b; donc le processus va converger en un nombre fini d'étapes.

Exemple : ordre(2/7, 3/10)= ordre(10/3, 7/2)= ordre(3+1/3, 3+1/2)= ordre(1/3, 1/2)= ordre(2/1, 3/1)= ordre(2+0/1, 3+0/1)= inférieur.

En fait, cet algorithme calcule simultanément le PGCD de a et b d'une part, de c et d d'autre part. Son efficacité est celle du calcul du PGCD le plus rapide; l'étude en complexité de l'algorithme d'EUCLIDE est effectuée dans [KNUT 81]; elle est ici inutile : les entiers rencontrés ayant une taille limitée, il existe un pire des cas; on peut donc considérer que la comparaison s'effectue en O(1) (si G est un paramètre du programme, la comparaison s'effectue en $O(\log(2G^2)) = O(\log G)$).

Le même argument vaut pour les autres opérations arithmétiques (+, -, *, /); les opérations peuvent donc être effectuées sans aucune erreur et en O(1) : l'algorithme de BENTLEY-OTTMAN fonctionne alors correctement et son efficacité théorique n'est pas modifiée, G étant considérée comme une constante.

♦ **Remarque** La solution proposée exploite la linéarité des segments de droite; une arithmétique rationnelle ne serait pas suffisante si l'on généralisait l'emploi de BENTLEY-OTTMAN aux arcs de courbe, par exemple des arcs de coniques ou de cubiques. La généralisation de l'algorithme aux arcs de courbe monotones ne va donc plus de soi.

Imaginons maintenant un logiciel de carte planaire dynamique : des arêtes peuvent être enlevées ou ajoutées dans la fenêtre $[0,G]^2$; notamment, une requête a priori sensée veut joindre par une nouvelle arête deux sommets de la carte, comme $(x_0/\Delta_0\ y_0/\Delta_0)$ et $(x_1/\Delta_1\ y_1/\Delta_1)$. La nouvelle droite, support de l'arête, a bien une équation "entière" $ax+by+c=0$, avec $a= y_0\Delta_1 - y_1\Delta_0$, $b= x_1\Delta_0 - x_0\Delta_1$, $c= x_0y_1 - x_1y_0$. a, b, et c peuvent donc être bien plus grands que pour les arêtes initiales, de "première génération"; |a|, |b| peuvent être de l'ordre de G^5 au lieu de G, et |c| peut être de l'ordre de G^6 au lieu de G^2. Bien sûr, rien n'interdit à ces arêtes de "deuxième génération" de s'intersecter en de nouveaux sommets, qui peuvent être les extrêmités d'arêtes de troisième génération... La taille des nombres nécessaires croît de façon explosive et il n'est plus possible d'admettre qu'une opération se fait en un temps constant. L'algorithme naïf, qui peut travailler avec une arithmétique flottante, finira par être plus rapide : son efficacité ne dépend pas de G.

L'algorithme de BENTLEY-OTTMAN est donc "sauvé" de l'imprécision, mais de justesse, et à quel prix : notre méthode ne résoud plus le même problème que l'algorithme original :

Initialement, l'algorithme de BENTLEY-OTTMAN détermine les arêtes intersectantes parmi N arêtes données, dont les sommets ont des coordonnées réelles; il existe bien sûr une infinité non dénombrable d'ensembles de N segments, vue la nature continue du problème. L'algorithme modifié détermine les arêtes intersectantes parmi N arêtes données, dont les sommets ont des coordonnées entières appartenant à $[0,G]^2$. Par un dénombrement trivial, cet univers ne compte plus que $S= (G+1)^2((G+1)^2 - 1)/2$ arêtes distinctes; il n'existe donc plus que 2^S problèmes avec N arêtes distinctes ($0 \leq N \leq S$), et non une infinité non dénombrable. Ces différences n'ont aucune portée pratique (il est exclu de précalculer la solution de tous les problèmes possibles !) mais elles montrent bien que la nature du problème traité a changé. L'algorithme modifié n'exploite pas la nature foncièrement discrète du nouveau problème traité, sauf bien sûr pour l'arithmétique. On peut se demander s'il n'existe pas une meilleure méthode que celle de BENTLEY-OTTMAN, qui exploiterait cette spécificité.... Une méthode de ce type est proposée dans le chapitre III, pour un autre problème.

♦ **Remarque** GREENE et YAO [GREE 86] proposent une méthode qui recale au mieux sur les points entiers du plan les points d'intersection d'un ensemble donné de segments dont les sommets ont des coordonnées entières. Pour des raisons différentes des nôtres, GREENE et YAO utilisent eux-aussi une arithmétique rationnelle avec les algorithmes de BENTLEY-OTTMAN et de MAIRSON-STOLFI [MAIR 83]. Ils procèdent ainsi non parce qu'ils estiment que l'algorithme de BENTLEY-OTTMAN peut ne pas fonctionner avec une arithmétique flottante, mais seulement parce qu'une arithmétique flottante ne donne que des résultats approchés. Or GREENE et YAO ont besoin de connaître les coordonnées des points d'intersec-

tion avec une totale précision.

5 COLORIAGE DES CARTES PLANAIRES

Cette section traite du coloriage d'une carte planaire sur un terminal graphique ou sur une mémoire d'images.

Le paragraphe 5.1 expose une première méthode d'affichage colonne par colonne : les bords sont supposés d'une couleur unie, sans dégradés ni textures; cette méthode ne prend pas non plus en compte les effets de marches d'escalier aux frontières des zones. Elle est aussi utilisée par ROMMEL [ROMM 87] pour la modélisation des terrains : elle permet de passer d'une description par des courbes de niveau à la description par une grille altimétrique.

Le paragraphe suivant, 5.2, généralise l'algorithme d'affichage et résoud ses problèmes d'aliassage : 5.2.1 montre comment adapter la méthode classique de suréchantillonnage global au coloriage des cartes planaires, et 5.2.2 propose une méthode nouvelle de suréchantillonnage local. 5.2.3 montre comment prendre en compte le filtrage des textures des bords internes de la carte planaire.

5.1 AFFICHAGE SANS FILTRAGE

La méthode affiche une partie rectangulaire $[x_0..x_1]$ [$y_0..y_1$] de la carte planaire. Elle détermine l'ordre des arêtes coupant l'axe de chaque colonne d'image; comme la carte planaire connait les deux bords (et leurs couleurs) de chaque arête, l'affichage d'une colonne est ensuite chose facile. Les ordres des arêtes dans chaque colonne sont déterminés lors d'un balayage de toute la carte, de gauche à droite, selon l'xy-ordre.

Ce balayage est plus efficace que celui de l'algorithme de BENTLEY-OTTMAN : les tests d'intersection et les tris sont inutiles.

L'y-ordre sur la droite de balayage peut être efficacement représenté par une liste doublement chainée; de plus, sur chaque arête est notée sa position dans la liste de l'y-ordre. La suppression d'une arête morte se fait alors en O(1). L'insertion d'une nouvelle active se fait aussi en O(1), même dans le cas de la première incidente d'un sommet de naissance; en effet la butée de ces sommets est connue, et elle précède immédiatement les naissantes du sommet dans l'y-ordre. Comme insertion et suppression se font en O(1), le balayage lui même se fait en O(A), A étant le nombre d'arêtes de la carte. Soit avancer-barre(), la procédure qui fait franchir à la barre de balayage le sommet situé immédiatement à sa droite; cette procédure extrait de la barre de balayage les mourantes au sommet, et insère les naissantes; le pseudocode suivant décrit alors l'algorithme d'affichage :

```
affichage =
{
  initialiser les matrices de transformation : image<->carte;
  initialiser le terminal graphique;
  initialiser la droite de balayage comme vide;
  sommet-à-passer ← le premier sommet;

  pour xécran de 1 à C /* l'écran a L lignes, C colonnes */
    { xcarte ← xécran dans le repère de la carte;
      tant que (sommet-à-passer existe)
            et ( abscisse (sommet-à-passer) < xcarte)
      faire avancer-barre() ;
      afficher-colonne();
    }
}
```

♦ **Remarque** La barre de balayage qui permet de générer une colonne d'image ne contient jamais d'actives verticales.

La procédure afficher-colonne() génère une colonne d'image à partir de la barre de balayage. Elle parcourt la liste de l'y-ordre, et calcule les ordonnées (qui peuvent être calculées incrémentalement) des points d'intersection entre les actives et la droite de la barre de balayage. Une liste de "plages" est obtenue; une "plage" est l'intervalle entre deux arêtes actives le long de la barre de balayage. Le fenêtrage à une seule dimension des plages verticales par l'écran est évident.

Pour chaque colonne d'image, la liste de l'y-ordre doit être parcourue depuis le début, même si la partie de la carte à afficher commence beaucoup plus haut. Une variante de la même méthode entretient lors du balayage un pointeur sur la première arête active dans la fenêtre. Ce pointeur est remis à jour quand l'arête pointée meurt, et avant de générer la colonne d'image; quand le pointeur est incorrect, la première arête active est en général une proche voisine de l'arête pointée dans la barre de balayage.

Le balayage de la carte est en O(A); soient L et C le nombre de lignes et de colonnes de l'écran ou de la mémoire d'images; la génération des plages est en O(CA); l'affectation des LC pixels est en O(LC); d'où une complexité finale de l'affichage en O(CA+LC). Cette expression peut être trompeuse, car les constantes implicites des termes CA et LC peuvent être très disparates. Ainsi, sur la configuration matérielle disponible à l'ENSMSE (HP9000, mémoire d'images LEXIDATA SOLIDVIEW de 640x512 pixels), l'affichage d'une carte vide exige 6 secondes (!), et celui des cartes de 5000 à 6000 arêtes, 20 secondes : le terme LC a une constante implicite très forte.

Le paragraphe précédent ne parle pas des problèmes d'imprécision pour l'affichage; et pour cause : l'imprécision ne pose alors aucun problème. Il est intéressant de comprendre pourquoi.

Passons sur la gestion du balayage (ordre des sommets, insertions et extractions), puisqu'elle ne nécessite aucun calcul; les seuls calculs effectués sont ceux des ordonnées des points d'intersection entre les arêtes actives et la droite de la barre de balayage. Ces calculs sont effectués en flottant; soient $y_1 < y_2$ ..., les ordonnées correctes; soient y^*_1, y^*_2...., les ordonnées calculées correspondantes. L'algorithme d'affichage colorie successivement les pixels $[y^*_1]$ à $[y^*_2]$ avec la couleur c_1, $[y^*_2]$ à $[y^*_3]$ avec la couleur c_2, etc; supposons qu'avec l'imprécision, $[y^*_1] < [y^*_3] < [y^*_2]$; l'affichage terminé, les pixels $[y^*_3]$ à $[y^*_2]$ seront alors de couleur c_3, au lieu de c_2, la couleur correcte. Mais cette erreur n'aura pas plus de conséquence sur l'algorithme d'affichage que l'aliassage; les résultats intermédiaires, corrects ou non, ne modifient pas le déroulement de l'algorithme d'affichage.

5.2 AFFICHAGE AVEC FILTRAGE

L'affichage précédent n'est pas sans défaut : il provoque l'apparition de marches d'escalier aux frontières des différentes zones; les zones longues et plus minces qu'un pixel apparaissent en pointillé.

Il s'agit là d'une manifestation classique d'aliassage [CROW 76] [CROW 81] [GHAZ 85] : ce phénomène est bien connu en théorie du signal, sous le nom de repliement (ou recouvrement) du spectre. Il est dû à l'échantillonnage insuffisant d'un signal; le signal est ici un signal bidimensionnel : la scène 2D.

Des méthodes d'antialiassage sont maintenant présentées :

5.2.1 Suréchantillonnage global

Premier remède au crénelage des frontières, un échantillonnage vertical correct; il n'est pas suffisant mais toujours nécessaire.

Un échantillonnage vertical correct suppose que l'on n'arrondisse pas les extrêmités des plages verticales sur la grille d'affichage. Ainsi, le pixel traversé par p plages verticales : la plage de couleur c_1 sur une longueur l_1, la plage de couleur c_2 sur une longueur l_2 ..., et la plage de couleur c_p sur une longueur l_p, doit être affecté d'une couleur $c = l_1c_1 + l_2c_2 + ... + l_pc_p$. c_i est un vecteur à trois composantes, dans l'espace des couleurs.

Cette précaution indispensable n'est hélas pas suffisante : elle ne filtre que dans une seule direction. Le remède le plus simple et le plus brutal est le suréchantillonnage global. Chacune des colonnes est divisée en k sous-colonnes; autrement

dit, chaque pixel est divisé en k tranches verticales d'égales épaisseurs; ces k sous-colonnes sont calculées par l'algorithme précédent; puis la valeur de chaque pixel de la colonne finale est obtenue par une moyenne simple des valeurs des k tranches correspondantes. Pratiquement, un facteur k=5 ou 6 est suffisant.

```
suréchantillonnage-global =
{
  initialiser les matrices de transformation : image<->carte;
  initialiser le terminal graphique;
  initialiser la barre de balayage comme vide;
  sommet-à-passer ← le premier sommet;

  pour xécran de 1 à C /*  l'écran a L lignes, C colonnes */
  { pour j de 1 à 1 faire pixel [j] = (0 0 0) ;
    pour sous-colonne de 1 à k faire
    { xcarte ←  xécran dans le repère de la carte;
      tant que (sommet-à-passer existe)
            et ( abscisse (sommet-à-passer) < xcarte)
      faire avancer-barre() ;
      ajouter à pixel [ ] la sous-colonne d'image
            correspondant à la barre de balayage;
    }
    afficher pixel [ ] sur l'écran;
  }
}
```

Cette méthode donne d'excellents résultats graphiques; elle filtre correctement arêtes, sommets et petits objets ou objets minces; elle est donc tout à fait générale.

Pour les plus grosses cartes (5000 à 6000 arêtes) testées, l'affichage avec suréchantillonnage global (k=6) exige environ 120 secondes, au lieu de 20 pour l'affichage sans antialiassage.

GHAZANFARPOUR-KHOLENDJANY et MOISSINAC ont développé une variante de ce suréchantillonnage global, où les traits ont une épaisseur [MOIS 84], et où l'intérieur des bords est texturé et antialiassé [GHAZ 85].

♦ **Remarque** Un seul défaut a été constaté sur les images produites : un trait sombre apparaît sur la frontière entre certaines couleurs, entre rouge et bleu d'une part, entre rouge et vert d'autre part. Ce défaut n'est pas imputable à la méthode d'affichage, mais au mode d'interpolation entre les couleurs : le principe, implicite, définissant la couleur d'un pixel comme $C = \Sigma S_i C_i$, la couleur C_i occupant une surface d'aire S_i du pixel, est correct seulement dans certains repères. Ce n'est pas le cas pour le repère RVB (Rouge Vert Bleu), où les calculs sont effectués: ce repère est en effet le plus commode car c'est très souvent celui des mémoires d'images, mais il ne tient pas du tout compte des caractèristiques physiologiques de l'oeil; ce dernier ne perçoit pas les couleurs de façon linéaire : ainsi la couleur (r v b) n'est

pas perçue comme ayant la même teinte que la couleur (r/2 v/2 b/2). L'expérience suivante met bien ce phénomène en évidence : dans une première partie d'un écran graphique, est affiché un damier; chaque case du damier est un pixel, alternativement (r_1 v_1 b_1) et (r_2 v_2 b_2). L'oeil perçoit une zone de couleur unie : c'est la couleur que devrait déterminer le calcul d'interpolation, pour un pixel couvert sur une moitié de sa surface par (r_1 v_1 b_1), et par (r_2 v_2 b_2) sur l'autre moitié. La deuxième partie de l'écran est coloriée avec la couleur 0.5 (r_1+r_2 v_1+v_2 b_1+b_2) : c'est la couleur trouvée par le calcul d'interpolation. En règle générale, les deux parties de l'écran ont des couleurs visuellement différentes. TURKOWSKI [TURK 86] présente des méthodes visuellement correctes d'interpolation des couleurs.

5.2.2 Suréchantillonnage local

Le suréchantillonnage global est brutal, car tous les pixels, même ceux où il ne se passe rien, sont suréchantillonnés et filtrés. Un suréchantillonnage local est maintenant présenté. Il traite l'aliassage sur les sommets, les arêtes, les petits objets ou les objets minces : il obtient donc les mêmes résultats graphiques que le suréchantillonnage global. Mais il ne considère qu'une seule fois les pixels "sans histoire", en factorisant certaines opérations du suréchantillonnage global; il est donc plus efficace.

Les définitions suivantes deviennent nécessaires :

• Une tranche est l'intersection d'un pixel et d'une sous-colonne. Un pixel est donc divisé en k tranches verticales d'égales épaisseurs. Un pixel n'a pas à être divisé horizontalement.

• Une plage (déjà définie en 5.1) est l'intervalle entre deux arêtes actives successives (on dit aussi une "*span*").

• Une sous-plage est l'intersection entre une tranche et une plage.

• Une tranche est dite unie si et seulement si elle est ne contient qu'une seule sous-plage.

• La couleur sortante d'une tranche est la couleur de la sous plage en haut de la tranche.

• Un pixel est uni si et seulement si chacune de ses k tranches est unie; attention : un pixel peut donc être uni et avoir des tranches de couleur différentes.

• L'énergie est le produit d'une couleur et d'une aire; un pixel a une aire de 1.0; on confondra énergie et couleur dans le cas des pixels, par abus de langage; rappelons qu'une couleur est un vecteur de trois intensités.

• Le pixel p est dit précédent du pixel p+1 de même colonne.

• L'énergie sortante d'une tranche est le produit de l'aire de la tranche (1.0/k) par la couleur sortante de la tranche.

• Chaque pixel a un "réservoir"; ce réservoir contient l'énergie des sous-plages

des tranches non unies du pixel; ce réservoir est initialement vide, avant le calcul de la colonne d'image, et le reste si et seulement si le pixel est uni.

Ces définitions impliquent :

La couleur d'un pixel, de tranches unies u_1, ...u_p, et de tranches non unies n_1, ...n_q, est la somme de son réservoir et de l'énergie sortante des sous tranches n_1, ...n_q du pixel précédent.

L'algorithme de suréchantillonnage local exploite cette propriété. Les plages sont d'abord tracées dans une certaine structure de données, présentée au paragraphe suivant; puis la colonne finale est déduite par un parcours de cette structure, du pixel le plus bas (pixel [1]) au pixel le plus haut (pixel [L]).

Formellement, une colonne est un tableau [1..L] de pixels; un pixel est un enregistrement du type :

```
( uni: booléen;
  réservoir: tcouleur;  /* ne sert que si uni est faux */
  tranche: tableau [1..k] d'enregistrements du type :
          ( unie: booléen;
            énergie_sortante: tcouleur /* ne sert que si unie est faux */
          )
).
```

Pour tracer une plage dans une sous-colonne, le suréchantillonnage global avait besoin de considérer tous les sous pixels traversés. Par contre, dans cette structure de données, seuls sont considérés les pixels début et fin de la plage :

```
constante ep=1.0/k; /* épaisseur d'une tranche */

procédure noter_plage (Y0, Y1: réel ; rvb : tcouleur; g : 1..k ) =
{
 /* Y0 < Y1 */
 y0 ←partie_entière(Y0); y1 ←partie_entière(Y1);
 si (y0=y1)
 alors { /* pixel début et pixel fin sont égaux */
        pixel [y0].uni ←faux;
        aire ←ep*(Y1-Y0);
        pixel [y0].réservoir ← pixel [y0].réservoir + aire*rvb ;
        pixel [y0].tranche [g].unie ←faux;
       }
 sinon { /* mise à jour du pixel début */
        pixel [y0].uni ←faux;
        aire ← ep*(y0+1-Y0);
        pixel [y0].réservoir ← pixel [y0].réservoir + aire*rvb ;
        pixel [y0].tranche [g].unie ← faux;
        pixel [y0].tranche [g].énergie_sortante ← ep*rvb ;

        /* mise à jour du pixel fin */
        pixel [y1].uni ← faux;
        aire ← ep*(Y1-y1);
        pixel [y1].réservoir ← pixel [y1].réservoir + aire*rvb ;
        pixel [y1].tranche [g].unie ← faux;
       }
}
```

♦ **Remarque** Certaines multiplications (par ep notamment) peuvent être factorisées; ces optimisations ne sont pas prises en compte pour simplifier l'exposé.

Quand toutes les plages des k sous-colonnes d'une colonne ont été "tracées" dans la structure de données : pixel [ ], un parcours de pixel [1] à pixel [L] détermine les couleurs finales des pixels de cette colonne. Ce balayage gère deux variables: une variable teinte, qui mémorise la couleur du pixel actuellement traité, si celui- ci est uni; et une variable sous_teinte [1..k] qui mémorise les énergies sortantes des k tranches du pixel actuellement traité. Ces deux variables doivent être mises à jour à chaque fois qu'un pixel p rencontré au cours du balayage est non uni : la variable pixel [p] contient alors toutes les informations nécessaires.

```
const Pixel_vide.uni=vrai;
      Pixel_vide.réservoir=(0 0 0) /* le noir */ ;
      Pixel_vide.tranche [1..k].unie=vrai;

générer_colonne() =
{
 variable fin [1..L] : tcouleur;  /* couleurs finales */
         teinte, sous_teinte [1..k] : tcouleur;

 pour p ← 1 à L faire
 {
   si (pixel [p].uni)
   /* pixel [1] ne peut être uni, et aucune de ses tranches ne l'est */
   alors fin [p] ← teinte
   sinon { coul_p ← pixel [p].réservoir;
           pour g ←1 à k faire
               si (pixel [p].tranche[g].unie)
               alors  coul_p ← coul_p + sous_teinte [g]
               sinon  sous_teinte [g]  ← pixel [p].tranche [g].énergie_sortante;
           fin [p] ←coul_p;
           si (p < L et pixel [p+1].uni)
           alors { teinte ← (0 0 0) /* le noir */;
                   pour g ← 1 à k faire teinte ← teinte + sous_teinte [g];
                 }
           /* réinitialisation du pixel, pour la colonne suivante : */
           pixel [p] ←  Pixel_vide;
         }
 }
 afficher la colonne fin [ ];
}
```

♦ **Remarque** Le premier pixel et ses tranches ne peuvent être unis. Les divisions par k, facteur de suréchantillonnage, peuvent bien sûr être factorisées.

Comme dans la méthode précédente, une valeur de 5 ou 6 pour k donne d'excellents résultats, et un facteur supérieur est inutile. Le suréchantillonnage local est deux à trois fois plus rapide que le suréchantillonnage global, selon les cas. VAN-THONG [VANT 87] a ultérieurement optimisé la programmation de ce suréchantillonnage local : il n'utilise plus aucun calcul flottant, et fait des économies substantielles de mémoire.

5.2.3 Affichage et filtrage des textures et des dégradés

La carte planaire permet donc d'antialiasser efficacement et simplement toutes les frontières, et d'éviter ainsi les marches d'escalier ou le phénomène d'apparition et de disparition des petits objets longs et minces.

L'aliassage à l'intérieur d'une zone texturée se manifeste typiquement par des moirés. Cet aliassage a des solutions spécifiques, discutées dans [FEIB 80] [CROW 81] [WILL 83] [GANG 84b] [CROW 84] [GHAZ 85]....

D'un point de vue théorique, un algorithme d'antialiassage de textures peut être "encapsulé" dans une fonction du type : couleur_du_pixel (ipixel, jpixel, telle_texture, telle_transformation); tous les précédents algorithmes d'affichage d'une carte planaire colonne par colonne peuvent alors être repris tels quels pour des cartes texturées : il leur suffit d'appeler cette fonction, au lieu d'affecter toujours la même couleur aux pixels d'un bord. Ceci est permis par la totale indépendance de principe des deux problèmes : antialiassage sur les frontières effectué par la carte planaire, et antialiassage à l'intérieur d'une zone texturée effectué par la fonction couleur_du_pixel().

Le même raisonnement peut être fait pour les zones avec des dégradés de couleur, dus par exemple à des lissages de PHONG ou de GOURAUD : un calcul de dégradé en un pixel peut lui aussi être encapsulé dans une fonction externe : dégradé (i, j, tel_dégradé); comme les dégradés ne contiennent pas de hautes fréquences, cette fonction n'a pas d'aliassage à corriger; par contre, elle doit prendre en compte les bandes de MACH qui peuvent nuire à la qualité visuelle du dégradé.

6 CONCLUSION, PROBLEMES OUVERTS

6.1 IMPRECISION ET INCOHERENCE NUMERIQUE

Ce chapitre a montré l'importance de l'effet des imprécisions numériques :

• La configuration de [GANG 87] (cf. 4.1.1) montre que certaines cartes planaires ne peuvent être correctement modélisées avec l'arithmétique flottante. Il ne s'agit pas seulement d'une approximation sur les coordonnées, mais d'une incohérence : des sommets distincts sont affectés de coordonnées égales. Ce phénomène se généralise aux scènes 3D de polyèdres. Il est largement méconnu en modélisation géométrique, sauf dans [GANG 87], et peut-être [TAKA 86].

• Le deuxième constat est lui aussi nouveau : l'imprécision de l'arithmétique flottante rend inutilisable l'algorithme de BENTLEY-OTTMAN; l'exploitation de la cohérence du plan est en effet incompatible avec l'imprécision numérique. Des méthodes moins efficaces résistent beaucoup mieux que l'algorithme de BENTLEY-OTTMAN aux imprécisions numériques; c'est le cas de l'algorithme naïf, et des méthodes de la synthèse d'images, comme le tampon de profondeur, le lancer de rayons, l'algorithme de WATKINS ou l'algorithme d'ATHERTON.

• Les contre-exemples montrant la non résistance de l'algorithme de BENTLEY-OTTMAN à l'imprécision s'appliquent bien sûr aux autres méthodes qui emploient cet algorithme, par exemple [SZIL 84] [OTTM 82] [OTTM 82b] [OTTM 85]....

• L'arithmétique rationnelle permet de résoudre les problèmes d'imprécision numérique des problèmes affines. Son coût doit cependant être pris en compte lors de l'évaluation des performances des algorithmes. Sous certaines conditions, l'efficacité de l'algorithme de BENTLEY-OTTMAN avec arithmétique rationnelle n'est pas modifiée, mais la nature du problème traité devient discrète, au lieu de continue.

• L'exemple de l'algorithme de BENTLEY-OTTMAN peut provoquer quelques inquiétudes sur le bon fonctionnement des algorithmes géométriques qui exploitent les propriétes de la relation d'ordre et effectuent des comparaisons sur des grandeurs calculées grâce à une arithmétique flottante. Les algorithmes géométriques (élimination des faces cachées, calcul de l'enveloppe convexe, intersection de polygones ou de polyèdres...) sont prouvés dans R^n, et leur efficacité est évaluée en supposant que les opérations réelles s'effectuent en un temps constant, comme dans l'arithmétique flottante. Mais le cas de la méthode de BENTLEY-OTTMAN prouve qu'il ne suffit pas de démontrer la validité d'un algorithme géométrique dans R^n pour que cet algorithme fonctionne correctement en machine; il faut aussi démontrer que l'imprécision numérique ne nuit pas au bon déroulement de l'algorithme. Quand la méthode ne résiste pas à l'imprécision, elle doit utiliser une arithmétique parfaitement précise, dont le surcoût peut modifier les performances théoriques de l'algorithme.

La prise en compte des phénomènes de l'imprécision numérique par la géométrie algorithmique me parait être une voie de recherche prometteuse; de plus, elle ne pourra que renforcer l'impact pratique de cette discipline.

6.2 LES CARTES PLANAIRES

La structure de données de carte planaire a été présentée, ainsi qu'un algorithme efficace de construction. Un exemple d'utilisation des cartes planaires est l'affichage de scènes 2D avec antialiassage. Les cartes planaires, ou des variantes, sont beaucoup utilisées dans la suite de cette thèse; ainsi, le chapitre suivant propose un algorithme d'élimination des faces cachées : les images produites sont des cartes planaires.

Cette conclusion évoque maintenant quelques limitations des cartes planaires, et propose quelques voies de recherche.

6.2.1 Rotation d'une carte planaire

Dans le principe, l'emploi des nombres rationnels permet de résoudre les problèmes d'imprécision et d'incohérence. Cependant, les nombres rationnels sont insuffisants dans certains cas. Ainsi, certaines applications (palette graphique, ou modeleur 2D) effectuent des rotations sur une partie des arêtes d'une carte planaire, et recalculent une nouvelle carte. Un calcul exact est alors impossible avec les rationnels; par exemple, une rotation de 45 degrés fait apparaître des termes $\sqrt{2}$.

Il est tentant d'effectuer la rotation des coordonnées en calcul flottant, de recaler sur la grille entière les points obtenus, et enfin de repasser en arithmétique rationnelle pour le calcul de la nouvelle carte. Cependant cette "rotation flottante" modifie parfois la topologie; par exemple, un sommet proche d'une arête passera "de l'autre coté" de cette arête; GREENE et YAO [GREE 86] donnent quelques exemples de changements de topologie provoqués par le calage de points d'intersection sur une grille entière; or ces perturbations surgissent aussi avec le recalage sur la "maille" des nombres flottants. Cela peut bien sûr poser des difficultés à certaines applications qui supposent (légitimement) qu'une rotation ne doit pas changer la topologie. Quatre solutions sont alors envisageables.

La première solution n'utilise que des "rotations rationnelles", ie des rotations dont le cosinus et le sinus de l'angle sont rationnels. Soient a/c et b/c le cosinus et le sinus de l'angle d'une rotation rationnelle; ils vérifient bien sûr $a^2 + b^2 = c^2$; on rappelle que les solutions sont toutes de la forme : $a=m^2-n^2$, $b=2mn$, $c=m^2+n^2$, avec m, n décrivant N. Cette solution "recale" donc les rotations réelles sur des rotations rationnelles.

La deuxième solution est une variante de la précédente; cette fois, la matrice de rotation est recalée sur une matrice rationnelle, qui n'est plus forcément celle d'une rotation.

La troisième solution représente fidèlement les "rotations algébriques", d'angle $k\pi$, où k est rationnel : les fonctions trigonométriques de ces angles sont en effet des nombres algébriques; le chapitre 4 étudie la représentation exacte de ces nombres. Mais il devient alors un peu absurde de restreindre les scènes 2D à des configurations de segments; aussi le chapitre 4 traite-t-il de la représentation exacte des courbes algébriques; il propose une "carte planaire algébrique".

Quatrième solution : ne plus admettre que la topologie est invariante par rotation ! Plus généralement, cette solution accepte et assume l'imprécision des calculs; elle ne peut utiliser la cohérence qu'avec beaucoup de prudence; en particulier, elle ne peut employer la méthode de BENTLEY-OTTMAN. Ce genre d'approche est pratiquée par TAKALA [TAKA 86] pour la modélisation solide.

6.2.2 Carte planaire et interactivité

La construction de la carte planaire est effectuée ex nihilo; elle ne se fait donc pas par une succession de modifications incrémentales d'une carte planaire déjà existante, modifications qui pourraient être l'ajout ou la suppression d'une ou plusieurs arêtes. Cette fonctionnalité serait pourtant bien utile, quand les cartes sont employées dans des situations interactives.

Le problème

Bien sûr, il est toujours possible de recalculer toute la carte; mais alors l'ajout d'une seule arête n'est pas plus efficace que la construction ex nihilo. Une solution brutale peut cependant être envisagée pour la construction de la CP-globale à partir de la CP-locale : cette phase est très rapide relativement à l'algorithme de BENTLEY-OTTMAN, et compatible avec l'interaction.

Citons quelques-uns des difficultés rencontrées :

Considérons d'abord la suppression d'une ou de plusieurs arêtes : la suppression des brins dans la CP-locale est triviale; la principale difficulté me semble être posée par la mise à jour des butées; certaines arêtes supprimées peuvent en effet être les butées de certains sommets de naissance; une méthode efficace pour retrouver ces sommets et mettre à jour leur butée n'est pas évidente; de même, la suppression d'arêtes peut transformer certains sommets en sommet de naissance : il faut alors déterminer leur butée.

Peut être est-il plus sage de reporter la suppression de ces arêtes et de les marquer ? Ainsi les butées n'ont pas à être remises à jour; les seules modifications concernent la CP-globale : d'une façon ou d'une autre, il faut fusionner les bords contigus séparés par une arête marquée "supprimée". Quand la carte contient "beaucoup" d'arêtes à supprimer, une reconstruction brutale est effectuée.

Autre solution possible : renoncer à l'utilisation des arêtes de butée...

L'ajout de nouvelles arêtes est encore plus difficile; non seulement des butées peuvent être modifiées, mais les points d'intersection doivent être déterminés. Enfin, l'ajout pose des problèmes arithmétiques : la taille des entiers nécessaires augmente très vite (cf 4.5.2).

Ces difficultés suggèrent qu'une autre structure de données est peut-être préférable.

Une autre structure de données ?

D'autres structures de données décrivant des partitions "dynamiques" du plan ont-elles été proposées ?

Une structure de données dynamique décrivant des configurations du plan a été proposée par OVERMARS et LEEUWEN en 1981 [OVER 81]. Fonctionnalité typique de cette structure de données : elle permet d'entretenir efficacement l'enveloppe convexe d'un ensemble de points du plan, subissant des insertions et des suppressions dynamique; ou encore elle permet de tenir à jour le sous ensemble des points dominants d'un ensemble dynamique de points; (x y) "domine" (x' y') si et seulement si x>x' et y>y'. Mais pas un mot n'est dit sur les incohérences que peut provoquer l'imprécision des calculs dans le cas de l'enveloppe convexe. Surtout, cette structure n'offre pas du tout les mêmes fonctionnalités qu'une carte planaire.

Une structure de données plus récente, utilisée en particulier par AYALA, BRUNET, JUAN et NAVAZO [AYAL 85], est le quadtree exact, ou quadtree non minimal. Ce quadtree est plus compact que le quadtree traditionnel, et il est exact car il permet de retrouver une description des frontières. Ce quadtree est un arbre représentant un polygone; la racine est un carré contenant le polygone; si le contenu du carré est trop complexe, il est coupé en quatre parties égales, filles du carré initial. Il existe cinq types de noeuds terminaux :

- les feuilles-sommets, qui contiennent un seul sommet, et les arêtes incidentes.
- les feuilles-arêtes, qui sont traversées par une arête unique.
- les feuilles-blanches, qui sont totalement en dehors du polygone.
- les feuilles-noires, qui sont totalement à l'intérieur du polygone.
- les feuilles-pixels, qu'il est inutile de diviser encore.

AYALA et al. utilisent le quadtree exact pour réaliser les opérations booléennes entre les polygones; puis ils généralisent cette approche au cas tridimensionnel, pour la modélisation du solide (*"solid modeling"*).

SAMET et NELSON [SAME 86] utilisent aussi ce genre de structure de données pour la description de bases de données géographiques.

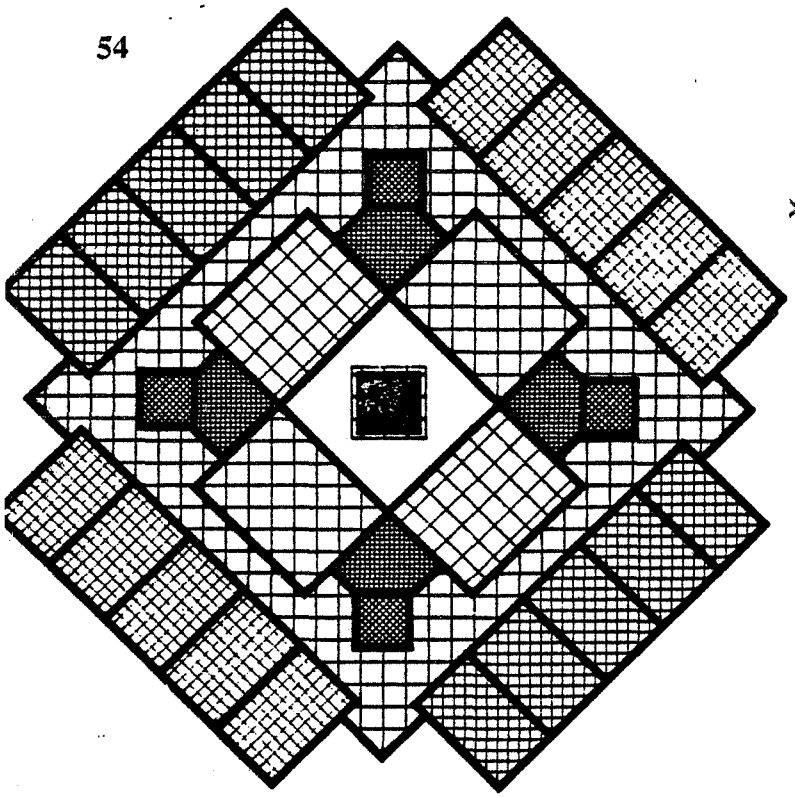
Les quadtree exacts permettent donc une description naturelle des partitions dynamiques du plan; peut être apportent-ils aussi une solution nouvelle au problème de l'imprécision numérique : on peut fort bien imaginer l'existence de quelques feuilles "indécidables", où l'on ne sait pas très bien ce qui se passe. Enfin, l'intro-

duction des courbes dans les quadtree exacts est a priori assez facile; il n'est pas nécessaire de savoir calculer explicitement les points d'intersection entre deux courbes, il suffit de savoir si une courbe passe ou non dans un carré donné; d'ailleurs, des techniques similaires ont déjà été employées en 3D par KUNII [KUNI 85] et GLASSNER [GLAS 84] pour optimiser le lancer de rayon.

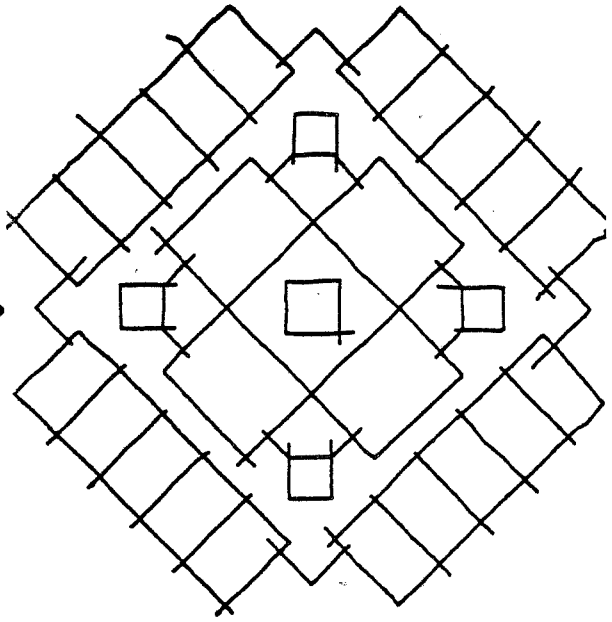
Les quadtree exacts ne sont cependant pas utilisés dans la suite de cette thèse, qui ne traite ni de l'interactivité, ni des partitions dynamiques du plan. Hors de ce cadre, les quadtree exacts semblent en effet moins intéressants : j'ai écrit un logiciel d'élimination de parties cachées utilisant une variante des quadtree exacts [MICH 86], mais ses performances sont nettement moins bonnes que celles de l'algorithme exposé au chapitre II, qui utilise les cartes planaires.

7 ANNEXE

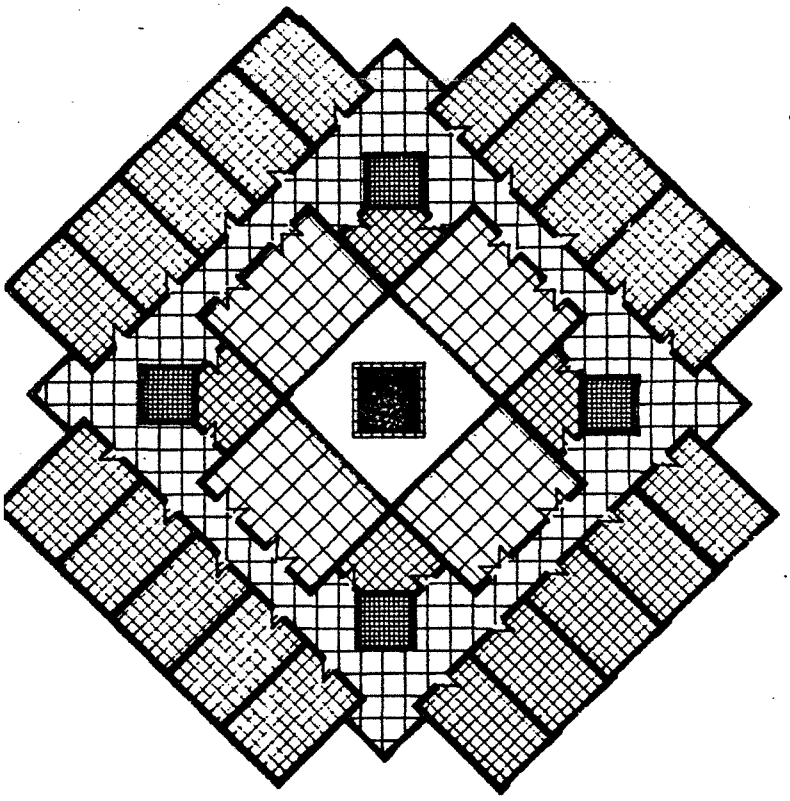
Les figures de la page suivante illustrent le tout premier emploi des cartes planaires à l'ENSMSE : des esquisses (ici d'architecture) sont saisies à main levée sur une table à numériser; il est possible de recaler les directions du tracé; une carte planaire est construite; elle permet de reconnaitre et d'enlever les arêtes pendantes; diverses désignations et instrumentations (épaississement des arêtes, coloriage des zones, ajout d'ouvertures) sont ensuite possibles.



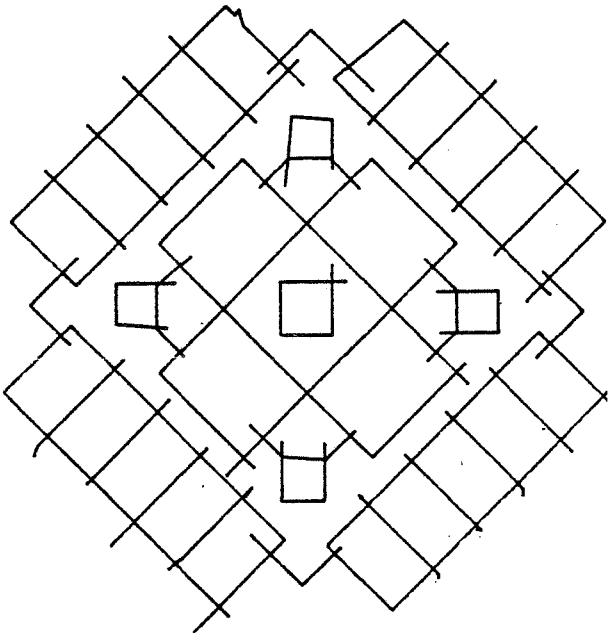
Carte des nus.



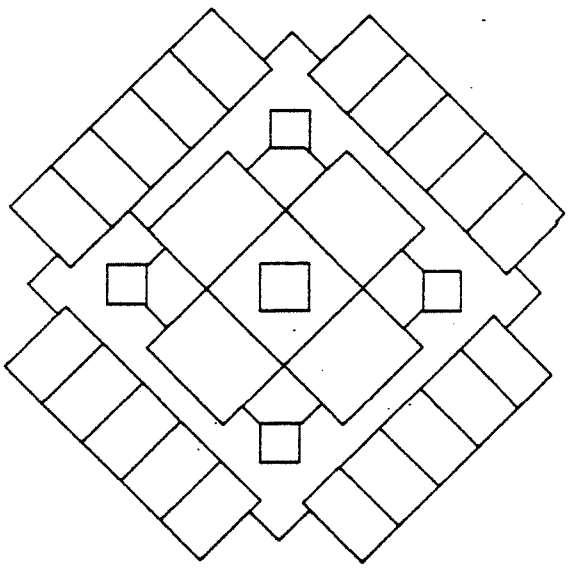
Tracé initial.



Ouvertures.



Après échantillonnage et recalage



Suppression des arêtes pendantes.

Chapitre 2

ELIMINATION DES PARTIES CACHEES SUR DES ARBRES CSG

1 INTRODUCTION

La modélisation de scènes 3D à l'aide d'un arbre CSG semble bien adaptée à la description des pièces de la CAO en mécanique; rappelons que dans un arbre CSG ("*Constructive Solid Geometry*" ou arbre de construction), les noeuds sont des transformations géométriques affines ou des opérateurs booléens, et que les feuilles sont des solides géométriques élémentaires (polyèdres, quadriques, etc...).

Les algorithmes de visualisation les plus populaires, comme le tampon de profondeur ("*z-buffer*") par exemple, nécessitent une connaissance explicite des contours. Or la détermination des frontières d'un objet modélisé par un arbre de construction est un problème délicat. C'est un des éléments qui explique le recours à d'autres modes de représentation, comme l'octree, ou à d'autres algorithmes de visualisation, comme le lancer de rayons. Hélas, ces techniques sont généralement très coûteuses en temps d'exécution ou en place mémoire sur les configurations actuelles, les machines dédiées à ces techniques n'étant pas encore disponibles.

D'autres algorithmes que le lancer de rayons permettent la visualisation des scènes modèlisées par des arbres CSG :

Un premier algorithme est celui d'ATHERTON [ATHE 83], qui généralise l'algorithme classique de WATKINS; il en a déjà été question en I.4.3. VAN THONG [VANT 85] a proposé un deuxième algorithme "*scan-line*", qui diffère du précédent par la solution apportée au problème bidimensionnel; les opérations booléennes sont effectuées par un emploi de cartes planaires. Une troisième méthode "*scan-line*" est due à PUEYO et MENDOZA [PUEY 87].

Un autre algorithme est celui de ROSSIGNAC et REQUICHA [ROSS 86] qui généralise le tampon de profondeur; les primitives de l'arbre CSG sont données par une inéquation algébrique $f(x,y,z) \geq 0$ et par leur "contour", une surface paramétrée en (u,v) : $x=X(u,v)$, $y=Y(u,v)$, $z=Z(u,v)$, $u,v \in [0,1]^2$. L'inéquation algébrique permet de savoir si un point (x y z) donné est intérieur, extérieur, ou sur le contour (à un epsilon près) d'une primitive de l'arbre CSG; une fonction récursive simple

peut alors calculer si un point est extérieur, intérieur, ou sur le contour de la scène; par exemple, un point P est sur le contour de A∪B si et seulement si P est sur le contour de A et non intérieur à B, ou si P est sur le contour de B et non intérieur à A; de même, un point P est sur le contour de A∩B si et seulement si P est sur le contour de A et non extérieur à B, ou si P est sur le contour de B et non extérieur à A; même principe pour A-B. Les formes paramétrées permettent de générer des points sur les contours des primitives : la méthode, particulièrement brutale, détermine un pas d'echantillonnage (du, dv), et calcule pour chaque sommet (u,v) de la trame rectangulaire ainsi définie les valeurs correspondantes (x y z). Le tampon de profondeur généralisé procède comme suit : les primitives sont traitées tour à tour, dans un ordre quelconque; pour chaque primitive, un ensemble de points (x y z) sur le contour est généré; ces points sont affichés dans le tampon en z et dans l'image si leur profondeur est inférieure à la profondeur courante du pixel correspondant, et s'ils appartiennent bien au contour de la scène. Cette méthode est simple, mais pas exempte de défauts : le pas d'échantillonnage (du, dv) doit être suffisament faible, sinon les surfaces affichées sont constellées de trous ou de pixels faux; l'imprécision du test d'appartenance à un contour provoque d'autres défauts, dans le cas de tangence ou de "quasi-tangence" des primitives. L'aliassage naturel du z-buffer est donc exacerbé; il semble que seul un suréchantillonnage global puisse venir à bout de ce défaut; mais calculer une image k^2 fois plus grande exige semble-t-il k^2 fois plus de temps et k^2 fois plus d'espace. Au passage, remarquons que cet algorithme arrive toujours à son terme, quelles que soient les erreurs d'imprécision qu'il commet : son déroulement est indépendant de ses résultats.

D'autres méthodes convertissent l'arbre de construction dans un autre type de représentation au cours de la visualisation; par exemple [KOIS 85] convertit l'arbre CSG en bintree; la conversion n'est pas effectuée pour les portions manifestement cachées de la scène.

GOLDFEATHER, HULTQUIST et FUCHS [GOLD 86] ont proposé une méthode de visualisation qui exploitent les propriétés matérielles de leur machine : le "*Pixel-Powers Graphics System*", qui peut évaluer, quasi simultanément, une forme quadratique f(x,y) en chaque pixel (x,y) de l'image. D'autres architectures matérielles ont été proposées spécialement pour la synthèse d'images, mais contrairement à [GOLD 86] elles emploient le plus souvent l'algorithme du lancer de rayons.

On peut noter que les méthodes qui précédent résolvent le problème avec la résolution de l'image.

Je n'ai pas trouvé trace dans la littérature d'algorithme exact pour la visualisation des scènes CSG; les seuls articles traitant d'élimination des parties cachées considèrent un ensemble de faces non intersectantes, et développent une généralisa-

tion [OTTM 82b] [MELH 86] de la méthode de BENTLEY-OTTMAN; le plan de l'image est ratissé par une droite de balayage, représentée par un segment-tree (arbre équilibré permettant la gestion d'intervalles, sorte de quadtree en 1D), qui permet de gérer l'ordre en profondeur des faces actives. L'algorithme a l'efficacité de BENTLEY-OTTMAN. Il faut cependant remarquer que le présupposé des faces non intersectantes n'est guère réaliste. De plus, l'algorithme est confronté dans la pratique à de délicats problèmes d'imprécision numérique 2D (cf I.4) et 3D : l'ordre en profondeur d'une face se propage au cours du balayage; une erreur initiale minime (interversion de deux faces très proches) provoque des erreurs de plus en plus grandes, voire des incohérences qui font avorter le programme. Même en supposant résolus les problèmes d'imprécision, il ne semble pas que cet algorithme puisse être réutilisé pour des scènes CSG.

Ce chapitre présente un nouvel algorithme d'élimination des parties cachées pour les scènes CSG; il calcule une image de la scène en un temps compatible avec l'interaction, et la perspective résultante a une précision qui dépend du degré de facettisation des primitives géométriques de l'arbre de construction, mais pas de la résolution de la mémoire d'images. Cet algorithme permet donc des sorties au trait, ce qui est une exigence traditionnelle en CAO ou en architecture. L'image rendue est une carte planaire, et elle peut aussi être affichée sur une mémoire de trame par les algorithmes du paragraphe I.5.

Cet algorithme peut aussi être utilisé en synthèse d'images, en particulier lors de la production de séquences d'animation dans les environnements statiques, pour la détermination des parties d'une scène qui sont éclairées par une source lumineuse donnée. (A l'ENSMSE, BEIGBEDER, TALLOT, et moi-même produisons justement de telles séquences, où l'oeil se déplace dans divers projets d'architecture conçus par CHOPIN; pour l'instant, les ombres portées sont ignorées, mais elles apporteraient un bien plus grand réalisme aux images produites)

La prise en compte des ombres portées peut certes être assurée par d'autres méthodes : le lancer de rayons, mais il nécessite actuellement une trop grande puissance de calcul, ou bien une trop grande patience; la méthode de WILLIAMS [WILL 78] mais elle exacerbe l'aliassage; les volumes d'ombres de CROW [CROW 77] [BERG 86] dont l'emploi peut être généralisé aux scènes CSG; mais cette méthode est assez lente, et surtout le calcul des ombres portées doit être effectué pour chacune des nombreuses images de la séquence d'animation. Au contraire, l'algorithme proposé ici permet de calculer une image de la scène avec l'oeil à la place de la source : les polygones vus sont éclairés par la source; l'image (ou plutôt la carte planaire décrivant l'image) permet de les déterminer une fois pour toutes; ces polygones sont ensuite ajoutés à la description de la scène; puis les nombreuses images (plusieurs centaines, voire quelques milliers) de la séquence d'animation sont ensuite calculées par une quelconque méthode, par exemple celle

d'ATHERTON : notre algorithme permet donc de factoriser le calcul des ombres portées et le surcoût de la prise en compte des ombres portées devient tout à fait raisonnable.

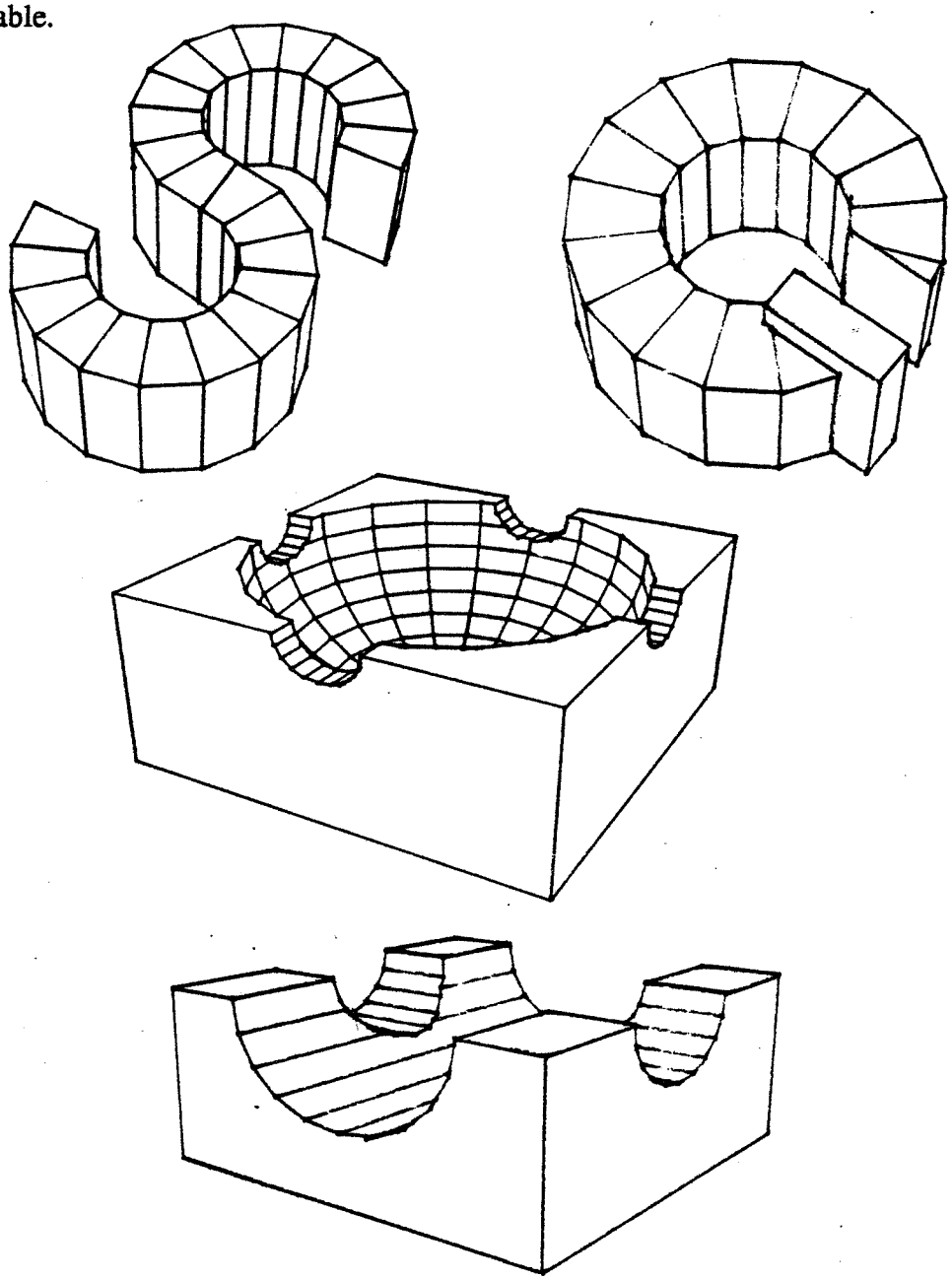


Fig. 1 : Quelques images d'arbres CSG

Le plan de ce chapitre est le suivant : les structures de données sont d'abord présentées au paragraphe 2. Le paragraphe 3 décrit ensuite **un algorithme de réduction** permettant de supprimer les faces qui, bien qu'appartenant à une des feuilles de l'arbre de construction, ne font pas partie de la scène. Le paragraphe 4

propose **un algorithme d'élimination des faces cachées** proprement dit. Le paragraphe 5 étudie la complexité théorique des divers algorithmes proposés. Le paragraphe 6 envisage diverses optimisations possibles. La fin de ce chapitre insiste sur les difficultés d'implantation de l'algorithme, notamment les problèmes d'imprécision.

Ces méthodes ont fait l'objet d'un article de PEROCHE et moi-même [MICH 87], dont s'inspire ce chapitre.

2 STRUCTURES DE DONNEES

Les primitives géométriques des feuilles de l'arbre de construction sont supposées être facettisées; la projection des arêtes des diverses primitives de la scène sur le plan Oxy de l'écran forme un graphe planaire.

Ce graphe est décrit par la structure de données de carte planaire, qui contient des sommets, des arêtes, des brins, des bords, toutes entités déjà définies au paragraphe I.2. La notion nouvelle de "zone" est introduite : une zone est constituée d'un bord interne et de ses bords fils; une zone est principalement caractérisée par une liste de faces actives : ce sont les faces des primitives qui se projettent en Oxy dans la zone.

De manière plus précise, la structure de données choisie est la suivante:

Une face est un enregistrement du type:

- (a, b, c, d), où l'équation de la face est : ax + by + cz + d = 0, la normale étant orientée; on précisera ultérieurement l'implantation du quadruplet (a, b, c, d);
- objet : un pointeur vers la primitive géométrique limitée par la face;
- arcs : une suite de triplets (x, y, z) qui définit le (ou les) contour(s) de la face; l'implantation exacte de arcs est ici sans importance.

Un sommet est un enregistrement du type:

- (x, y) : ses coordonnées dans le plan de l'écran. L'implantation exacte des coordonnées a été discutée en I.4.5.2;
- incidente : un pointeur vers la première arête de ce sommet;
- cote_incidente : 1 (pour gauche) ou 2 (pour droite), indique si le sommet est le premier ou le second sommet de l'arête pointée par incidente.

Une arête est un enregistrement du type:

- (a, b, c), où l'équation du support de l'arête dans le plan de l'image est : ax+by+c=0; l'implantation exacte du triplet (a, b, c) a été précisée en

I.4;
- brin : un tableau [1..2] de deux brins, le gauche et le droit.

Un brin est un enregistrement du type :

- point : pointeur vers le sommet du brin;
- voisine : pointeur vers l'arête du brin voisin de même sommet;
- cote_voisine : 1 ou 2, indique si le brin voisin est un brin un ou deux;
- fb : tête d'une liste de pointeurs sur les faces actives pour la zone de ce brin;
- zone : pointeur vers la zone du brin.

On rappelle que le brin d'indice un d'une arête rassemble toute l'information sur la partie gauche de l'arête, et sur la région le long et au dessus (dans le plan Oxy) de cette arête. Symétriquement, le brin d'indice deux rassemble les informations sur la partie droite de l'arête, et sur la région le long et en dessous (dans le plan Oxy) de cette arête.

Enfin, une zone est un enregistrement du type :

- aret_zone : liste de pointeurs vers des arêtes; le brin numéro deux de chacune de ces arêtes permet de parcourir un des bords de la zone; il y a une arête pour chaque bord de la zone;
- factives : liste de pointeurs vers les faces actives de cette zone.

3 ALGORITHME DE REDUCTION

Dans ce paragraphe, nous supposons effectuées les opérations suivantes :

- facettisation des primitives géométriques de l'arbre de construction; le degré de facettisation croît avec la taille de la primitive dans l'image;
- mise dans le repère de l'oeil de chaque primitive géométrique de l'arbre de construction;
- fenêtrage;
- éventuellement, transformation perspective.

L'algorithme de réduction comporte cinq étapes :

- initialisation des structures de données;
- recherche dans le plan de l'image des intersections entre les arêtes de toutes les faces de toutes les primitives de l'arbre de construction;
- détermination des zones et de l'ensemble des faces actives pour chaque zone;
- le long de chaque arête de chaque zone (c'est à dire pour chaque brin),

détermination de l'ordre en profondeur (selon l'axe Oz) de chaque face; si nécessaire, des arêtes d'intersection entre faces seront ajoutées;

- élimination des faces et des arêtes inutiles.

3.1 INITIALISATION DES STRUCTURES DE DONNEES

La structure de données est initialisée de la façon suivante :

```
{ pour chaque primitive géométrique de l'arbre de construction faire
  pour chaque face f faire
   pour chaque arc (px,py,pz) (qx,qy,qz) de la face faire
     { s1 ← le sommet de coordonnées (px, pz);
       s2 ← le sommet de coordonnées (qx, qz);
       /* les sommets sont créés s'ils n'existaient pas */
       a ← arête(s1, s2);
       /* l'arête est créée si elle n'existait pas */
       c ← /* 1 ou 2, selon le coté de a où se trouve f */
       ajouter à la liste fb du brin (a, c) une référence à f;
     }
}
```

A ce stade, les listes fb des brins ne comptent donc le plus souvent qu'une ou deux faces : une seule face, pour les arêtes appartenant à une face avant et à une face arrière d'une primitive géométrique de l'arbre de construction; et deux faces pour les arêtes appartenant à deux faces de même orientation (deux faces avant ou deux faces arrière); les listes fb ne contiennent donc pas encore toutes les faces actives. On remarquera qu'aucune liste fb n'est physiquement partagée entre plusieurs brins, et ceci devra rester vrai par la suite, tout au long de l'algorithme. Lors de l'initialisation, les intersections ou confusions partielles entre arêtes sont inconnues, et ignorées; par ailleurs les pointeurs des brins vers les zones sont initialisés à nil.

3.2 RECHERCHE DES INTERSECTIONS ENTRE ARETES

Dans cette phase, aucune information de profondeur n'est prise en compte : tout se passe dans le plan Oxy[1] de l'image. Il s'agit de déterminer les points d'intersection entre les arêtes; ces points d'intersection seront décrits par les mêmes structures de données que les sommets. L'algorithme utilisé est celui de BENTLEY-OTTMAN, déjà présenté en I.

---

1 Ce repère Oxyz est donc différent de celui employé en I.4.3; l'idée est que le "plan de travail" soit toujours le plan Oxy, et que le repère Oxyz doit être orthonormé direct, conformément à deux cents ans de tradition mathématique...

Les structures de données des sommets et des arêtes doivent bien sûr être mises à jour pour chaque intersection détectée. Dans le cas général, un nouveau sommet est créé, et toute arête intersectée, a_1, est scindée en deux: a_1 et a_2; les listes fb des brins de a_2 seront des copies physiques de celles de a_1. Dans le cas particulier où deux arêtes sont confondues, leurs listes fb sont simplement fusionnées dans un ordre quelconque.

Deux effets de bord intéressants de cet algorithme sont utilisés par la suite dans l'algorithme de réduction : d'une part, les sommets (initiaux ou points d'intersection) sont classés par l'algorithme suivant un ordre total; d'autre part, le balayage permet de déterminer l'arête située juste sous les sommets de naissance dans le plan de l'image; cette information, stockée sur chaque sommet de naissance, est utilisée lors de l'étape suivante : elle permet de situer les nouveaux contours par rapport aux anciens.

3.3 DETERMINATION DES ZONES

La procédure qui suit permet de parcourir les brins incidents à un sommet :

```
{ a ← S->incidente; c ← S->cote_incidente;
  faire { a2 ← a->brin[c].voisine ; c2 ← a->brin[c].cote_voisine;
          a ← a2 ; c ← c2 ; }
  tantque ( a ≠ S->incidente ) ;     }
```

Quand les arêtes n'ont plus aucun point d'intersection inconnu, une légère variante de la procédure précédente permet de parcourir tous les brins d'un bord (intuitivement, un "contour"), à partir d'un brin initial d'arête A et d'indice C :

```
{ a ← A; c ← C;
  faire { a2 ← a->brin[c].voisine ; c2 ← a->brin[c].cote_voisine;
          a ← a2 ;
          c ← 3 - c2 ; /* ceci permet d'obtenir l'autre coté */ }
  tantque ( a ≠ A ou c ≠ C ) ; }
```

Tous les brins d'un bord peuvent être retrouvés à partir de l'un d'entre eux; une zone est faite d'un bord interne et de ses bords fils; les brins d'une zone peuvent donc bien être retrouvés grace à la liste aret_zone de la zone, chacun des brins d'indice deux de cette liste d'arêtes permettant de trouver tous les brins d'un des bords de la zone. La troisième étape de l'algorithme crée les zones de l'image : pour chacune est déterminée sa liste aret_zone et sa liste de faces actives.

Une zone zone_mère est d'abord créée; sa liste des faces actives est vide. Cette zone est un cas particulier, car elle n'a pas de bord interne; elle contient tous les bords les plus externes et correspond à la "face infinie" des cartes planaires.

Ensuite, pour chacun des sommets S, selon l'ordre lexicographique, les arêtes A de sommet gauche S sont considérées tour à tour, selon l'ordre des brins. Deux cas sont possibles:

• **cas 1 :** le brin droit (numéro 2) de l'arête A a déjà une zone; on passe à l'arête suivante, ou bien au sommet suivant.

• **cas 2 :** le brin d'indice 2 de A n'a pas encore de zone; ce brin est la naissance d'un bord; deux cas sont possibles:

• **cas 2.a :** S est un point de naissance et A est la première incidente en S: le bord naissant est alors un bord externe et sa zone a déjà été créée; elle se retrouve ainsi : soit "sous", l'arête située juste sous S; si sous existe, alors la zone cherchée est la zone du brin droit de sous, sinon, la zone cherchée est zone_mère. "sous" est un souvenir du balayage de l'étape 3.2.

• **cas 2.b :** dans l'autre cas, le bord naissant est un bord interne; le brin traité est aussi la naissance d'une zone, que l'on crée.

Dans les deux cas 2.a et 2.b, les brins du bord naissant au brin un de l'arête A sont parcourus et leur pointeur zone est affecté; enfin, A doit être ajoutée à la liste aret_zone de la zone. Pour le cas 2.b, il reste à expliquer comment peut être déterminée la liste des faces actives de la zone naissante Z. L'arête A ne peut être, dans ce cas, la première arête du sommet S : soit A_2 l'arête qui précéde immédia_tement A dans l'ordre des brins incidents de S; la zone du brin gauche de A_2 : Z_2, a déjà été déterminée, vu l'ordre de traitement des brins; la liste L_2 des faces actives de Z_2 est donc connue. L'ensemble des faces actives de Z est alors : $L = L_2$ - [faces de la liste fb du brin droit de A_2] + [faces de la liste fb du brin gauche de A]. La procédure créer_zones se formule ainsi :

```
procedure créer_zones =
  { créer la zone zone_mère; /* la seule zone sans bord interne */

    pour chaque sommet S, dans l'ordre lexicographique, faire
      pour chaque arête A de premier sommet S faire
       si le brin droit de A n'a pas encore de zone
       alors
        { si (A est la première arête incidente en S)
             et (S est un point de naissance)
          alors { sous ← arête située juste sous S dans le plan de l'image;
                  si sous n'existe pas alors Z ← zone_mère
                  sinon Z ← zone du brin (sous, droit)
                }
          sinon { créer une nouvelle zone : Z ;
                  A2 ← arête précédant A en S;
                  Z2 ← zone du brin (A2, droit);
                  [faces actives de Z] ← [faces actives de Z2]
                                       - [faces de la liste fb de (A2, droit)]
                                       + [faces de la liste fb de (A2, gauche)]
                }
          pointeur "zone" des brins du bord de (A, droit) ← Z;
          ajouter A en tête de la liste aret_zone de Z;
        }
  }
```

3.4 ORDRE DES FACES ACTIVES DANS UNE ZONE

Cette phase traite chaque zone, successivement, et dans un ordre quelconque. Elle détermine l'ordre des faces actives de la zone traitée en chacun de ses brins. Si nécessaire, des arêtes d'intersection entre faces sont ajoutées; en ce cas, la zone ne correspondra plus totalement à sa définition première, puisqu'elle pourra être faite de deux bords internes accolés, par exemple. Par contre, la principale propriété d'une zone reste vraie: la liste des faces actives est la même pour tous les brins, et cela seul importe pour l'instant. Une zone est donc ici seulement une liste de brins ayant le même ensemble de faces actives.

L'algorithme commence par ramasser dans une liste L tous les brins de la zone, Z; les brins créés lors de cette phase, en cas d'intersection entre faces, seront ajoutés à la fin de L.

Les listes fb des brins sont utilisées pour représenter l'ordre des faces actives le long des arêtes; elles sont initialisées comme vides; puis chaque face active de la zone est insérée tour à tour dans la liste fb de tous les brins de la zone, connus par la liste L.

L'ajout à un brin d'une face active f n'ayant aucune intersection avec des faces déjà présentes dans la liste fb du brin est simple: il suffit d'insérer f de telle façon que la liste reste ordonnée.

Que faire si f coupe une face g déjà présente dans un brin de l'arête a ? Soit i le point a $\cap$ (f $\cap$ g). Dans le cas général, i est différent des sommets de a; dans les structures de données, a est alors coupée en a, a_2 par le sommet i; le (ou les, si a est une arête d'intersection entre deux faces) brin créé d'arête a_2, et de zone Z, est ajouté à la fin de L. Il faut, de plus, ajouter une arête qui représente l'intersection entre f et g; mais seul un de ses brins est connu, celui de sommet i : l'autre brin est inconnu pour l'instant; pourtant le point d'intersection i' complémentaire existe, et sera ou a déjà été trouvé. Provisoirement, est créée une arête dite "moitié": le brin de sommet i est correctement affecté, et l'autre brin prend une valeur quelconque; les listes fb des arêtes moitiés ne sont pas prises en compte pour l'instant.

Quand la face f a été ajoutée à la liste fb de tous les brins de L, nous sommes sûrs que pour toute arête moitié existe une arête moitié complémentaire. Pour chaque face g intersectant la dernière face f ajoutée, les arêtes moitiés de f$\cap$g sont ordonnées sur leur sommet connu; on obtient une suite d'arêtes moitiés u_1, u_2, ...u_n dont le brin d'indice zéro est correct, et une suite d'arêtes moitiés complémentaires v_1, v_2, ..v_n, dont le brin d'indice un est correct; il suffit alors de fusionner les moitiés complémentaires u_i et v_i en une arête uv_i . Les deux brins de celle-ci sont ajoutés à la fin de L.

Enfin, quand toutes les arêtes des intersections de f et des autres faces ont été obtenues par fusion, leurs listes fb sont déterminées: ce sont de simples copies physiques des listes fb de brin de même bord. L'algorithme se formule ainsi :

```
procédure traiter_zone ( Z: zone) =
{ L ← liste de tous les brins de Z;
  initialiser à vide l'ordre des faces pour chaque brin de L;

  pour chaque face active f de Z faire
  {   pour chaque brin (a, c) de L faire
      { si (f n'intersecte aucune des faces déjà insérées)
        alors insérer f à sa place dans la liste fb
        sinon
          /* supposons que f coupe une face g en i= a ∩ (f ∩ g) */
         si i est différent des extrêmités de a
         alors { créer un sommet pour i;
                 couper a par i en a, a2;
                 ajouter en fin de liste L le brin (a2, c);
                 si (a2, 3-c) est aussi un brin de zone Z
                    /* a était une arête d'intersection entre faces*/
                 alors ajouter en fin de liste L le brin (a2, 3-c);
                 en i ajouter une arête moitié correspondant à f ∩ g;
                 continuer de traiter le brin (a,c);
                }
         sinon /* i est un des sommets de a */
                { si ( l'arête moitié de f ∩ g n'est pas encore créée
                       et est bien dans la zone )
                  alors ajouter en i l'arête moitié pour f ∩ g;
                  continuer de traiter le brin (a,c);
                }
      } /* fin de pour chaque brin */

      /* ajouter les arêtes d'intersection entre f et les autres faces: */
      pour chaque face g intersectant f sur Z, faire
      { ordonner sur leur sommet connu les arêtes moitiés;
        fusionner les moitiés gauches et droites complémentaires;
        ajouter les deux brins de ces arêtes en fin de liste L;
      }
      marquer l'ordre des faces actives sur les brins des arêtes d'intersection;
      /* il est égal à celui d'un brin de même bord */

  } /* fin de pour chaque face */
} /* fin  procédure */
```

Cet algorithme suppose qu'il ne peut exister dans la zone traitée de point d'intersection inconnu (fig. 2) entre deux arêtes d'intersection de la face f avec deux autres faces déjà insérées dans la zone, les faces g, puis h. Justifions cette supposition :

Soit a l'arête d'intersection de f avec la face g, et soit b l'arête d'intersection de f avec la face h. Soit i un point hypothètique, intersection de a et b, qui se trouve strictement à l'intérieur de la zone. En ce cas i appartient à la droite d'intersection de g et de h. Or la ou les arêtes d'intersection de g et h ont été détectées par l'algorithme, lors de l'ajout de h : i ne peut donc exister. Par contre a et b peuvent éventuellement avoir un sommet en commun, mais ce n'est pas là un point

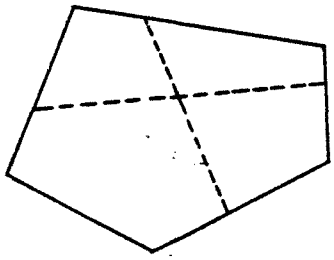


Fig. 2 : Cette situation est impossible

d'intersection inconnu.

Cet algorithme suppose aussi qu'il ne peut exister de bord uniquement formé avec des arêtes d'intersection de la face f avec d'autres faces (fig. 3); ce qui est bien vérifié :

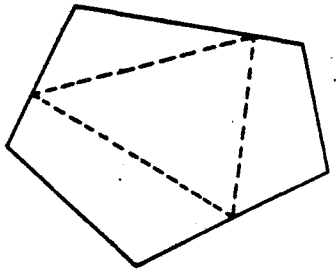


Fig. 3 : Autre situation impossible

Supposons qu'il existe un bord B uniquement constitué d'arêtes créées par intersection de plans. Soient a, b et c, trois arêtes de ce bord (il y en a au moins trois). On peut supposer que : $a = f \cap g_1$, $b = f \cap g_2$, $c = f \cap g_3$.

Soit i un point quelconque situé dans ce contour. Les plans des faces actives f, g_1..., g_n sont ordonnés suivant leur profondeur en ce point; on peut supposer que l'on a, par exemple, $g_1 < f < g_2 < g_3$...

Or, quand on franchit l'arête a, on doit intervertir l'ordre entre f et g_1; il en est de même quand on franchit b pour f et g_2 et quand on franchit c pour f et g_3. Ceci signifie que g_1, g_2 et g_3 doivent être tous les trois prédecesseurs ou successeurs immédiats de f dans l'ordre des plans en i, ce qui est évidemment impossible.

3.5 SUPPRESSION DES FACES INUTILES

Cette étape considère chaque bord de l'état actuel de l'image: les bords sont parcourus par la même méthode qu'au paragraphe 3.3. L'ordre des faces actives en un bord est connu, puisque c'est l'ordre en un quelconque de ses brins. Cette étape va purger cet ordre des références inutiles à des faces. L'orientation (avant ou

arrière) de chaque face est connue, de même que la primitive géométrique (feuille de l'arbre de construction) limitée par la face. L'algorithme parcourt l'ordre des faces actives, et entretient au fur et à mesure un booléen "appartient" pour chaque feuille de l'arbre de construction, qui mémorise si l'intervalle courant appartient ou non aux diverses feuilles de l'arbre:

```
procédure marquer_intervalle_dedans_ou_dehors =
{ L ← liste des faces actives;
  tant que L non vide faire:
    { f ← première face de L;
      p ← primitive de l'arbre de construction limitée par f;
      l'intervalle appartient à p ← f est une face avant;
      l'intervalle appartient à la scène ← app_csg (f, la_scène);
      marquer ce résultat sur l'intervalle /* c.à.d. sur f */;
      L ← suite de L;
    }
}


function app_csg ( f, arbre_csg ): boolean;
{ si (arbre_csg est une primitive de l'arbre)
       alors app_csg ← l'intervalle [f..f'] appartient à arbre_csg
       /* f' est la face suivant f dans l'ordre des faces actives */
  sinon si (arbre_csg= a ∩ b)
       alors app_csg ← app_csg (f, a) et app_csg (f, b)
  sinon si (arbre_csg= a ∪ b)
       alors app_csg ← app_csg (f, a) ou app_csg (f, b)
  sinon /* arbre_csg = a - b */
       app_csg ← app_csg (f, a) et not app_csg (f, b)
}
```

Dans la liste des faces actives, les références inutiles à des faces sont ensuite supprimées : une face est inutile quand les deux intervalles qu'elle sépare appartiennent tous deux à l'objet défini par la racine de l'arbre de construction, ou quand aucun de ces deux intervalles n'appartient à cet objet.

Quand cette purge a été effectuée pour chaque bord, les arêtes inutiles peuvent être supprimées : une arête est inutile quand ses deux brins ont des ordres de faces actives semblables. De même, certains sommets sont éliminés : d'une part ceux qui n'ont plus aucune arête incidente, d'autre part ceux qui appartiennent à deux arêtes colinéaires, qui sont fusionnées. Enfin, un ultime balayage inverse le vecteur normal de certaines faces, quand elles limitent des objets soustraits à la scène.

4 ALGORITHME DE VISUALISATION

Après réduction, la structure de données ne contient plus de faces inexistantes dans la scène; mais elle contient encore des faces cachées.

Déterminer la liste LV des faces vues pour un brin, à partir de la liste ordonnée et purgée LA des faces actives, est trivial. Si LA= (f_1, f_2, ... f_o, ... f_n), et si f_o est la première face opaque de LA, alors LV= (f_1, f_2, ... f_o). Si LA ne contient aucune face opaque alors LV=LA. Une arête est vue si et seulement si ses deux brins ont des listes de faces vues différentes; dans le cas contraire elle est cachée et est donc supprimée. De même les sommets devenus isolés, ou entre deux arêtes colinéaires, sont éliminés.

Les arêtes de l'image finale sont donc connues, de même que les faces vues à gauche et à droite de chaque arête. Cette structure permet donc des sorties graphiques au trait, comme sur les images de la figure 1 (les primitives géométriques utilisées sont le cube, la sphère et le cylindre); pour épurer le dessin, certaines arêtes peuvent ne pas être tracées : on peut, par exemple, ne pas représenter les arêtes séparant deux faces avant d'une sphère discrétisée en marquant ces arêtes lors de la facettisation. La connaissance des contours permet aussi la réalisation d'images antialiassées sur des mémoires de trame, par exemple avec l'algorithme de balayage avec filtrage par suréchantillonnage local de I.5.2.2.

5 COMPLEXITE THEORIQUE

Notons S le nombre de sommets de la figure initale et A le nombre d'arêtes. D'après la procédure de facettisation, on a A = O(S).

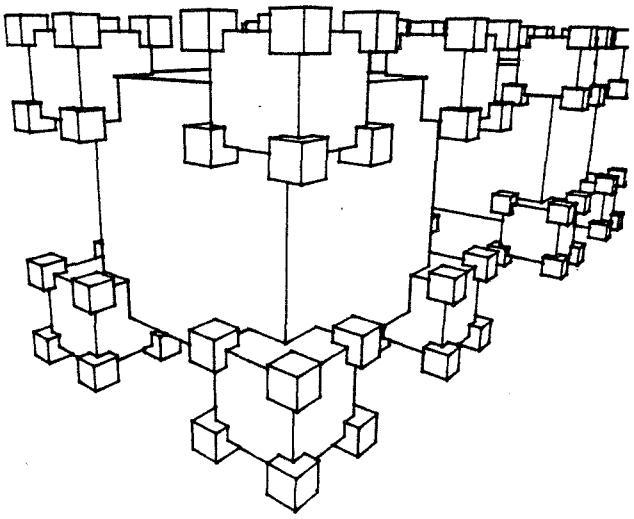
L'étape d'initialisation est en O(A).

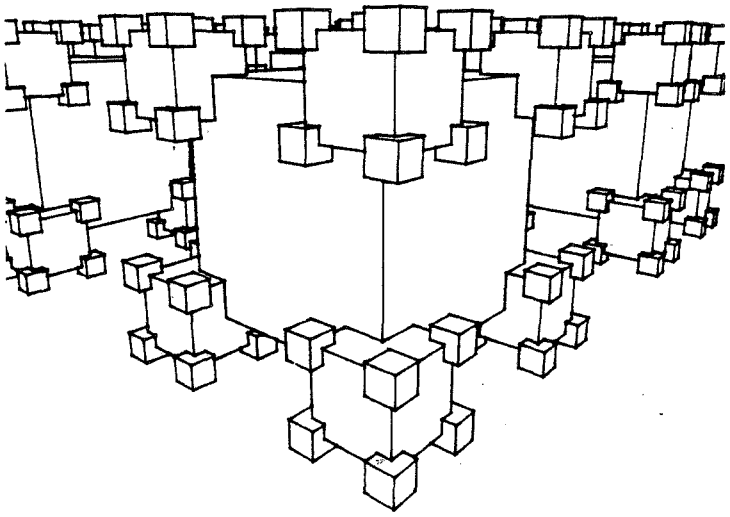
L'algorithme de BENTLEY-OTTMAN est en O((A+K)logA), où K est le nombre des points d'intersection entre les arêtes; notons que, dans le pire des cas, $K=O(A^2)$.

La procédure de création des zones se fait en O((A+K)P), où P est le nombre maximum de faces actives dans une zone; dans le pire des cas, P est en O(A).

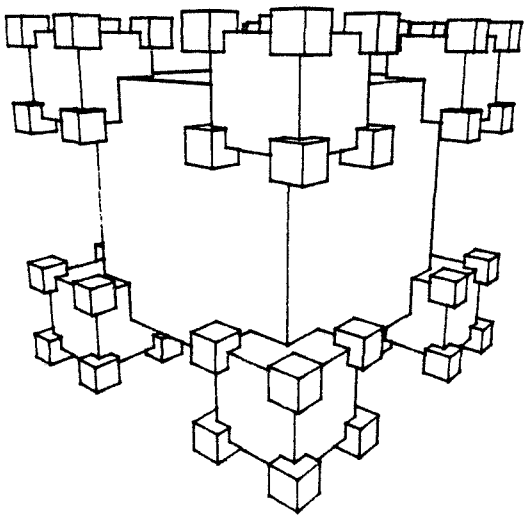
Notons J le nombre de points d'intersection entre les arêtes et les droites d'intersection entre faces. La détermination de l'ordre des faces actives sur les brins est en $O(P^2(A+J+K)+JlogJ)$. On pourrait remplacer le terme P^2 par : PlogP, si l'ordre des faces actives était représenté par un arbre binaire plutôt que par une liste.

L'élimination des faces inutiles est en O(P(A+J+K)), de même que l'algorithme de visualisation.





Sur ces quatre scènes fractales, j'ai constaté empiriquement que le nombre de points d'intersection entre les arêtes d'une scène, après projection sur le plan de l'image, peut être multiplié par approximativement quatre quand le nombre d'arêtes initiales est multiplié par 2. Le nombre de points d'intersection entre toutes les arêtes projetées de la scène, vues ou cachées, peut donc croître comme le carré du nombre d'arêtes initiales, dans certains cas. Pout éviter cet effet en N^2, l'algorithme élimine d'abord les parties cachées sur des sous-scènes, puis fusionne les résultats partiels. Ainsi, sur ces scènes, la méthode met à peu près deux fois plus de temps pour traiter les données que pour les lire. Il est vrai que cette situation est un cas favorable : l'arbre CSG ne contient que des unions.



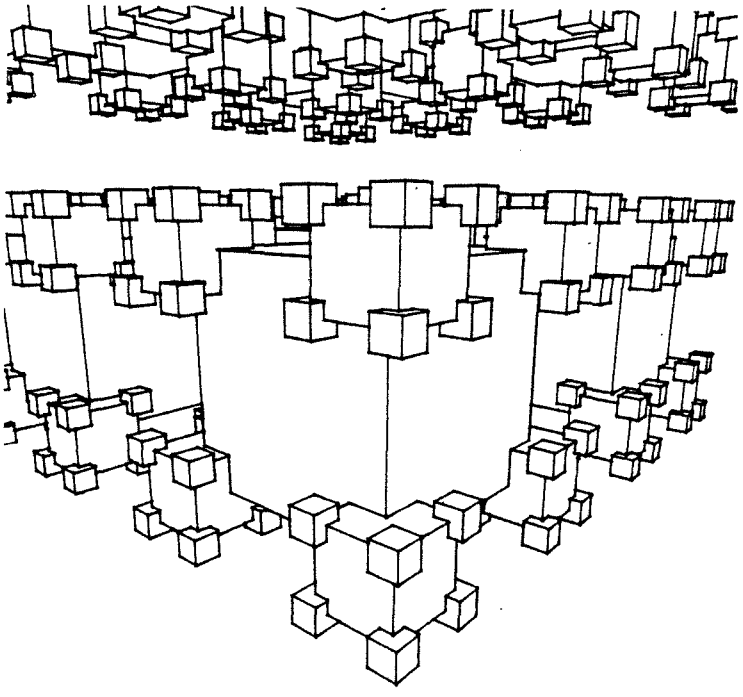


Fig. 4 : Quatre scènes fractales

6 OPTIMISATIONS POSSIBLES

6.1 REDUCTION DE SOUS-ARBRES DE CONSTRUCTION

Dans l'implantation actuelle, la structure choisie pour représenter l'ordre en profondeur est une simple liste. Ce choix n'est théoriquement pas optimal : si P est le nombre de faces actives, l'insertion se fait en O(P) dans une liste alors qu'elle pourrait se faire en O(log P) dans un arbre.

Ce constat est exact, mais partiel. Il néglige notamment une conséquence importante de la complexité en profondeur (selon l'axe Oz) sur le nombre des points d'intersection entre les arêtes dans le plan de l'image. Dans le pire des cas, ce nombre peut être de l'ordre du carré du nombre d'arêtes initiales. Or l'expérience (fig. 4) montre que ce cas est en fait fort probable (nous reviendrons ultérieurement sur ce point).

Il existe heureusement un recours simple : l'algorithme de réduction d'un arbre peut être appliqué d'abord aux sous-arbres, de manière récursive. Quelques menus aménagements sont nécessaires: par exemple un sous-arbre réduit doit être considéré ultérieurement comme une feuille, notamment par les faces qui le limitent et qui pointeront dorénavant sur lui. Quand la complexité en profondeur (le nombre de faces actives dans une zone) est limitée, il n'est en pratique plus tellement avantageux de représenter l'ordre en profondeur par un arbre.

Après cette réduction itérative, l'algorithme de visualisation est toujours appliqué en phase finale : c'est la raison pour laquelle j'ai distingué entre l'algorithme de réduction des arbres de construction, et l'algorithme de visualisation de l'arbre réduit.

6.2 ALGORITHME D'ELIMINATION DES PARTIES CACHEES

Une autre optimisation est possible lorsque, après éventuelle réduction des sous-arbres, les seuls opérateurs restants sont de simples unions. Très fréquemment, aux plus hauts niveaux de l'arbre de construction, les scènes sont de simples unions d'objets, et les autres opérations booléennes ne sont utilisées qu'aux plus bas niveaux de l'arbre pour définir des objets assez simples. Le problème est alors réduit à la détermination des parties visibles d'un ensemble de faces éventuellement intersectantes. En ce cas, les faces arrières n'appartenant pas à des objets transparents peuvent être éliminées d'office. D'autre part, lors de la détermination de l'ordre des faces actives sur chaque brin, les faces qui se trouvent derrière une face opaque n'ont pas à être insérées dans les listes fb: il n'est plus utile de gérer l'ordre des faces actives éventuellement cachées, mais seulement l'ordre des faces vues. L'algorithme obtenu est alors une méthode particulièrement efficace d'élimination des parties cachées : si la complexité en profondeur est limitée, il est aussi efficace

que l'algorithme de BENTLEY-OTTMAN. Cependant, là aussi, la complexité en profondeur (selon l'axe Oz) a le même effet que pour l'algorithme de réduction. Le même recours est possible: l'ensemble des faces est divisé en sous-ensembles de taille comparable, et l'algorithme d'élimination des parties cachées est appliqué sur chacun d'eux; puis les diverses perspectives partielles sont fusionnées, par le même algorithme. Ainsi la complexité en profondeur ne dépasse jamais certaines bornes.

7 DIFFICULTES D'IMPLANTATION

L'implantation de cet algorithme se heurte à des problèmes d'imprécision numérique :

Imprécision 2D : les méfaits de l'imprécision numérique sur l'algorithme de BENTLEY-OTTMAN ont été exposés en I.4 et résolus en I.4.5.

Imprécision 3D : dans R^3 , trois plans s'intersectant deux à deux ont un point d'intersection unique. Avec l'emploi de la représentation flottante, les trois droites d'intersection des plans pris deux à deux ne se couperont généralement pas en un point unique, mais formeront en projection sur le plan de l'image un triangle (on aura trois droites gauches deux à deux dans l'espace); l'ordre en profondeur des trois plans n'est pas cohérent dans ce triangle... En clair, aucun des raisonnements géométriques que l'on peut faire dans R^3 n'est applicable avec la représentation flottante, car l'imprécision fait perdre jusqu'à la transitivité de l'égalité.

Les paragraphes suivants proposent quelques solutions pratiques.

7.1 PREMIERE SOLUTION PROPOSEE

Un premier remède (celui implanté) consiste, là aussi, à caler les équations des faces : ax+by+cz+d=0. Les nombres a,b,c sont calés dans $[-T..+T]^3$, et les nombres d dans [-L..+L]. Cette quantification des positions des faces assure : d'une part, que leurs droites d'intersection auront des équations homogènes avec celles des arêtes initiales, d'autre part que les trois droites d'intersection de trois plans s'intersectant deux à deux se coupent bien en un point. Nous pourrons donc effectuer les raisonnements géométriques usuels dans R^3 .

La représentation par des nombres flottants implique un autre défaut : la droite d'intersection de deux faces contiguës en une arête n'est pas exactement confondue avec la droite support de l'arête (fig. 5 et 6). Ce défaut n'est pas corrigé par le simple calage décrit ci-dessus; il est même accentué, en règle générale.

Ce défaut n'a guère de conséquence pour des faces contiguës situées de part et d'autre d'une arête (fig. 5). Simplement, si une face F vient à couper deux faces contiguës f_1 et f_2 séparées par l'arête a, les droites d'intersection de f_1 et F, et de f_2

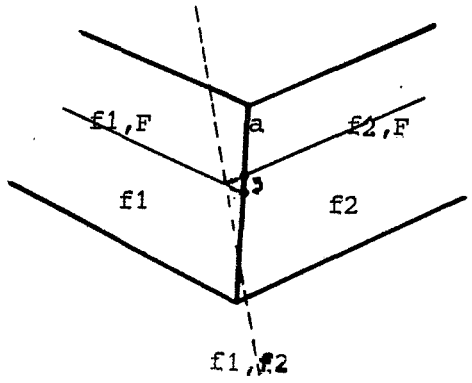


Fig 5 : Le défaut (très exagéré) sur deux faces contiguës

et F, ne se couperont pas tout à fait sur a; ce défaut est visuellement imperceptible, quand on choisit des valeurs de T et L suffisamment grandes. Ce défaut n'implique aucune incohérence, car le point d'intersection de ces deux droites est bien situé sur la droite d'intersection de f_1 et f_2.

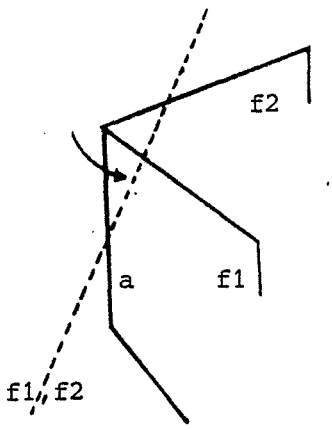


Fig 6 : Le défaut (très exagéré) sur deux faces superposées

Par contre, ce défaut a une conséquence fâcheuse dans le cas de deux faces d'orientations différentes qui se superposent le long d'une arête (fig. 6). Soit a une arête faisant partie du contour d'un cube vu en perspective; a limite donc une face avant : f_1, visible, et une face arriere : f_2, cachée. La droite d'intersection de f_1 et de f_2 est légèrement différente de a. Dans le cas où cette droite passe sur f_1 et f_2, dans le plan de l'image, la zone entre a et la droite d'intersection a un ordre des faces incohérent : f_2 passe devant f_1, c'est à dire qu'un rayon visuel sort du cube avant d'y être entré. Cette incohérence perturbe évidemment le bon fonctionnement de

l'algorithme de réduction ! D'où la nécessité d'un conditionnement préalable des primitives géométriques : chacune d'elles subit l'algorithme de réduction de sous-arbre de construction présenté; cette passe permet de détecter les zones incohérentes. Dans le cas de primitives géométriques convexes, il suffit de supprimer dans ces zones les deux faces présentes; puis les arêtes devenues inutiles sont elles aussi supprimées.

Comment caler au mieux un vecteur normal flottant sur un vecteur entier relatif, dont les composantes sont bornées ? Le problème 2D, plus simple, est d'abord étudié.

Soit (a b) un vecteur normal où a et b sont flottants; sans perte de généralité, on supposera $a \geq b \geq 0$; on cherche (A B), deux entiers, tels que A/B soit le plus proche possible de a/b, avec $A \geq B \geq 0$.

Les fractions continues apportent une réponse simple et efficace. On rappelle que le développement en fractions continues f_0, f_1 ..., d'un réel X s'obtient par : $f_0 = \text{tronc}(X)$, $X_n = 1/(X-f_n)$, puis $f_{n+1} = f_n + 1/\text{tronc}(X_n)$; si $X \in Q$, ce développement cesse en $X = f_n$, sinon il est infini et converge vers $X \leftarrow f_n$.

a/b est développée en fractions continues : une suite a_1/b_1, a_2/b_2, ... est obtenue; elle converge vers a/b; la théorie élémentaire des fractions continues démontre que le dernier terme de cette suite, a_n/b_n, tel que $a_n \leq T$, est la meilleure approximation A/B possible de a/b. La meilleure approximation, avec A le plus petit possible, ne peut être que a_{n+1}/b_{n+1} ([KNUT 81], "Analysis of EUCLID's Algorithm").

Exemple : (a b) = (4374.21 178.95). Supposons $T=10^3$. Le développement en fractions continues de a/b est la suite 24/1, 49/2, 171/7, 220/9, 3911/160, ... Donc (A B) = (220 9). a/b= 24.4437..., et A/B= 24.444... L'approximation (A' B') meilleure que (220 9), avec A' et B' le plus petit possible, est (3911 160).

Quel est le nombre de directions (A B) possibles, ou de rationnels A/B avec $T \geq A \geq B > 0$? En supposant une répartition à peu prés homogène de ces rationnels B/A dans]0,1], ce nombre peut mesurer la qualité du calage. Soit Q(T) ce nombre :

Q(T) = cardinal { (A B) tel que $T \geq A \geq B > 0$ et PGCD(A,B)=1 }
= Q(T-1) + cardinal { (A B) tel que $T \geq A \geq B > 0$ et A=T et PGCD(A,B)=1 }
= Q(T-1) + cardinal { (T B) tel que $T \geq B > 0$ et PGCD(A,T)=1 }
= Q(T-1) + ϕ(T)
= Q(T-2) + ϕ(T-1) + ϕ(T)
= ϕ(1) + ϕ(2) + ... + ϕ(T-1) + ϕ(T)

On rappelle que $\phi(x)$ est l'indicateur d'EULER de x; $\phi(x)$ est le nombre d'entiers y inférieurs ou égaux à x, tels que PGCD(x,y)=1; $\phi(x)=x-1$ signifie que x est premier; $\phi(x)$ est toujours strictement inférieur à x, sauf pour $x=1=\phi(x)$.

Un théorème de théorie des nombres vient à point affirmer que $Q(T) = \phi(1) + \phi(2) + ... + \phi(T-1) + \phi(T)$ est approximativement égal à $3T^2/\pi^2$ [HARD 79]. Hélas, le théorème ne borne pas la qualité de l'approximation. Pour T=100, Q(T) vaut 3043 et l'entier le plus proche de $3T^2/\pi^2$ vaut 3040. Pour T=1000, Q(T) vaut 304191 et l'entier le plus proche de $3T^2/\pi^2$ vaut 303963. Pour T=10000, Q(T) vaut 30.397.485 et l'entier le plus proche de $3T^2/\pi^2$ vaut 30.396.355.

Un autre algorithme, plus brutal, est envisageable. La direction (a b) est celle du segment [(0 0) (a b)] dans le plan; traçons le segment sur une mémoire d'image de TxT pixels, par une variante de l'algorithme de BRESENHAM; les coordonnées du pixel "le plus proche" du segment donnent le vecteur (A B) cherché. Supposons que $a > b > 0$: le segment a une pente inférieure à 45 degrés. La méthode de BRESENHAM afficherait alors un pixel par colonne; mais il nous suffit d'afficher un pixel par ligne. Le pseudocode suivant décrit l'algorithme :

```
caler(a b) =  /* assumer que a > b */
{ B←0; A← arrondi(a/b*B); B'←0;
  répéter {  B'←B'+1; A'← arrondi(a/b*B');
             si A'≤T et si (A' B') est meilleur que (A B)
             alors (A B)← (A' B');
          } jusqu'à A' > T;
  rendre( (A B) );
}
```

Une direction (A' B') est meilleure qu'une direction (A B) quand l'angle entre les vecteurs (A' B') et (a b) est inférieur à l'angle entre (A B) et (a b).

Considérons le problème 3D; la première méthode, utilisant les fractions continues, ne peut hélas pas être réutilisée : on sait que les fractions continues se généralisent mal aux dimensions supérieures; par contre, le deuxième algorithme se généralise d'une façon naturelle, en traçant un segment dans une matrice de voxels.

Pour toutes les images présentées, T=370; la version utilisée du logiciel recale fort mal les directions, puisque $(a \geq b \geq c)$ est systématiquement recalé sur : (T arrondi(bT/a) arrondi(cT/a)) : seules $\Theta(T)$ directions sont donc effectivement employées. D'où quelques défauts visibles sur les images pour les faces presque de profil.

7.2 DEUXIEME SOLUTION PROPOSEE

Le premier calage proposé n'est pas totalement satisfaisant; aussi peut-on envisager, après le calage des positions des faces, de recalculer sommets et arêtes, de façon à ce qu'ils se trouvent exactement sur les faces calées. Ainsi, sachant que le sommet S appartient aux trois faces f_1, f_2 et f_3, il est facile de calculer des coordonnées rationelles telles que S appartienne bien, numériquement parlant, à ses trois faces. Malheureusement, cela n'est pas aussi évident pour les sommets de degré supérieur à trois... De plus, dans cette solution, les tailles des entiers (les termes r et d dans la représentation rationnelle) manipulés augmentent très vite : l'arithmétique entière sur 64 bits qui suffisait pour la solution précédente devient ici insuffisante.

7.3 TROISIEME SOLUTION PROPOSEE

La troisième solution génère une géométrie parfaitement cohérente; pour cette raison, elle est réutilisée au chapitre suivant, qui traite des opérations booléennes sur les BREP de polyèdres ("*Boundary-Representation*" ou représentation par les frontières). Malheureusement l'emploi d'une véritable arithmétique rationnelle devient inévitable.

Cette fois, ce sont les sommets qui sont recalés, sur des entiers ou sur des rationnels (les flottants sont des représentations approximatives des rationnels). Puis les paramètres (a b c d) des équations des faces sont recalculés à partir des coordonnées des sommets. Généralement, trois sommets sont nécessaires et suffisants pour définir le plan d'une face : un quatrième sommet se trouve le plus souvent légèrement à coté du plan défini par les trois autres. Cette méthode n'est donc applicable que pour des faces triangulaires; d'où l'idée de trianguler d'abord toutes les faces. Les primitives employées étant des cubes, des pyramides, ou des facettisations de quadriques, cette triangulation est toujours possible et simple, bien qu'un peu ennuyeuse; de plus elle ne modifie pas l'ordre de grandeur de la taille des données; de ce point de vue, elle est donc sans incidence sur les performances théoriques de l'algorithme.

Pour donner un ordre de grandeur des entiers manipulés, on supposera que les sommets sont recalés sur une grande grille entière tridimensionnelle $[0..G]^3$; les paramètres des plans des faces ne peuvent ainsi excéder certaines bornes; la normale au plan ABC est le produit des vecteurs AB et AC; donc $|a|, |b|, |c| \leq 2G^2$; $d = -A\ (a\ b\ c)^t$, d'où $|d| \leq 6G^3$. Les coordonnées des points d'intersection entre trois plans sont les nombres rationnels les plus encombrants qui doivent être stockés. Ces points d'intersection sont les solutions rationnelles d'un système linéaire à coefficients entiers relatifs : $a_1x+b_1y+c_1z+d_1 = a_2x+b_2y+c_2z+d_2 = a_3x+b_3y+c_3z+d_3 = 0$; d'après les bornes sur $|a_i|$, $|b_i|$, $|c_i|$ et $|d_i|$, le déterminant Δ ne peut jamais être supérieur, en valeur absolue, à $48G^6$; Δx, Δy et Δz ne peuvent jamais être supérieur, en

valeur absolue, à $144G^7$. Si les coordonnées sont représentées sous forme homogène (Δx Δy Δz Δ), les entiers les plus grands qu'il est nécessaire de stocker ne dépassent pas $144G^7$; si les coordonnées sont représentées sous forme euclidienne ($xe+xr/\Delta$ $ye+yr/\Delta$ $ze+zr/\Delta$), les entiers les plus grands qu'il faut stocker ne dépassent pas $48G^6$.

Ainsi, même pour une valeur de $G=10^4$ certainement insuffisante, des entiers comme $48G^6 = 48\ 10^{24}$ voire $144G^7 = 144\ 10^{28}$ peuvent être atteints. Pour obtenir une précision du millimètre dans un univers d'un kilomètre cube, typiquement pour des applications architecturales et urbanistiques, il faut une valeur de $G=10^6$, et les entiers à stocker peuvent atteindre $48G^6 = 48\ 10^{36}$ ($\sim 2^{125.174...}$), voire $144G^7 = 144\ 10^{42}$ ($\sim 2^{146.690...}$)... Une arithmétique doit les représenter fidèlement en machine; son coût n'est pas négligeable.

On peut estimer préférable de ne pas limiter a priori la taille de la grille entière : alors les entiers manipulés peuvent être arbitrairement grands, et il n'est plus possible d'admettre qu'une opération (+ - * / < =) s'effectue en un temps constant. Les performances théoriques sont dégradées.

8 CONCLUSION

Deux algorithmes ont été présentés : une méthode de réduction, et une méthode d'élimination des parties cachées. La seconde méthode est très efficace, d'après les tests empiriques : le temps de calcul croît à peu prés linéairement avec le nombre d'arêtes, pour les données testées (les fractales). On peut seulement regretter une constante implicite assez forte, dû aux précautions arithmétiques.

La méthode de réduction construit implicitement une BREP de la scène, en privilégiant un plan de projection. Cette technique a des avantages et des inconvénients :

Avantages : le problème des opérations booléennes est très simplifié, grâce à la construction d'un ordre total sur les faces actives dans une zone (la profondeur); les cas particuliers sont relativement peu nombreux; surtout, les imprécisions numériques peuvent être résolues efficacement (7.1). Enfin les performances sur toutes les images présentées sont correctes : le programme met entre deux (quand l'arbre CSG ne porte que des unions) à quatre fois plus de temps pour traiter les données que pour les lire.

Inconvénients : je ne connais pas de travaux théoriques sur le nombre K moyen de points d'intersection entre les projections sur un plan des N arêtes d'une scène, visibles ou non. Il n'est par contre pas difficile de faire quelques mesures empiriques, ou de regarder quelques exemples. Considérez par exemple la figure des cubes fractals (fig. 4), et pensez à toutes les arêtes visibles ou cachées, et à

leurs points d'intersection.... K augmente très vite avec la complexité en profondeur; il n'est pas rare que K grandisse comme N^2; c'est le cas des scènes fractales, d'après un constat empirique.

Cela nuit évidemment aux algorithmes d'élimination des parties cachées qui utilisent le balayage de BENTLEY-OTTMAN, et entendent procéder en une passe unique : s'ils sont en O((N+K)logN), et si K est en N^2... Les articles d'algorithmique négligent ce phénomène; c'est certes légitime pour les applications typiquement 2D : dessins de circuits électroniques, plans, croquis et graphismes divers, où les valeurs de K restent relativement faibles; cela le parait beaucoup moins pour les applications 3D. Cependant, ces algorithmes peuvent toujours travailler itérativement sur des sous-scènes et éliminer au fur et à mesure les faces cachées : la complexité en profondeur reste donc limitée (sauf dans certains cas d'école, par exemple quand toutes les faces sont transparentes). Ainsi les temps de calcul pour les scènes fractales sont-ils linéaires, et non quadratiques.

Ce recours est par contre impossible pour l'algorithme de réduction, avec certains arbres CSG. Soit S l'arbre CSG représentant, par exemple, une scène de huit cubes fractals, et C une primitive quelconque, par exemple un cube; pour réduire l'arbre $S \cap C$, ou l'arbre S-C, les faces inexistantes peuvent être éliminées, mais pas les faces cachées. D'où l'existence de $O(N+K) = O(N^2)$ zones avec de tels arbres CSG; l'algorithme de réduction exige alors un espace mémoire et un temps de calcul qui ne peut être meilleur; de fait, sur un tel arbre CSG, le programme avorte après avoir saturé l'espace mémoire disponible; l'algorithme d'ATHERTON, lui, est très ralenti, mais fonctionne correctement. A priori, ce défaut semble inhérent au fait de raisonner sur un plan de projection. Il convient cependant de relativiser cette conclusion quelque peu pessimiste sur l'algorithme de réduction; je pense que de tels arbres constituent des cas extrêmes (cf 6.2).

Le chapitre suivant surmonte cet "effet du N^2"; un algorithme déterminant les frontières de l'union, de l'intersection, ou de la soustraction de deux polyèdres y est présenté : il ne travaille pas en projection.

9 ANNEXE : NOTE SUR LA MODELISATION DES SCENES

Cette note indique rapidement comment sont modélisées et générées les scènes des figures : cendrier, cubes fractales, escalier hélicoïdal, etc. La description est indépendante de l'utilisation; plusieurs logiciels de synthèse d'images acceptent en entrée cette description et visualisent la scène ainsi décrite : un lancer de rayons dû à ARGENCE [ARGE 88], un facettiseur dû à BEIGBEDER [BEIG 87], et un tampon en z antialiassé dû à BEIGBEDER [BEIG 87] et GHAZANFARPOUR-KHOLENDJANY [GHAZ 85], utilisable quand seules sont employées des unions d'objets. En effet, les scènes sont modélisées par des arbres

CSG. En machine, l'arbre CSG est une expression symbolique LISP, une S-exp :

9.1 DESCRIPTION LISP DE L'ARBRE CSG

Les primitives : la S-exp (cube r v b c) décrit le cube unité de couleur rvb dans le repère RVB, c étant un complément d'information sans intérêt ici. De même, (sphere r v b c) décrit la sphère unité. (cylindre r v b c) décrit le cylindre d'axe Oz, entre z=0 et z=1, de rayon 1. (cône r v b c) décrit le cône de sommet (0 0 1), de base le cercle centré à l'origine et de rayon 1. (prisme r v b c (x1 y1 x2 y2 x3 y3 ...)) décrit le prisme de sommet (0 0 1), de base dans Oxy (x1 y1) (x2 y2)...

Les opérateurs booléens : la S-exp (union A B C ...) décrit l'union de A, B, C....; A, B, C... sont des S-exp décrivant les objets A, B, C.... De même la S-exp (inter A B C ...) décrit l'intersection des objets A, B, C... Enfin (moins A B C D ...) décrit la différence de A et de l'union de B, C, D...

Les transformations affines : la S-exp (trans dx dy dz A) décrit l'objet obtenu en translatant de (dx dy dz) l'objet A. De même (rotat rx ry rz theta A) décrit l'objet obtenu après rotation de l'objet A, d'un angle theta, autour de l'axe (rx ry rz). Enfin (affin kx ky kz A) décrit l'objet obtenu après affinité sur l'objet A. Ces opérations suffisent théoriquement pour engendrer toutes les autres, comme la symétrie par rapport à un plan quelconque.

```
(union   (trans 0 0 300 (moins (rotat 0 0 1 45 (cube 100 125 43 0))
                               (trans 0 0 -0.5 (cylindre 100 125 43 0))))
         (affin 1 1 4 (cône 100 220 55 0)))
```

est un exemple d'arbre CSG modélisant une scène simple. Cette description est parfois bien fastidieuse; mais LISP apporte toute sa puissance d'expression et de manipulation symbolique (fonctions, macros, récursivité, etc) :

9.2 MANIPULATIONS LISP

Transformations affines : il n'est pas très agréable de composer manuellement les transformations affines pour exprimer, par exemple, une affinité de centre quelconque; LISP permet de définir une fois pour toutes cette affinité de centre (x y z) :

```
(de affin-centre-qcq (x y z kx ky kz A)
   (trans x y z (affin kx ky kz (trans -x -y -z A) ) ) )
```

On peut procéder de même pour les rotations d'axe quelconque, les affinités et les symétries de plans quelconques, etc, et enrichir ainsi les fonctions affines disponibles.

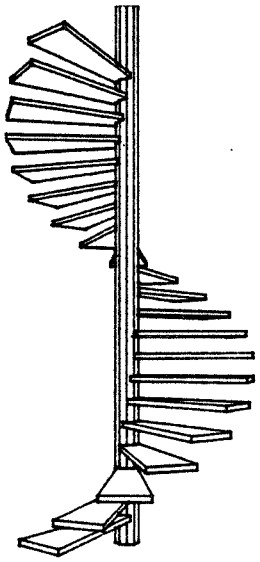


Fig. 8 : Exemple d'objet défini par une répétition

Répétitions : LISP permet de définir quelques opérateurs puissants et bien commodes, par exemple celui de répétition; ainsi un escalier de trois marches est l'union de la première marche, de la translation de cette marche, et de la translation de la deuxième marche. Plus généralement, la fonction "repeter" prend en argument un déplacement, un nombre d'itérations, et l'objet répété; LISP l'écrit en 4 lignes :

```
(de repeter (deplacement nb_fois objet)
   (cons 'union (literer deplacement nb_fois objet)))
(de literer (f n arg)
   (if (eq 0 n) (list arg) (cons arg (literer f (1- n) (funcall f arg)))))
```

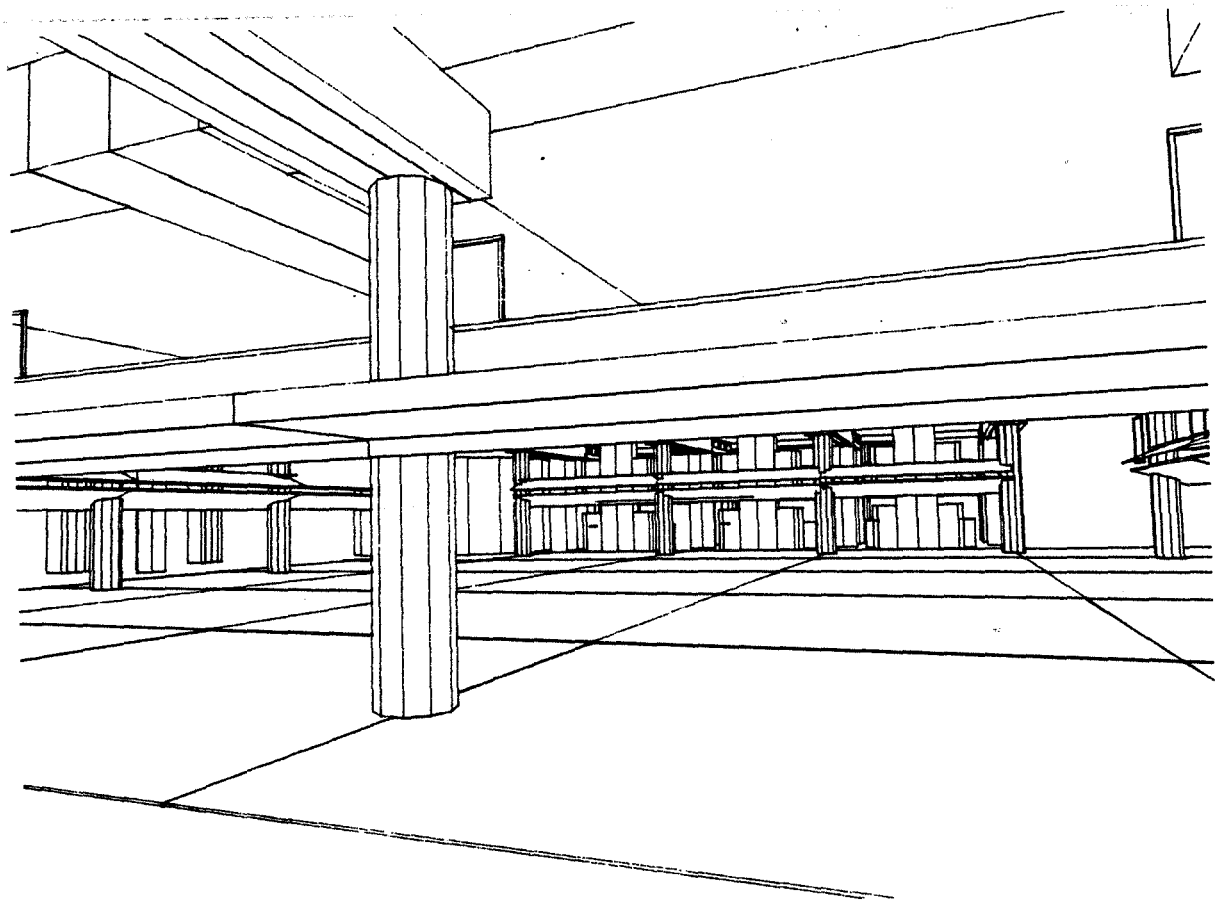
Un escalier hélicoïdal se définit ensuite en 5 lignes :

```
(union
  (affin 10 10 600 (cylindre 100 100 100 0)) ;le pivôt
  (repeter
    (lambda(marche) (trans 0 0 18 (rotat 0 0 1 18 marche))); le déplacement
    (truncate (/ 360.0 18.0))    ; le nombre de marches
    ; l'objet à répéter, la marche, est un prisme :
    (trans 0 0 18 (affin 1 1 4
                  (prisme 100 200 34 0 (6 -8 100 -18 100 18 6 8)))
    )
  )
)
```

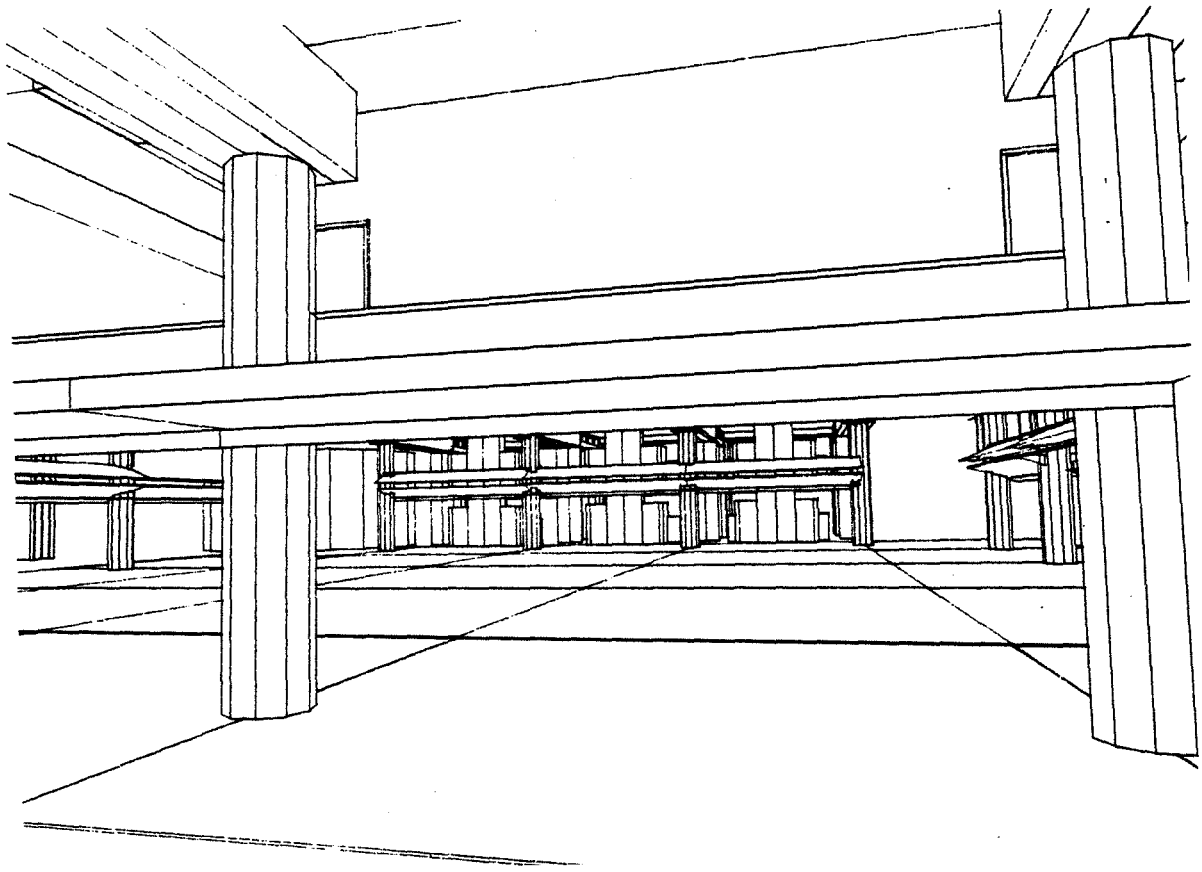
Objets paramétrés : LISP permet d'écrire des fonctions rendant des objets paramétrés, par exemple des escaliers hélicoïdaux, ou des meubles divers. A titre d'exemple, (cubfrac 0 0 0 100 2) engendre la description CSG d'un "cube fractal", avec :

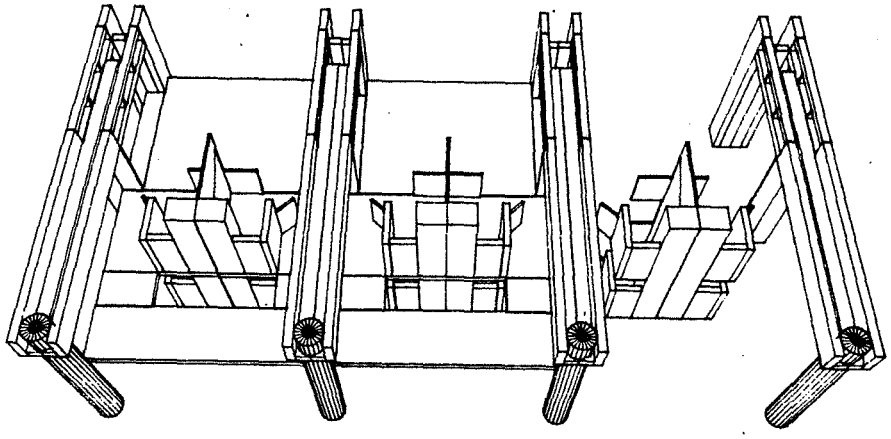
```
(de cubfrac ( x y z cote niveau_recursion)
 (if (eq 0 niveau_recursion)
    (un_cube x y z cote)
    (let ((k (cubfrac (- x (/ cote 2.0)) (- y (/ cote 2.0))
                      (- z (/ cote 2.0)) (/ cote 3.0) (1- niveau_recursion))))
         (union (un_cube x y z cote)
                (trans 0 0 0 k) (trans 0 0 cote k)
                (trans 0 cote 0 k) (trans 0 cote cote k)
                (trans cote 0 0 k) (trans cote 0 cote k)
                (trans cote cote 0 k) (trans cote cote cote k)
         )
    )
)
)
(de un_cube (x y z c)
   (trans x y z (affin c c c (trans -0.5 -0.5 -0.5 (cube 100 100 100 0)))))
```

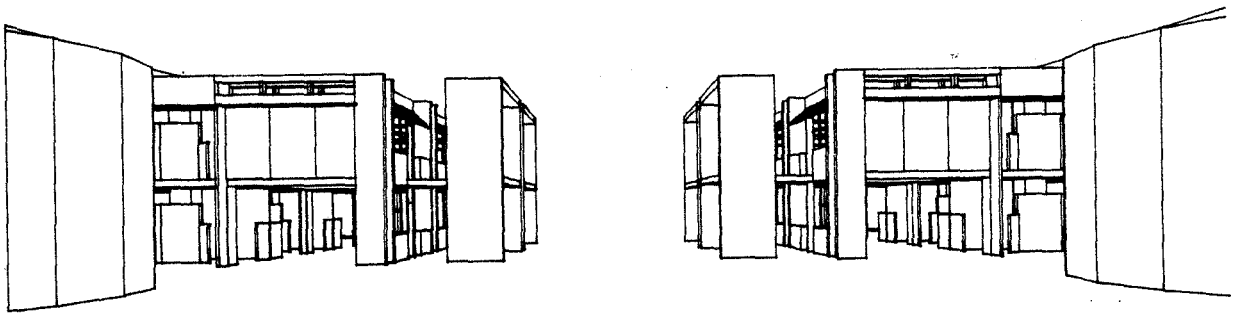
Cette description ne permet hélas pas d'"interroger" la scène, de savoir par exemple quel est l'objet présent en un point donné, ou sous un point donné; ce genre d'opérateurs permettrait, entre autres choses, de poser des objets sans préciser explicitement l'altitude. Le problème de la modélisation est traité plus longuement dans [BEIG 87].

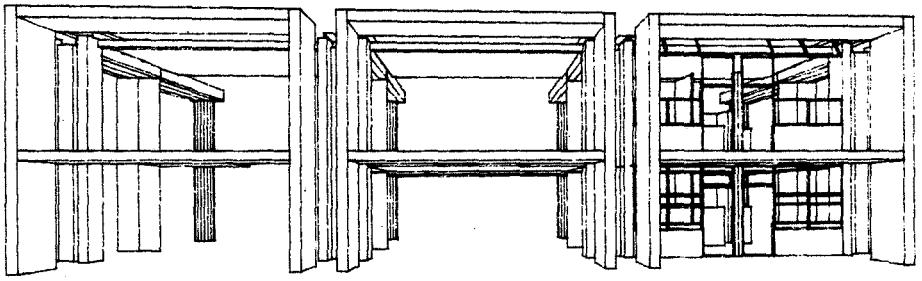


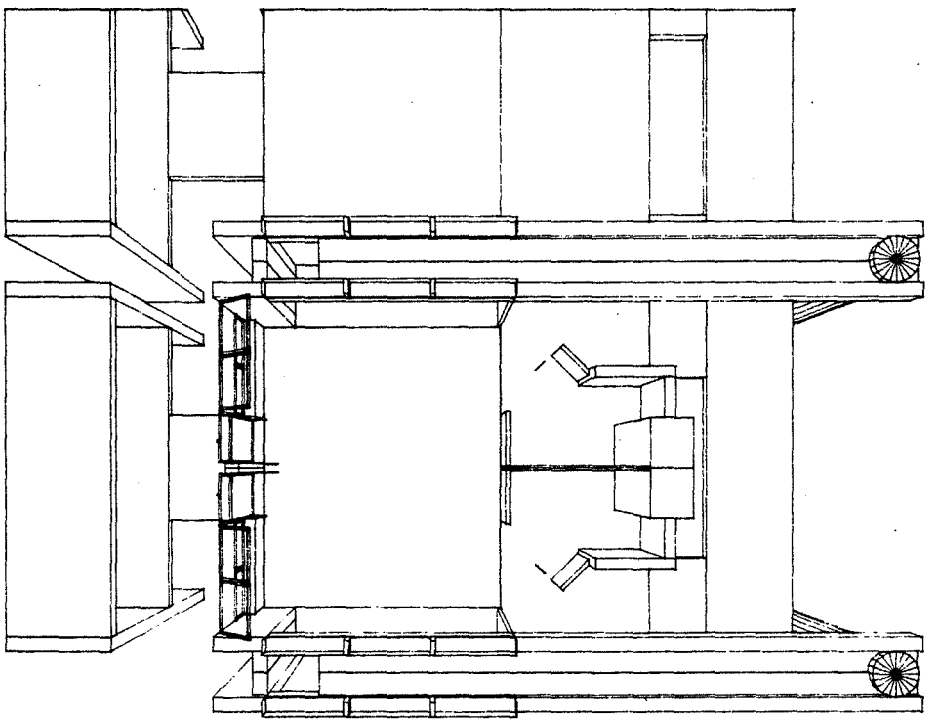
Ces images sont celles d'un projet d'architecture de CHOPIN et al. La saisie a été effectuée par CHOPIN, qui a utilisé le langage de modélisation CASTOR de BEIGBEDER [BEIG 87]. Après facétisation, la scène est visualisée par le logiciel présenté dans ce chapitre. On note que l'arbre CSG ne porte que des unions. Pour chaque image, le programme met environ dix minutes pour lire les données et vingt minutes pour calculer la carte planaire de l'image, sur HP9000, sous HP-UX, en C.











Chapitre 3

REPRESENTATIONS PAR FRONTIERES DES POLYEDRES ET OPERATIONS BOOLEENNES

1 INTRODUCTION

Ce chapitre propose un algorithme déterminant les frontières des polyèdres booléens : un polyèdre booléen est le résultat d'une opération booléenne effectuée sur deux autres polyèdres A et B dont les frontières sont connues; ces opérations booléennes peuvent être l'union, l'intersection, la différence, etc.

Les opérations booléennes employées sont les opérations booléennes régularisées introduites par TILOVE en modélisation des solides ("*solid modeling*"); on se reportera à [TILO 80] pour une définition rigoureuse de la régularisation. Brièvement, on peut rappeler que la régularisation a pour effet d'éliminer des solides les portions "pendantes", n'adhérant pas à un intérieur. A titre d'exemple, l'intersection régularisée de deux cubes accolés est le solide vide.

On rappelle que les seize opérations booléennes peuvent toutes s'exprimer en fonction de l'intersection et du complémentaire; pour ces raisons nous ne traiterons que le problème du calcul de l'intersection $A \cap B$ et du complémentaire -A.

Seul est traité le cas des solides limités par des faces polygonales : les mots "polyèdre" et "solide" seront ici employés indifféremment. Le solide le plus simple est le solide vide; les autres polyèdres sont classiquement définis ainsi par MARTIN [MART 87] :

"*Les solides sont des objets dont le bord sépare l'espace en deux régions, l'intérieur et l'extérieur. Le bord est une surface fermée, formée de facettes planes polygonales. Le solide est indéformable, c'est à dire la distance entre deux de ses points est constante. Le solide peut être en plusieurs morceaux, il n'est pas nécessairement convexe mais les excroissances filiformes ou planes sont interdites.*"

Ces propriétés sont préservées par les opérations booléennes régularisées.

Précisons qu'un polyèdre peut être fini (cas d'un polyèdre régulier) ou infini (complémentaire d'un polyèdre régulier); qu'il compte un nombre quelconque d'anses, au sens topologique (0 pour la sphère discrétisée, 1 pour un tore ou une tasse facetisés, 2 pour un bretzel); qu'il peut contenir un nombre quelconque de trous.

[MART 87] décrit les polyèdres par les propriétés de leur bord, ce qui est peut-être malcommode. D'ailleurs [LAKA 84] montre bien les difficultés que ce type de description a causé dans les diverses démonstrations mathématiques de la formule d'EULER sur les polyèdres. Aussi peut-on préférer définir les polyèdres de cette façon :

Un polyèdre est l'objet résultant de la composition des opérations booléennes régularisées et de l'opération de complémentation régularisée sur des polyèdres triviaux. Il suffit d'ailleurs d'un seul polyèdre trivial : le cube ! Le cube et les opérations régularisées [TILO 80] se définissent simplement et sans ambiguïté.

C'est d'ailleurs cette définition qu'emploie l'arbre de construction, ou arbre CSG ("*Constructive Solid Geometry*") [REQU 80]. Par commodité, il ajoute en général au cube les quatre autres polyèdres réguliers (platoniciens), les polyèdres semi-réguliers, les prismes à base polygonale, les sphères, cônes, cylindres... Nous ne considérerons que des approximations polyèdriques de ces dernières primitives.

Le problème du calcul des frontières des solides booléens s'est posé très tôt en modélisation des solides; en effet, la description la plus naturelle des pièces mécaniques de la CAO est l'arbre de construction. Cette représentation ne donne hélas pas de description explicite des frontières du solide; or cette description est très utile pour l'usinage des surfaces et pour le calcul du volume, du centre de gravité ou des axes principaux d'inertie du solide, bref ses caractéristiques mécaniques les plus élémentaires. De plus, les algorithmes de visualisation de la synthèse d'images ont longtemps nécessité une connaissance explicite des frontières des objets; c'est le cas des méthodes du tampon en z, de WATKINS ou de NEWELL...

La détermination des frontières des solides booléens est enfin un problème théorique intéressant, mais encore relativement peu étudié par les auteurs de la géométrie algorithmique [CHAZ 80] [MULL 78], par rapport aux problèmes de l'enveloppe convexe ou aux problèmes 2D. A mon sens, il y a deux raisons essentielles :

- Les auteurs des modeleurs 3D ont été confronté à ce problème et d'une façon ou d'une autre ont dû le résoudre, au moins partiellement; mais les algorithmes utilisés par les logiciels industriels ne sont pas souvent publiés;

- La littérature algorithmique est beaucoup plus riche sur ce même problème en 2D : le plan peut être balayé de façon simple et efficace, et des variantes de l'algorithme de BENTLEY-OTTMAN effectuent les opérations booléennes sur des polygones [OTTM 85] [PREP 85]. Mais ces solutions élégantes et rapides ne se généralisent pas facilement en 3D [PREP 85].

Le paragraphe 2 présente les principaux algorithmes existants, et les difficultés auxquelles ils sont confrontés. Le paragraphe 3 présente la structure de données décrivant les polyèdres. L'algorithme d'intersection est exposé en 4, et l'algorithme déterminant le complémentaire en 5. Le paragraphe 6 traite d'optimisation.

2 LES DIFFICULTES DU PROBLEME

2.1 RAPPEL DE QUELQUES METHODES CONNUES

Ce paragraphe va présenter une liste de quelques méthodes connues, et quelques brefs rappels qui permettront au lecteur de se les remémorer; ces rappels ont surtout pour but de montrer comment le nouvel algorithme proposé se démarque des méthodes existantes.

L'algorithme de MULLER et PREPARATA [MULL 78] détermine l'intersection de deux polyèdres convexes en O(NlogN), N étant la somme du nombre de sommets des deux polyèdres. On n'est pas encore sûr que cet algorithme soit optimal; après tout, l'intersection entre deux polygones convexes se détermine en O(N). De toutes façons, cet algorithme ne peut pas, semble-t-il, être généralisé aux polyèdres quelconques. Cette méthode utilise une structure de DCEL ("*Doubly-Connected Edge List*"), une variante de la "*Winged Edge Structure*" de [BAUM 75]; ce type de structure de données est utilisé par pratiquement toutes les méthodes.

CHAZELLE [CHAZ 80] partitionne les polyèdres en parties convexes, et effectue les opérations booléennes sur cette partition : le problème est réduit à l'intersection de deux polyèdres convexes, et à la partition en convexes. Selon la classification de REQUICHA [REQU 83], cette représentation des polyèdres est cependant plus une "énumération spatiale" qu'une représentation par frontières du polyèdre (ou BREP : "*Boundary-REPresentation*"). Cette approche semble assez lourde, et d'une programmation plus difficile que les méthodes qui suivent; les inconvénients et les avantages de cette représentation et de la méthode de CHAZELLE sont plus longuement discutés par MANTYLA [MANT 82]. On remarquera que les méthodes suivantes emploient assez souvent, pour les faces et non pour les polyèdres, une partition en convexes ou en triangles :

SZILVASI-NAGY [SZIL 84] effectue les opérations booléennes sur des polyèdres euléríens ("*manifolds*"). S'inspirant de la géométrie projective de MONGE, la méthode proposée détermine les points d'intersection entre les arêtes d'un polyèdre et les faces de l'autre polyèdre en raisonnant dans deux plans de projection Oxy et Oxz. L'algorithme emploie la méthode de BENTLEY-OTTMAN sur une des projections et a une bonne complexité théorique en O((N+S)logN) où N est la somme du nombre de sommets des deux polyèdres et S le nombre de points d'intersection entre toutes les arêtes dans le plan de projection Oxy; $S=O(N^2)$. Cependant les solides ne restent pas euléríens ("*manifolds*") par les opérations booléennes, et l'algorithme n'est pas réentrant; peut-être est-il possible de généraliser l'algorithme. Apparement, [SZIL 84] n'a pas été confronté à des difficultés particulières en employant la méthode de BENTLEY-OTTMAN (cf le chapitre I).

MANTYLA [MANT 83] s'est beaucoup préoccupé du problème des incohérences topologiques dans les BREP : comment être sûr que la structure de données produite décrit bien un polyèdre correct ? [MANT 83] prend bien soin de distinguer opérations géométriques et opérations topologiques, et il assure la validité topologique de la BREP produite en n'utilisant que des opérateurs euléríens (insertion et suppression d'un sommet, d'une arête...), qui conservent la cohérence de la topologie. On peut noter que les structures de données et les algorithmes sont loin d'être triviaux, malgré les soucis de modularité de MANTYLA; cette complexité est cependant inévitable, selon MARTIN et al. [MART 87].

Au passage, il convient de rappeler que, au contraire des BREP, les descriptions par des arbres de construction ou par des partitions spatiales (octrees, bintrees, tableaux de voxels) sont, elles, toujours cohérentes. Les BREP peuvent être incohérentes parce qu'elles sont d'une certaine façon redondantes. La redondance provoque systématiquement des problèmes d'incohérence en informatique : ainsi les systèmes de gestion de fichiers, les bases de données, ou les bases de connaissances de l'IA sont confrontés à ces difficultés.

Comme MANTYLA, YAMAGUCHI [YAMA 84] utilise les opérateurs d'EULER et une structure de données de type "*Winged Edge Structure*" ou DCEL pour décrire les frontières des polyédres. YAMAGUCHI triangule, sans perte de généralité, les faces susceptibles de s'intersecter : la triangulation est d'un coût mineur, et permet de simplifier quelque peu algorithmes et structures de données.

LAIDLAW et HUGHES [LAID 86] ne considèrent, eux, que des solides à faces convexes : cette condition de convexité est une autre façon de simplifier le problème, et ce sans aucune perte de généralité. Malgré tout, [LAID 86] recensent pas moins de 27 cas particuliers; ils détaillent le traitement des cas les plus complexes. Avec franchise, [LAID 86] reconnait que l'algorithme ne résoud pas absolument tous les cas de figure, à cause de problèmes d'imprécision numérique :

"*Floating-point granularity must be considered when implementing this algorithm. A CSG combinaison that strains many commercial solid modelers combines two units cube, one rotated N degrees first around the x-axis then around the y-axis and finally around the z-axis. Most commercial solid modelers fail when 0.5 degree < N < 1 degree. Our implementation is successful for N > 0.1 degree.*" [LAID 86] utilise quelques méthodes classiques pour corriger partiellement les effets de l'imprécision, comme les epsilons. Nous y reviendrons ultérieurement.

Enfin, [MART 87] et [TAKA 86], qui ne détaillent pas vraiment d'algorithmes pour les opérations booléennes, émettent quelques vues pénétrantes sur la question, dont se réclamera ultérieurement ce chapitre.

Ce rapide survol de quelques méthodes typiques a montré certaines difficultés du problème. A mon sens, il y a trois difficultés essentielles :

2.2 LE NOMBRE DE TESTS ENTRE FACES

Il est bien sûr nécessaire de calculer toutes les intersections entre les a faces de A et les b faces de B; la méthode naïve, qui compare toutes les faces de A à toutes les faces de B, est au moins en O(ab), en supposant que la comparaison de deux faces s'effectue en O(1). Dans l'absolu, il suffit de tester les faces qui s'intersectent vraiment, et les autres tests sont inutiles ! Cette phase de tests est la plus coûteuse; des optimisations classiques sont rappelées en 5.1; une optimisation originale, directement inspirée des techniques du "*Multi-Dimensionnal Searching*", est présentée en 5.2.

2.3 LES CAS PARTICULIERS

Le lecteur aura remarqué que les deux chapitres précédents ne traitaient pas les quelques cas particuliers : en 2D, ils étaient peu nombreux et pouvaient se régler trivialement. Il n'en est pas de même avec le problème traité par ce chapitre; les cas particuliers sont nombreux, enchevêtrés, pénibles : sommet d'un polyèdre sur un sommet, une arête ou une face de l'autre solide; arête d'un polyèdre sur une face de l'autre solide; arêtes partiellement ou totalement confondues; faces coplanaires. Ces cas particuliers font dire à MARTIN et al. [MART 87] :

"Les programmes de calcul d'intersection de solides définis par leur bord sont donc de gros programmes, délicats à mettre en oeuvre, si l'on souhaite qu'ils traitent correctement les très nombreuses situations pouvant réellement se présenter."

[MART 87] présente auparavant plusieurs cas particuliers (fig. 1 et 2) mal traités par de nombreux algorithmes, notamment ceux de MANTYLA et al. [MANT 83] [MANT 83b]. L'intérêt du travail fondateur de MANTYLA et al. corrections, mais "*qui ne font qu'augmenter encore la complexité d'implantation*",

selon ses propres termes.

L'algorithme proposé dans ce chapitre surmonte la difficulté des cas particuliers : il est concis, simple (on peut même dire : naïf), et ne compte plus qu'un seul et unique cas particulier, dont le traitement, détaillé en 4.3, nécessite un ajout mineur.

2.4 LES INCOHERENCES

La principale difficulté est due aux possibles incohérences des BREP; MANTYLA et al. voient là un problème strictement topologique, qu'ils veulent logiquement corriger par l'emploi des opérateurs d'EULER. Mon approche est différente; personnellement je pense, comme [TAKA 86], que l'incohérence est fondamentalement causée par l'imprécision numérique. Je rappelle brièvement que le premier chapitre a établi les faits suivants :

- Certaines configurations géométriques 2D ou 3D comme celles de [GANG 87] ne peuvent pas être bien représentées avec des coordonnées flottantes : il y a fatalement une contradiction entre des sommets topologiquement distincts mais numériquement indiscernables. Ces configurations sont mal traitées par absolument tous les algorithmes utilisant l'arithmétique flottante; certains n'y résistent pas;

- Même sur des configurations apparemment banales (cf I.4.2), l'imprécision numérique peut fausser certaines comparaisons : l'algorithme qui propage des résultats partiels propage alors des informations fausses, parfois jusqu'à une erreur fatale.

Dans la littérature spécialisée, ce phénomène est largement méconnu (ou passé sous silence); je ne connais que deux exceptions : [GANG 87] non publié, et [TAKA 86] bien que ce dernier ne cite pas d'exemples précis. Paradoxalement, les programmeurs en synthèse d'images ou en "*solid modeling*" y sont confrontés quotidiennement. Ecartons tout d'abord les solutions fausses ou partielles :

- L'emploi des epsilons, déjà proscrit en I.4.4; [LAID 86] utilise certes les epsilons, mais reconnait que cette méthode n'est pas totalement satisfaisante. Les epsilons peuvent cependant être employés avec profit par les algorithmes et les BREP tolérant l'incohérence. A ma connaissance, les seules BREP tolérant parfaitement l'incohérence sont celles de TAKALA : elles sont présentées plus loin;

- L'emploi des opérateurs d'EULER; il garantit certes la cohérence de la topologie, mais il ne règle pas les incohérences numériques, ou les contradictions entre les données topologiques et numériques.

J'envisage quatre solutions possibles au problème des incohérences numériques :

- Renoncer aux BREP; le problème n'est pas résolu, plutôt éludé; l'avenir choisira peut-être bien cette voie; après tout, la synthèse d'images peut d'ores et déjà se passer des BREP. Dans le futur, la CAO pourrait opter pour d'autres modèles de représentation, comme celui de la partition spatiale (octrees, tableaux de voxels...);

- Savoir gérer des "nombres flous", des "ordres flous", des "topologies floues", en bref : des BREP floues. L'approche de ZADEH, ou la théorie des possibilités, sont employées avec quelque succès pour modéliser le flou des connaissances ou de la logique humaines en IA. Mais cette théorie traite fondamentalement des données qualitatives, discrètes (le vrai, le faux, et les valeurs entre les deux), non intrinsèquement continues; il me paraît difficile de l'employer ici. De plus il serait quand même bien agréable de pouvoir réutiliser les résultats de la géométrie euclidienne ou de l'algèbre linéaire, toute une tradition mathématique de raisonnements exacts dans R^n. Cette approche est pourtant citée, car [CASA 87] emploie les ensembles flous, mais sans préciser ces techniques;

- Assumer l'incohérence des BREP; cette approche est celle de TAKALA [TAKA 86]. A titre d'exemple, pour cette approche, un polygone est simplement une liste quelconque de couples de points, un couple par arête : TAKALA n'impose pas que les arêtes se raccordent exactement aux sommets. Pour décider si un point se trouve à l'intérieur ou à l'extérieur de ce polygone, TAKALA utilise une variante statistique du principe de parité; il calcule les nombres de points d'intersection entre les arêtes et plusieurs demi-droites issues du point; si ces nombres sont le plus souvent pairs, alors le point est extérieur, sinon il est intérieur. Remarquons que l'imprécision numérique fausse parfois certains tests d'intersection, et que cela n'a aucune conséquence dommageable. TAKALA utilise systématiquement ce genre d'algorithmes probabilistes, extrêmement robustes, qui tolèrent les incohérences et ne les propagent pas. Il ne détaille pas ses algorithmes pour les opérations booléennes, mais son article est suffisament suggestif pour que le lecteur de [TAKA 86] puisse les imaginer. L'approche de TAKALA est extrêmement séduisante par sa simplicité et son efficacité conceptuelle. On peut noter qu'elle exige une très grande puissance de calcul, car la propagation des informations, toujours suspectes, est interdite, et tous les tests doivent être effectués plusieurs fois (l'approche de TAKALA n'est pas sans rappeler la méthode "Perturbation-Permutation"). On peut cependant se demander si ce type de BREP, sans aucune information de

nature topologique, ne perd pas quelques informations essentielles : d'ailleurs, est-ce toujours une BREP ?

- Ne pas commettre d'erreurs de calcul, grâce à l'emploi de solides rationnels et d'une arithmétique rationnelle.

Cette dernière solution est celle que nous allons utiliser : elle a déjà été présentée et évaluée en II.7.3. Cette solution appelle quelques commentaires :

- Cette approche rend inutiles les opérateurs d'EULER : en composant l'intersection et le complémentaire régularisées sur des descriptions cohérentes, nous n'obtiendrons jamais que des représentations cohérentes;

- Au moins deux implantations sont possibles pour l'arithmétique sur des entiers de longueur finie non bornée [KNUT 81]. La première ne fait que transposer les opérations manuelles, bien qu'employant en général une base non décimale pour des raisons d'efficacité. La deuxième est l'arithmétique modulaire; là, un entier n'est pas représenté par une suite de chiffres dans une base donnée, mais par la liste de ses restes (e_1, e_2 ... e_n) modulo n nombres premiers (p_1, p_2 ... p_n), avec $p_1 < p_2 < ... p_n$; d'après le théorème des restes chinois, cette représentation permet de décrire $p_1p_2...p_n$ entiers; la forme décimale des entiers peut être retrouvée à partir de la forme modulaire. Le test de nullité d'un entier, l'addition, la soustraction et la multiplication de deux entiers s'effectuent simplement et en O(1) sur chacun des modules; la division est plus lente et requiert O(n log p_n) opérations au total; de même pour le classement de deux entiers. Cette arithmétique est utilisée avec succès pour résoudre efficacement et de façon exacte de très grands systèmes d'équations linéaires à coefficients entiers ou rationnels : on effectue peu de divisions, et pas du tout de comparaisons, à part le test trivial de nullité du pivot; l'arithmétique modulaire semble cependant moins indiquée pour notre problème, pourtant voisin, car ce dernier nécessite de très nombreuses comparaisons. En outre, il paraît malvenu de limiter a priori la taille des entiers nécessaires; les tests de débordement de capacité avec l'arithmétique modulaire ne sont d'ailleurs pas évidents;

- Les calculs effectués ne doivent pas faire sortir de Q; tel est bien le cas : les calculs de puissance d'un point par rapport à une droite ou un plan, le classement des arêtes coplanaires incidentes en un sommet (ce problème se posait déjà en I), l'intersection de deux plans ou d'un plan et d'une droite... ne nécessitent bien sûr que les produits scalaires et vectoriels, c'est à dire les opérations (+ - * / < =). (Remarque : les

cosinus, sinus, tangentes... manipulés ont tous des carrés rationnels);

- La triangulation des faces est uniquement un moyen commode d'obtenir des polyèdres primitifs rationnels : l'algorithme d'intersection proposé n'exige en rien la triangulation des faces, qui peuvent être concaves ou non connexes; d'ailleurs, l'algorithme proposé génère des polyèdres A∩B aux faces quelconques, et il est réentrant : il arrive donc que les polyèdres données A et B aient des faces non triangulaires, non convexes, non connexes, tout à fait quelconques. (Ceci est à opposer à [YAMA 84], qui triangule les faces des polyèdres d'entrée, et à [LAID 86] qui n'accepte que des faces convexes).

3 REPRESENTATION DES POLYEDRES

Nous décrivons maintenant la structure de données représentant les polyèdres d'entrée A et B, et le polyèdre résultat A∩B. Il va de soi que ces polyèdres sont rationnels.

3.1 LA BREP : STRUCTURES DE DONNEES

Un polyèdre est décrit par :

- un booléen disant si le polyèdre est fini ou infini;
- la liste des faces du polyèdre. Ces faces peuvent être concaves et non connexes.

Une face est décrite par :

- (a, b, c, d) : un quadruplet d'entiers non bornés; ax + by + cz +d = 0 est l'équation du plan de la face; (a b c) est un vecteur normal à la face, dirigé vers l'extérieur; cette convention est arbitraire. Il est commode d'imposer PGCD(a, b, c, d) = 1;
- la liste des pans de la face.

Un pan est l'incidence d'une face en une arête, d'un coté bien déterminé de cette arête; en effet, la droite portant l'arête partage en deux demi-plans le plan de la face, et le pan "connaît" le demi-plan où se trouve la face, le long de l'arête du pan. Le pan est donc une arête orientée, un prolongement de la notion de brin des cartes planaires.

Informatiquement, un pan est décrit par :

- un pointeur sur l'arête du pan;
- le sens du pan, qui vaut 1 ou -1. Le sens permet de savoir de quel coté de l'arête se trouve le pan; son fonctionnement est précisé ultérieure-

ment;
- un pointeur vers le pan suivant de la même face;
- (a, b, c, d) : un quadruplet semblable à celui de la face du pan; les raisons de cette redondance apparaitront en 4.2;
- ls : une liste des solides dont ce pan fait partie : au cours de l'algorithme décrit en 4.2, un pan peut en effet appartenir à la fois à A et à B.

Une arête est décrite par :

- un couple (p,q) de points. p est le point inférieur de l'arête, q est le point supérieur : les points de R^3 sont en effet ordonnés lexicographiquement, sur x, puis y, puis z. Cette convention est arbitraire.

Un point est un quadruplet d'entiers (x, y, z, t); ce sont les coordonnées homogènes du point rationnel (x/t y/t z/t); il est commode d'imposer t > 0 et PGCD(x, y, z, t)=1. Ces enregistrements peuvent ou non être partagés par les arêtes incidentes en un même point : c'est là un détail d'implantation sans aucune importance. Par contre, une arête, qui est toujours référencée par plusieurs pans, n'est décrite qu'une seule fois dans la structure de données.

Le sens des pans permet de savoir de quel coté de l'arête [p,q] se trouve la face : la droite (pq) coupe en effet le plan de la face en deux demi-plans. Le sens se définit ainsi : soit m un point quelconque sur [p,q], soit pq le vecteur porté par l'arête [p,q], et soit le vecteur mn = (sens*pq) ^(a b c), ^ désignant le produit vectoriel et * le produit scalaire; le vecteur mn est dans le plan de la face et perpendiculaire à la droite (pq); localement à l'arête [p,q], la face doit se trouver dans le demi-plan contenant le point n : cette convention fixe la valeur du sens des pans. Les vecteurs mn sont réutilisés plus loin, et appelés "vecteurs indicateurs" des pans.

Cette description d'un polyèdre doit vérifier les contraintes de cohérence suivantes :

- deux arêtes a et b distinctes ne peuvent s'intersecter qu'en un point extrêmité de a et extrêmité de b;
- deux faces distinctes ne s'intersectent que sur des arêtes présentes dans la description de ces faces;
- il n'existe pas d'arête [p,q] seulement référencée par deux pans coplanaires. Cette contrainte élimine les arêtes inutiles des figures 3, 4, 5. Cette contrainte n'est pas vraiment indispensable, mais simplifie grandement la démonstration du 4.1.2.

Observons les conséquences de ces contraintes sur la description de quelques solides. Sur la figure 6, cinq arêtes sont incidentes au même sommet. Sur la figure 7, quatre arêtes sont incidentes en un même sommet. Sur la figure 8, l'arête [p, q] est référencée par quatre pans, appartenant à trois faces distintes.

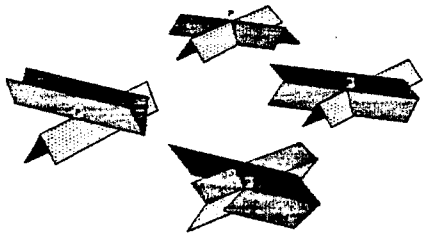


Fig. 1 (MART 87)

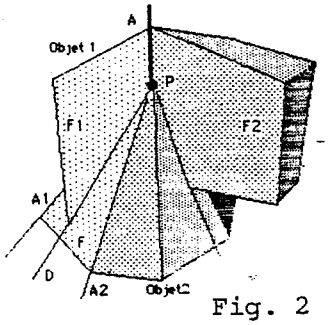


Fig. 2 (MART 87)

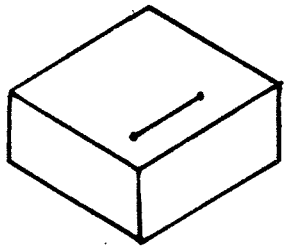


Fig. 3

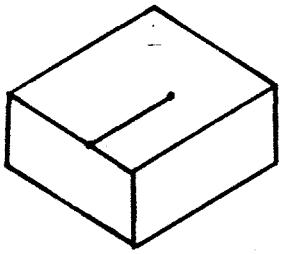


Fig. 4

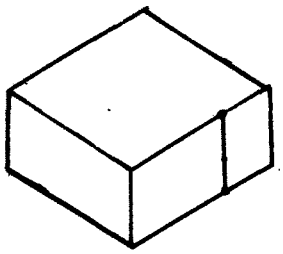


Fig. 5

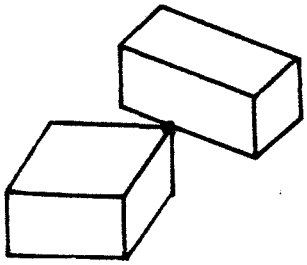


Fig. 6

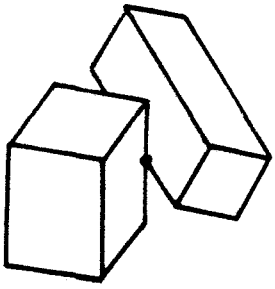


Fig. 7

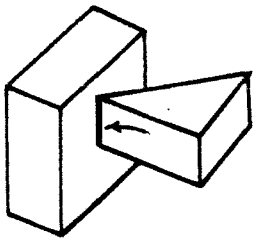


Fig. 8

Remarquons que pour la structure de données proposée, ces cas n'ont rien de particulier. Il n'existe qu'un seul cas particulier, traité ultérieurement.

(On peut éventuellement imposer qu'une description ne contienne pas de sommets inutiles, comme celui de la figure 9, et qu'il n'y ait pas deux faces coplanaires de même orientation; ainsi, à l'ordre près des termes dans les listes, la description devient unique)

3.2 INTERET DE CETTE BREP

3.2.1 La prise en compte des cas particuliers

Cette structure de données choisie pour décrire les polyèdres est d'une brutalité extrême quand on la compare aux variantes de la "*Winged Edge Structure*" utilisées dans [ANSA 85], [YAMA 84], [MANT 83], [LAID 86]... et d'autres méthodes. A

ma connaissance, c'est la première fois qu'est proposée une structure de données aussi fruste pour un problème réputé aussi complexe ([TAKA 86] propose aussi des descriptions brutales, mais il ne garantit plus la cohérence).

Cette structure de données ne permet pas de savoir immédiatement de combien de morceaux est fait le solide; elle ne permet pas d'accéder directement à toutes les faces incidentes en une arête; elle ne donne pas les contours d'une face; elle ne donne pas les arêtes incidentes en un sommet... mais par sa minimalité, elle permet d'unifier simplement de très nombreux cas particuliers (tous, sauf un), comme :

- les faces trouées des figures 10, 11 et 12;
- les arêtes multiples des figures 13 et 14;
- les sommets multiples des figures 6, 12 et 15.

Il n'existe qu'un seul cas particulier, celui de la figure 16 : le sommet situé strictement à l'intérieur d'une face; dans un premier temps, ce cas est ignoré dans l'exposé de l'algorithme; il est pris en compte en 4.3.

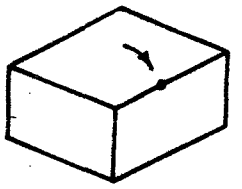


Fig. 9

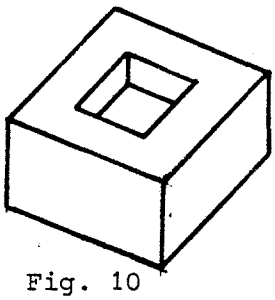


Fig. 10

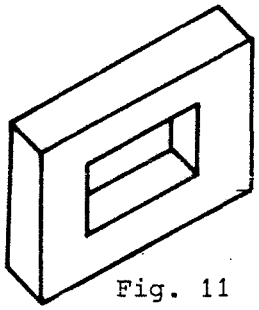


Fig. 11

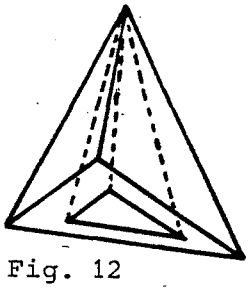


Fig. 12

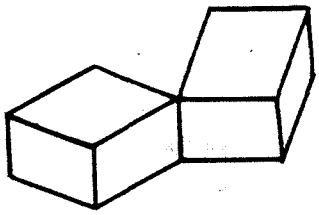


Fig. 13

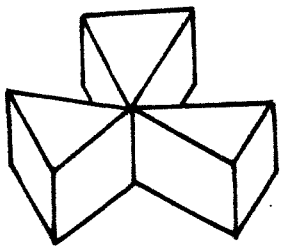


Fig. 14

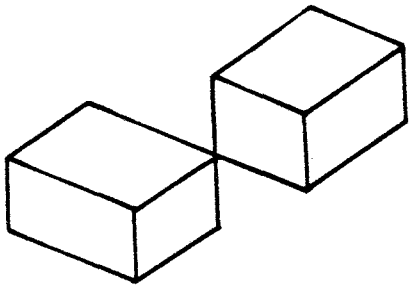


Fig. 15

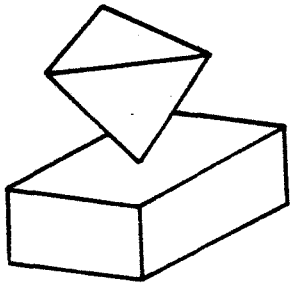


Fig. 16

3.2.2 La regénération d'une description explicite

Cette structure de données contient, de façon implicite, toutes les informations explicites des diverses "*Winged Edge Structure*" utilisées par [ANSA 85], [YAMA 84], [MANT 83], [MART 87] ... Un parcours de la structure permet de regénérer toutes les informations implicites. Ce n'est pas le cas pour les structures de [TAKA 86], non forcément cohérentes.

Par exemple, supposons que nous voulions connaître, pour chaque sommet, la liste des adresses des arêtes incidentes, et, pour chaque arête, la liste des adresses des faces incidentes. Il suffit alors d'utiliser une "*hash table*" HS, initialement vide, où sont stockés des sommets qu'on accède par le quadruplet de leurs coordonnées homogènes entières (x, y, z, t) qui sert de clé. Un sommet S est un enregistrement d'un type nouveau, contenant des coordonnées et une liste S.arcs initialement vide de pointeurs vers les arêtes dont S est une extrêmité. Enfin sur chaque arête a est greffée un nouveau champ : a.faces, qui est la tête de la liste initialement vide des adresses des faces incidentes en l'arête. Alors la procédure regénérer() retrouve les arêtes incidentes aux sommets, et les faces incidentes aux arêtes :

```
procédure regénérer (A : polyèdre) =
{
HS ← ∅;
pour chaque face f de A faire
   pour chaque pan P de f faire
     {
       a ← arête de P;   /* a = [p, q] */
       a.faces ← f ∪ a.faces ;
       s1 ← le-sommet( xp, yp, zp, tp );
       s2 ← le-sommet( xq, yq, zq, tq);
       s2.arcs ← a ∪ s1.arcs;
       s1.arcs ← a ∪ s2.arcs;
   }
}


fonction le-sommet (x, y, z, t : entier) =
{ si ( s de coordonnées (x, y, z, t) ∈ HS)  rendre (s)
  sinon { s ← créer-sommet(x, y, z, t);
          HS ←  s ∪ HS ;
          rendre (s);
        }
}
```

Il est aussi possible de reconstruire les contours d'une face : cela revient à déterminer les diverses composantes connexes d'un graphe connaissant les voisins de chaque sommet.

La regénération de ces informations ne nécessite aucun calcul géométrique, mais seulement un parcours linéaire des structures de données; elle est donc en $\Theta(S)$, S étant le nombre de sommets.

Les seuls calculs géométriques éventuellement nécessaires sont ceux qui situent les diverses composantes connexes les unes par rapport aux autres : par exemple, les divers contours d'une face entre eux (cf I), ou les diverses parties de l'enveloppe du polyèdre entre elles; ce problème est traité en 4.2.3; la regénération de ce type d'information n'est pas plus difficile, mais plus lente :

Situer deux composantes connexes l'une par rapport à l'autre se réduit à situer un point quelconque de l'une par rapport à l'autre (cf 4.2.3); si cette dernière compte N arêtes, O(N) opérations sont nécessaires; il s'agit d'opérations rationnelles, et elles ne s'effectuent pas en O(1). L'emploi des boites englobantes, ou d'autres méthodes présentées en 6, peuvent accélérer cette phase.

Le principe de cette BREP est de n'utiliser que les structures de données juste suffisantes; ainsi les informations superflues n'ont pas à être gérées : l'exemple des méthodes existantes a montré combien cette gestion est difficile.

4 ALGORITHME D'INTERSECTION

Exposons maintenant l'algorithme de détermination des frontières du polyèdre rationnel $A \cap B$, intersection de deux polyèdres rationnels A et B donnés. Il va de soi que tous les calculs sont effectués en arithmétique rationnelle. Cet algorithme comporte trois étapes :

- la première détermine une liste des couples de faces de A et de B qui sont susceptibles de s'intersecter; tous les couples de faces (f de A, g de B) qui s'intersectent vraiment, ne serait-ce qu'en un seul point, doivent figurer dans cette liste : sinon des intersections entre faces seraient oubliées, ce qui serait fatal à l'algorithme. Par contre cette liste peut contenir quelques couples de faces qui ne s'intersectent pas. L'idée est bien sûr d'éviter les O(ab) tests de l'algorithme naïf par un prétraitement efficace. Malgré son importance cruciale sur les performances, ce prétraitement est totalement indépendant de l'algorithme lui-même; aussi les diverses techniques envisageables sont-elles présentées à part, dans le paragraphe 6.

- la deuxième étape calcule effectivement les intersections entre les faces; les arêtes initiales des deux solides sont découpées en sous-arêtes homogènes; les deux cas particuliers : (sous-arête incluse dans une face de l'autre solide, sous-arête incluse dans une arête de l'autre solide) sont pris en compte par les structures de données de façon très naturelle. Le

traitement du cas particulièrement complexe de l'intersection de deux faces coplanaires est totalement explicité.

- la troisième étape détermine l'utilité de tous les pans : un pan est utile quand il doit faire partie de la description de A∩B. L'utilité, ou l'inutilité, se propage selon quelques règles récursives simples, ce qui minimise les calculs nécessaires : il est inutile de calculer pour chaque sommet de A (respectivement de B) s'il est inclus ou non dans le solide B (respectivement A); ce calcul est en effet fort coûteux. Finalement, une suppression simple des pans marqués inutiles génère la description du solide résultat.

4.1 CALCUL DES INTERSECTIONS ENTRE FACES

C'est par la deuxième étape que nous commençons l'exposé de l'algorithme. Elle calcule l'intersection de tous les couples de faces "suspects" d'intersection.

Une liste d'"arêtes d'intersection", initialisée à vide, contiendra à la fin de cette phase toutes les arêtes nées des intersections ordinaires entre une face de A et une face de B : une "arête d'intersection ordinaire" est une arête d'intersection qui ne se trouve ni sur une arête de A ni sur une arête de B. Pour chacune de ces arêtes d'intersection sont stockées : les coordonnées des deux extrêmités, la face de A et la face de B qui s'intersectent. Les extrêmités des arêtes d'intersection peuvent être des sommets de A et/ou de B, des points d'intersection entre une arête de A et une arête de B, ou plus simplement l'intersection banale entre une arête d'un solide et une face de l'autre solide : l'algorithme ne considère aucun de ces cas comme particulier.

De plus, chaque arête de A, et chaque arête de B, va être découpée en sous-arêtes homogènes; "homogène" signifie ici qu'une sous-arête est soit incluse dans l'autre solide, soit hors de l'autre solide, soit incluse dans une arête de l'autre solide, soit incluse dans une face de l'autre solide sans être incluse dans une arête de cette face.

Les sous-arêtes de chaque arête sont mémorisées dans une liste ordonnée sur les extrêmités. Pour chaque sous-arête a, sont déterminées et stockées les coordonnées de l'extrêmité inférieure, l'éventuelle arête de l'autre solide dans laquelle a est incluse (nil le plus souvent), et l'éventuelle face de l'autre solide dans laquelle a est incluse (nil le plus souvent). Comme les extrêmités des arêtes d'intersection, les extrêmités des sous-arêtes peuvent être des sommets de A et/ou de B, des points d'intersection entre une arête de A et une arête de B, ou plus simplement l'intersection banale entre une arête d'un solide et une face de l'autre solide : l'algorithme ne considère aucun de ces cas comme particulier.

Remarquons que certaines informations vont être trouvées plusieurs fois; ainsi un point d'intersection ordinaire entre une arête et une face est trouvé en général deux fois, une fois pour chaque face incidente en cette arête. Avant de couper une sous-arête en deux par un point d'intersection, il convient donc de vérifier que ce n'est pas déjà fait.

Deux faces parallèles non coplanaires ne peuvent bien sûr pas s'intersecter; a priori, ces couples ne doivent même pas figurer dans la liste des faces suspectes. Détaillons d'abord l'intersection de deux faces sécantes (non coplanaires et non parallèles), puis l'intersection de deux faces coplanaires.

4.1.1 Intersection entre deux faces sécantes

La détermination de l'intersection entre deux faces sécantes f de A et g de B doit tenir compte de tous les cas particuliers possibles, fort nombreux. Elle peut cependant être effectuée d'une façon simple : l'idée consiste à calculer d'abord l'intersection entre chaque face et le plan de l'autre face, par une procédure détaillée ultérieurement.

Soit D la droite d'intersection des deux plans. L'intersection entre f et le plan de g, c'est à dire l'intersection entre f et la droite D, est alors connue et représentée par une suite d'intervalles ordonnés lexicographiquement; pour chaque intervalle sont connus : les coordonnées de l'extrêmité inférieure, i; l'arête qui coupe D en i si i n'est pas un sommet de f; et, s'il y a lieu, l'arête de f confondue avec cet intervalle, c. L'intersection entre g et D est représentée de la même manière.

L'intersection entre f et g s'obtient alors par un parcours simultané de F∩D et de g∩D (cette procédure est une variante de la fusion de deux listes ordonnées, exercice classique en programmation) :

Si un intervalle appartient à la fois à f et à g, et ne contient aucune arête de f ou de g confondue avec D, alors une arête d'intersection ordinaire entre f et g vient d'être découverte, et doit être ajoutée dans la liste des arêtes d'intersection; remarquons au passage qu'une arête d'intersection n'est trouvée qu'une seule fois par l'algorithme; les extrêmités de cette arête sont connues. Quand une de ces extrêmités est un nouveau point d'intersection ordinaire entre une face et une arête a, la liste des sous-arêtes de a doit être remise à jour.

Si cet intervalle contient une arête u de f, appartient à g, et ne contient pas d'arête de g, alors une partie u' de l'arête u se trouve sur la face g; les extrêmités de la sous-arête u' sont connues; la mise à jour de la structure de données décrivant les sous-arêtes de u va de soi. De même, quand un intervalle contient une arête de f et une arête de g, une confusion entre une arête de f et une arête de g est alors détectée le long d'une sous-arête bien déterminée.

Le calcul de f∩D et de g∩D simplifie donc énormément le calcul de f∩g. La procédure déterminant l'intersection d'une face f et d'un plan Q sécants est maintenant détaillée. Cette procédure tient compte de tous les cas particuliers : arête ou sommet situé sur la droite d'intersection, sommet de degré supérieur à 2... (Bien sûr, la face est cohérente, et tous les calculs sont effectués sans imprécision aucune en arithmétique rationnelle)

Quelques définitions préalables sont nécessaires. Soit Q, un plan. On définit, arbitrairement, le "dessus" de Q comme étant l'ensemble des points de puissance strictement positive par rapport à Q. Un contact entre une arête a de f et le plan Q est représenté par un enregistrement du type : ((x y z t) segment indice). (x y z t) sont les coordonnées homogènes entières du point d'intersection; elles permettent d'ordonner lexicographiquement les contacts le long de D, la droite d'intersection entre les plans des deux faces. Le champ "indice" mémorise si des points de a se trouvent strictement au dessus de Q : 0 pour non, 1 pour oui; par exemple, cet indice vaut toujours 1 pour une arête a ayant avec Q un point d'intersection "ordinaire", strictement compris entre les sommets de a, et il vaut toujours 0 pour une arête appartenant à D. Le champ "segment" est un pointeur sur une arête, nil dans le cas général; dans le cas particulier où a = [p, q] est sur D, on considère qu'il existe deux contacts : ((px py pz pt) 0 a) et ((qx qy qz qt) 0 nil); comme la face est cohérente, ces deux contacts sont forcément consécutifs sur D : un troisième contact ne peut s'intercaler entre eux.

L'algorithme considère successivement chaque arête de f, et détecte les contacts :

```
f_inter_Q (f: face; Q:plan)=
{
 ensemble des contacts ← ∅ ;
 pour chaque pan de f, d'arête a = [p, q] de f
 {
   selon que  ([p, q] ∩ Q) =
    {
    • vide : ne rien faire;
    • i, avec i ≠ p et i ≠ q : créer_contact ((ix iy iz it) 1 nil);
    • [p, q] : {  créer_contact ((px py pz pt) 0 a);
                  créer_contact ((qx qy qz qt) 0 nil);  }
    • p, et puissance (q) > 0 : créer_contact((px py pz pt) 1 nil);
    • p, et puissance (q) < 0 : créer_contact((px py pz pt) 0 nil);
    • q, et puissance (p) > 0 : créer_contact((qx qy qz qt) 1 nil);
    • q, et puissance (p) < 0 : créer_contact((qx qy qz qt) 0 nil);
   }
 }
 trier les contacts sur leurs coordonnées;
 fusionner les contacts de coordonnées égales;
}
```

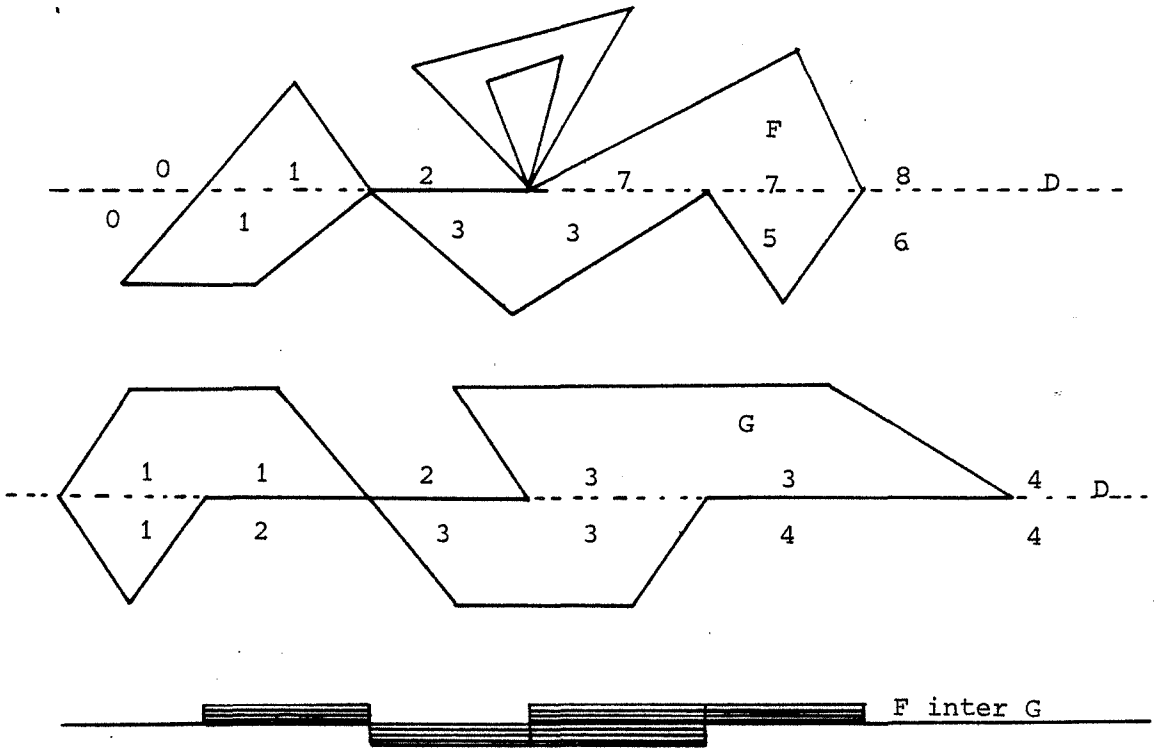


Fig. 17 : Intersection entre deux faces sécantes quelconques

Des contacts successifs ont des coordonnées égales dans les cas particuliers où un sommet de coordonnées m se trouve sur la droite d'intersection D; alors l'indice du contact de fusion (cf) s'obtient en additionnant les indices des contacts fusionnés : l'indice de cf est donc le nombre d'arêtes de F incidentes en m et ayant des points au dessus de Q.

Comment déterminer le segment d'un contact de fusion cf ? Deux cas sont possibles : soit les segments des contacts à fusionner valent tous nil, et alors le segment de cf vaut nil lui aussi; soit un seul des contacts à fusionner porte un segment différent de nil, dont hérite cf. Cette propriété s'explique par la cohérence de f, elle est évidente sur une figure.

Après les éventuelles fusions, une suite ordonnée de contacts le long de D est alors disponible. Deux contacts successifs i et j déterminent implicitement un intervalle (i, j). Si i porte un segment (qui ne peut être que [i, j]) alors (i, j) appartient bien sûr à f∩Q. Un intervalle (i,j) sans segment appartient à f∩Q si et seulement si la somme de tous les indices des contacts précédant j est impaire : simple application du principe de la parité (les sommes sur les indices peuvent donc se faire

modulo 2).

4.1.2 Intersection entre deux faces coplanaires

Le problème de l'intersection de deux faces coplanaires f de A et g de B est en général considéré comme particulièrement complexe. Aussi son traitement est-il maintenant explicité jusqu'au moindre détail. La programmation est relativement aisée; en effet, dans le cas de deux faces coplanaires, l'algorithme ne fait rien du tout !

Il convient de justifier cette réponse un peu surprenante; l'objectif de cette étape est d'obtenir une liste d'arêtes d'intersection et de sous-arêtes homogènes, et de leurs pans incidents. Considérons une arête quelconque de f, qui se trouve au moins partiellement dans la face g. Vu la cohérence de A, il arrive au moins une autre face de A en cette arête, f', non coplanaire avec f et g. L'appartenance à g de la sous-arête en question sera (ou a été) détectée lors du calcul de l'intersection de f' et de g. Même raisonnement pour toutes les sous-arêtes de f contenues dans g, et pour toutes les sous-arêtes de g contenues dans f. Le problème des arêtes partiellement ou totalement confondues de f et g se règle de la même manière.

Cette démonstration utilise le fait qu'un solide ne contient pas d'arêtes inutiles, n'appartenant qu'à deux pans coplanaires. En fait, je pense que l'algorithme fonctionne aussi pour des solides contenant des arêtes inutiles, mais je ne connais pas de démonstration simple (sans récursivité et sans parcours de graphe en profondeur) de ce résultat.

4.2 UTILITE DES PANS, PROPAGATION.

A ce stade, toutes les intersections entre les faces de A et les faces de B sont connues; les arêtes d'intersection ordinaires sont connues, ainsi que les sous-arêtes homogènes de chaque arête de A et de B; pour chaque sous-arête, est connue l'éventuelle arête ou face de l'autre solide qui la contient.

Ces structures de données sont fort commodes pour consigner au fur et à mesure les résultats des comparaisons entre les faces; c'est d'ailleurs pour cette raison qu'elles ont été choisies; mais leur hétérogénéité (arêtes initiales des solides / arêtes d'intersection / sous-arêtes) rendrait pénible l'étape finale, qui nécessite de plus quelques pointeurs inverses. La dernière étape a essentiellement besoin des structures de données suivantes :

- une liste de tous les pans; ils se déduisent des arêtes d'intersection et des sous-arêtes des arêtes de A et B; les pans confondus de A et B ne doivent plus être représentés qu'une seule fois. La notion de face devient presqu'inutile ici : ne subsiste plus d'elles que l'attribut (a, b, c,

d) des pans. Le lecteur comprend maintenant la raison d'être des champs (a, b, c, d) et ls dans les enregistrements de type pan. Un champ supplémentaire : "utile ?" est ajouté à chaque pan; il est initialisé à "inconnu";

- une arête est maintenant une donnée du type : (p, q: pointeurs sur les extrêmités inférieure et supérieure; lp : liste des adresses des pans pointant sur cette arête); comme précédemment, une même arête n'a qu'une seule occurence dans la structure de données;

- il devient nécessaire qu'un point (x y z t), sommet ou point d'intersection, soit représenté physiquement une seule fois dans la structure de données. Lui est attachée une liste des adresses des arêtes incidentes.

La génération de ce type de structure a été exposée en 3.2.2.

L'étape finale détermine ensuite l'utilité de chaque pan dans l'enveloppe du solide A∩B. L'utilité d'un pan peut se propager de proche en proche, ce qui limite considérablement les calculs nécessaires; cette idée n'est pas nouvelle, elle est exploitée, entre autres, par LAIDLAW et HUGHES [LAID 86].

L'algorithme procède ainsi : tant qu'il existe des pans dont l'utilité reste inconnue, l'utilité d'un des pans est déterminée; ce pan pourrait être quelconque, mais pour des raisons d'efficacité évidentes, il vaut mieux s'occuper d'abord des pans dont l'utilité (ou l'inutilité) est la plus flagrante : souvent l'utilité peut être déterminée par un test local et rapide. Puis l'utilité de ce pan est propagée selon les règles du 4.2.3.

Dans un premier temps, la méthode laisse parfois subsister les pans de quelques arêtes inutiles (comme celles des figures 3, 4, 5) c'est à dire que leurs pans sont trouvés utiles. Ils ne seront éliminés qu'à la fin. Cette solution n'est peut être pas la meilleure, mais c'est celle qui s'expose le plus simplement.

Quelques définitions deviennent nécessaires :

4.2.1 Définitions

Pans voisins : soit p un pan, d'extrêmités m et n, et de face f. Le pan voisin de p au sommet m est un pan q ayant aussi le sommet m pour extrêmité, de même face et de même solide que p; de plus, au point m, la face f doit se trouver "entre" les pans p et q, et aucun autre pan ne doit se trouver entre p et q; cette dernière condition définit q sans aucune ambiguîté. Remarquons que si q est le pan voisin de p au sommet m, alors p est aussi le pan voisin de q au sommet m : la relation est symétrique.

Les structures de données permettent de déterminer le pan voisin d'un autre pan p en un point donné : le sommet permet d'accéder à la liste des arêtes incidentes; ces arêtes permettent d'accéder aux pans incidents. On ne retient que ceux qui sont de même face que p, de même solide (ils peuvent appartenir aussi à l'autre solide), et différents de p; quand plusieurs pans sont candidats, un test géométrique les départage : il suffit de savoir ordonner angulairement les arêtes coplanaires incidentes en un même sommet (problème déjà rencontré au chapitre I).

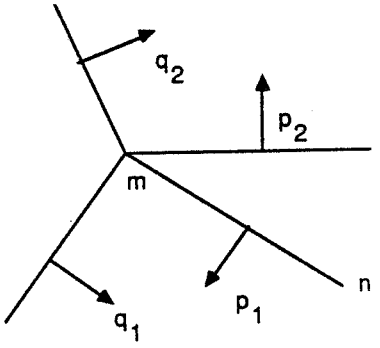


Fig. 18 : Pans voisins

Pans opposés : soit a, une arête d'intersection ordinaire : quatre pans sont donc incidents en a : deux pans de face f et de polyèdre A, et deux pans de face g et de polyèdre B. Ces deux pans de face f sont dits "opposés". De même pour les deux pans de face g.

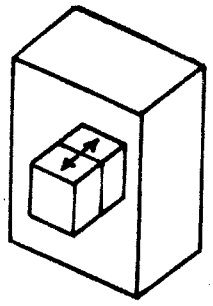


Fig. 19 : Pans opposés

Pans dos à dos : deux pans sont dits dos à dos quand ils ont la même arête, qu'ils sont coplanaires et avec des vecteurs normaux de direction contraire, et qu'ils sont du même coté de l'arête (les sens des deux pans sont donc opposés). En conséquence, les deux vecteurs indicateurs sont colinéaires et de même sens. Intuitivement, des pans dos à dos sont créés lorsque deux polyèdres sont en partie accolés, par exemple quand deux cubes sont empilés.

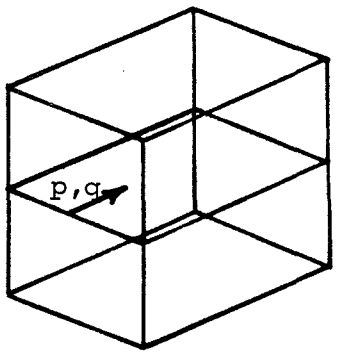


Fig. 20 : Pans dos à dos

Arête banale : une arête dont tous les pans incidents appartiennent au même solide, et rien qu'à lui, est dite "banale" (ne pas confondre avec les arêtes d'intersection ordinaire). Avec les structures de données, la reconnaissance d'une arête banale est triviale.

Sommet banal : un sommet est banal quand toutes ses arêtes incidentes sont banales, et de même solide.

Arête non banale : une arête qui n'est pas banale : les pans incidents référencent les deux solides. Bien sûr, il n'existe pas d'autres arêtes que les arêtes banales et les non banales.

Arête d'intersection ordinaire: il s'agit d'une arête non banale où exactement quatre pans sont incidents; il y a deux pans opposés du solide A (et rien que A), et deux autres pans opposés du solide B (et rien que B); le plan des deux pans de solide A diffère du plan des deux pans de solide B. La notion d'arête d'intersection ordinaire n'est pas vraiment indispensable, mais l'utilité des pans incidents en une arête d'intersection ordinaire peut se déterminer plus efficacement que pour les autres arêtes non banales.

4.2.2 Calcul de l'utilité d'un pan

Comment déterminer l'utilité d'un pan, avant de la propager ? Les méthodes pour ce faire sont maintenant explicitées, par ordre d'efficacité décroissante; bien sûr, il faut d'abord étudier les pans dont l'utilité se calcule le plus rapidement; la propagation des résultats minimise le nombre d'appels à la toute dernière méthode, la plus onéreuse.

1. Un pan appartenant aux deux solides A et B est forcément utile; aucun calcul n'est nécessaire.

2. Deux pans dos à dos sont tous deux inutiles; aucun calcul n'est nécessaire.

3. En une arête d'intersection ordinaire, un test local suffit pour déterminer une utilité. Soit nf le vecteur normal d'une des faces; et soit P un des deux pans de l'autre face. On a vu en 2.2 comment calculer le vecteur indicateur d'un pan P. Le pan P est utile si et seulement si l'angle entre le vecteur indicateur de P et le vecteur nf est supérieur à 90 degrés; le signe du cosinus de cet angle doit donc être calculé. (Ce cosinus ne peut être nul : l'intersection ne serait pas ordinaire; un produit scalaire en arithmétique rationnelle est suffisant)

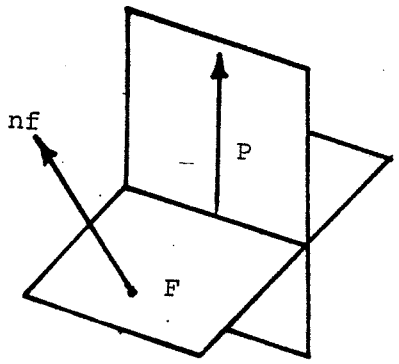


Fig. 20 : Utilité d'un pan d'une arête d'intersection ordinaire : ici, p est inutile.

4. Comment déterminer l'utilité d'un pan en une arête non banale ? Nous supposerons que cette arête n'est pas une arête d'intersection ordinaire, cas plus simple et déjà traité. Des pans des deux solides sont incidents en cette arête. Considérons une coupe de l'arête et de ses pans. Pour fixer les idées, la coupe est faite dans un plan perpendiculaire à l'arête, et l'oeil se trouve du coté du point inférieur de l'arête. Dans cette coupe, l'arête devient un point, les pans deviennent des demi-droites portées par les vecteurs indicateurs, et les solides A et B deviennent des secteurs angulaires entre les pans.

L'idée consiste à effectuer l'opération booléenne d'intersection sur ces secteurs angulaires. En effet, l'intersection de la coupe de A et de la coupe de B est la coupe de l'intersection de A et de B. Les pans incidents en a seront donc utiles si et seulement si leur vecteur indicateur (leur coupe) est utile dans la coupe. Encore faut-il savoir déterminer l'intersection de deux suites d'intervalles angulaires. Ce problème est analogue à celui du calcul de l'intersection de deux suites d'intervalles le long d'une droite, déjà exposé en 4.1.1.

5. Comment déterminer l'utilité d'un pan en une arête banale ? Ce test est fort lent dans le cas général, et ne doit être effectué que s'il est vraiment impossible de faire autrement. Il existe cependant un cas où la réponse est rapide. Les pans incidents de l'arête banale a sont tous soit des pans de A, soit des pans de B, par définition des arêtes banales. Pour fixer les idées, supposons que a soit une arête de A. Si l'un des sommets de a se trouve hors de la boîte englobante des sommets de

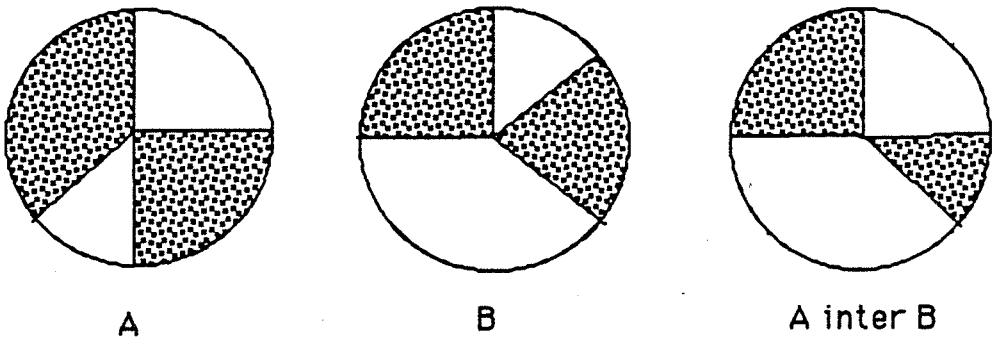


Fig. 21 : Utilité d'un pan en une arête non banale.

B, alors l'utilité (ou l'inutilité) des pans de a est évidente : a n'est utile que si B est un solide infini.

Si les deux sommets de l'arête a se trouvent dans la boîte englobante de B, l'utilité du pan ne peut être déterminé par un test local : il faut déterminer pour un point quelconque de a (par exemple le milieu de l'arête) s'il appartient ou non au solide B. Le test d'appartenance d'un point à un polyèdre est un problème classique. La solution la plus simple exploite le principe de parité; un rayon est lancé à partir du point, et la parité du nombre de points d'intersection entre le rayon et les faces du solide est calculée. Pour un solide borné, le point est intérieur quand le nombre des points d'intersection est impair. Pour un solide infini (pour nous, le complémentaire d'un solide fini), le point est intérieur quand le nombre des points d'intersection est pair. Dans cette méthode, il est nécessaire de savoir reconnaître si un point, intersection entre le rayon et le plan d'une face, se trouve ou non à l'intérieur de ladite face; ce problème peut se résoudre là aussi en lançant un rayon dans le plan de la face, et en réutilisant le principe de la parité. Les cas particuliers peuvent être éludés : le vecteur directeur du rayon est choisi de façon aléatoire; si malgré tout, par une malchance insigne, ce rayon rencontre un sommet ou une arête, il est plus simple de lancer un autre rayon (et de recommencer le décompte des points d'intersection bien sûr) que de prendre en compte les cas particuliers. L'utilisation des boîtes englobantes est une façon simple d'améliorer les performances de cette méthode; sa relative inefficacité ne porte pas à conséquence, car elle est appelée peu souvent. LAIDLAW et HUGHES utilisent aussi cette technique (mais avec une arithmétique flottante).

On a donné des méthodes déterminant l'utilité des pans incidents à tous les types d'arêtes possibles. Remarquons que les méthodes 4 et 5 seraient suffisantes, mais les méthodes 1 à 3 sont plus efficaces.

4.2.3 Règles de propagation

Trois règles évidentes permettent de propager l'utilité des pans dans la description du solide final :

Tous les pans incidents à une arête banale ont des utilités égales.

Deux pans opposés ont des utilités opposées.

Deux pans voisins ont des utilités égales.

Ces règles ne souffrent aucune exception.

D'autres règles pourraient être formulées, comme par exemple : tous les pans incidents en un sommet banal ont des utilités égales. Mais elles ne sont que des conséquences de l'application répétée des trois règles de propagation; elles sont donc inutiles. La procédure de propagation ci-dessous exploite ces trois règles de façon très classique; il s'agit en fait du parcours en profondeur d'un graphe virtuel, qui a pour sommets les pans, et pour arcs les relations : "pan voisin de", "pan opposé à", "pan incident à la même arête banale". L'utilité d'un pan se propage à toute la composante connexe de ce pan, pour ce graphe.

```
empiler-si ( Q : pan; U:  utilité )=   /* l'utilité U est connue */
{
 si (le champ "utile ?" de Q vaut : inconnue)
 alors {  /* Q n'a jamais été empilé */
         champ "utile ?" de Q  ← U;
         empiler(Q, U);
       }
 sinon { /* Q est ou a déjà été empilé */ }
 /* un pan n'est empilé qu'une seule fois par la procédure calculant A∩B */
}
```

```
propager ( P : pan; U : utilité )=
{
 empiler-si (P,U);
 tant que pile non vide faire
      {
       dépiler (P,U);
       empiler-si ( pan-voisin ( P, sommet inférieur de P ) ,U);
       empiler-si ( pan-voisin ( P, sommet supérieur de P ) ,U);
       a ← arête de P;
       si ( a est une arête banale )
       alors pour (chaque pan Q incident en a et différent de P)
            empiler-si(Q,U);
       si ( a est une arête d'intersection ordinaire )
       alors empiler-si ( pan-opposé(P), non(U));
       /* non(utile) = inutile; non(inutile) = utile */
      }
}
```

Quand l'utilité de tous les pans est connue, le problème est presque résolu : éliminer les pans marqués "inutiles" ne suffit pourtant pas tout à fait. Il subsiste quelques arêtes doubles facultatives, qui se reconnaissent facilement : exactement deux de leurs pans incidents sont marqués "utiles", et les champs (a, b, c, d) de ces deux pans sont égaux; ces arêtes doubles facultatives, et leurs pans, doivent être supprimés. Enfin, il est bienvenu de simplifier la description du solide A∩B en éliminant certains sommets, inutiles : ces sommets sont de degré deux, et les deux arêtes incidentes sont alignées; ces deux arêtes, et les pans incidents, sont fusionnés.

Alors, la liste des pans restants décrit la frontière du solide A∩B.

4.3 LE CAS PARTICULIER DES SOMMETS ISOLES

Pour plus de lisibilité, l'exposé précédent ne prenait pas en compte le cas particulier des sommets situés à l'intérieur d'une face. Ces sommets sont dits "isolés".. Une erreur peut survenir dans le cas où soit une arête d'intersection, soit une arête de l'autre solide, passe juste par un sommet isolé (ce cas est bien sûr extrêmement rare) : alors, une arête d'intersection, ou une sous-arête, passe par un sommet sans qu'on le sache : il y a création d'un contact implicite, ce qui n'est pas prévu par l'algorithme.

Le reméde est évident : quand une arête [i,j] contient un sommet isolé s, il faut la couper en deux arêtes [i,s] et [s,j]. Une variante possible est maintenant détaillée :

Supposons que pour chaque face f soit connue la liste L[f] (vide le plus souvent) des sommets isolés à l'intérieur de la face f. Alors, chaque fois qu'est calculée l'intersection de deux faces de plans sécants le long d'une droite D, et qu'un de ces points s appartient à D, l'algorithme doit vérifier pour chaque arête

d'intersection créée, et pour chaque sous-arête incluse dans D, si, par hasard, elle ne contiendrait pas s. Si tel est le cas, l'arête d'intersection ou la sous-arête doit être coupée en deux par s. Ainsi, il ne se crée aucun contact implicite et l'algorithme fonctionne correctement.

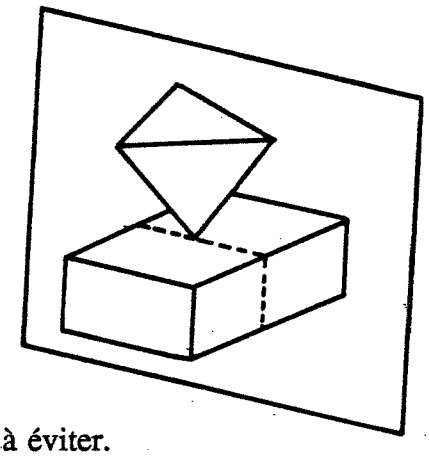


Fig. 22 : Le cas particulier : l'erreur à éviter.

Comment tenir à jour ces listes de points isolés ? Détecter qu'un sommet de A se trouve strictement à l'intérieur d'une face f de B (et réciproquement) se fait de façon simple lors du calcul de l'intersection des faces de A et de B : toutes les données nécessaires sont immédiatement disponibles dans les structures de données, et la mise à jour de L[f] va de soi.

Quand un point p strictement intérieur à une face f doit-il être retiré de sa liste L[f] ?

Remarquons que si aucun de ces points isolés ne doit être oublié, par contre les listes L peuvent fort bien contenir des points en excédent, par exemple des points devenus inutiles : cela ne perturbe en rien le fonctionnement de l'algorithme. Simplement, quelques arêtes seront parfois inutilement coupées, avec beaucoup de malchance : ces sommets inutiles seront de toutes façon détectés et supprimés à la fin de l'algorithme. La suppression des points isolés devenus inutiles dans les listes L n'est donc même pas indispensable.

Pour des raisons d'esthétique, on peut préférer supprimer les sommets isolés devenus inutiles. Quand un sommet isolé s de L[f] devient-il inutile ? Manifestement, quand plus aucune arête du solide résultat, $A \cap B$, ne passe par s; ou quand f n'appartient plus à $A \cap B$; ou quand une arête de f a s pour extrêmité, car alors s n'est plus strictement intérieur à f; ou enfin quand s ne se trouve plus à l'intérieur de la face f : il suffit d'appeler la fonction testant l'inclusion d'un point dans une face, qui est nécessaire en 4.2.2. Dans tous les autres cas, la présence de s dans L[f] reste nécessaire.

En conclusion, la prise en compte du cas particulier des sommets isolés implique seulement quelques ajouts mineurs. Vu la très faible probabilité de ce cas, des jeux de données doivent être générés spécialement pour tester cette partie du code : c'est là le principal inconvénient de ce cas particulier.

Peut-être existe-t-il des méthodes plus efficaces de régler ce cas particulier. La grande rareté de ce cas autorise de privilégier la simplicité au dépend de l'efficacité.

5 COMPLEMENTAIRE D'UN POLYEDRE

Le complémentaire d'un solide est obtenu en inversant les équations abcd des faces, le sens des pans du polyèdre, et sa variable booléenne "fini ?". (Moyennant une indirection, il est possible d'effectuer ces opérations en O(1))

Toutes les opérations booléennes s'obtiennent ensuite par simple composition de l'intersection et du complémentaire.

6 OPTIMISATION

6.1 TECHNIQUES UTILISEES EN CAO ET EN SYNTHESE D'IMAGES

Comment déterminer rapidement une liste correcte des faces de A et de B susceptibles de s'intersecter ? Cette liste est correcte si et seulement si y figurent au moins tous les couples de faces intersectantes.

Cette section présente quelques méthodes classiques en synthèse d'images ou en "solid modeling". La section suivante étudie l'apport de la géométrie algorithmique pour le même problème.

6.1.1 Les boîtes englobantes

L'idée la plus classique est celle des boîtes englobantes; la boîte englobante d'une face est le plus petit parallèlépipède [xmin, xmax] x [ymin, ymax] x [zmin, zmax] qui contienne tous les sommets de la face; sa détermination est triviale, de même que le test d'intersection entre deux boîtes. Or deux faces ne peuvent s'intersecter que si leurs boîtes englobantes s'intersectent. La boîte englobante d'un solide est définie de façon similaire : c'est le plus petit parallèlépipède [xmin, xmax] x [ymin, ymax] x [zmin, zmax] qui contienne tous les sommets du solide. Remarquons que le solide n'est pas forcément inclus dans cette boîte englobante : par exemple dans le cas du complémentaire d'un cube.

Le pseudocode suivant utilise classiquement les boîtes englobantes, comme par exemple LAIDLAW et HUGHES [LAID 86] :

```
liste des faces susceptibles de s'intersecter (A, B) =
{ L ← ∅;
  si boîte(A) ∩ boîte(B) ≠ ∅
  alors pour chaque face f de A
        {  si boîte(f) ∩ boîte(B) ≠ ∅
           alors pour chaque face g de F
                 si boîte(f) ∩ boîte(g) ≠ ∅
                 alors L ← L ∪ (f, g);
        }
  rendre(L);
}
```

D'après les nombreux constats empiriques, ces tests sur les boîtes englobantes améliorent très nettement les performances des programmes; cependant, d'un point de vue théorique, O(|A| |B|) tests sont toujours effectués.

6.1.2 La partition spatiale

Le deuxième paradigme utilisé est celui de la partition spatiale : par exemple, MANTYLA [MANT 83] insère les boîtes englobantes des faces de A et de B dans une variante de bintree. Dans une première définition, le bintree découpe l'espace jusqu'à ce que les voxels contiennnent moins de k boîtes, de façon à séparer les boîtes disjointes; mais alors l'algorithme d'insertion s'enfonce à l'infini dans l'arbre quand plus de k boîtes sont réellement superposées dans l'espace ! Aussi MANTYLA modifie-t-il quelque peu la définition du bintree. Quand toutes les boîtes ont été insérées dans l'arbre, un parcours des voxels donne toutes les faces suspectes d'intersection : leurs boîtes occupent un même voxel. MANTYLA a effectué de nombreux tests, qui prouvent l'efficacité pratique de cette technique. L'analyse théorique, elle, n'est pas simple; le bintree peut cependant être vu comme un mauvais kd-tree [BENT 75] , car non équilibré :

Rappelons rapidement les performances théoriques du kd-tree [LEE 77] : pour une dimension $d \geq 2$, la recherche dans un kd-tree des K points (parmi N) contenus dans un hyper-rectangle de dimension d requis est en $O(K + d\ N^{1-1/d})$, ce qui est un peu décevant. [ALT 81] démontre qu'une structure plus efficace que le kd-tree doit forcément être plus encombrante; le kd-tree occupe une place mémoire en O(N), manifestement optimale.

6.1.3 Le balayage

Un troisième technique envisageable est le balayage. Deux faces ne peuvent s'intersecter que si elles s'intersectent en projection sur un plan. Les contours superposés dans un plan peuvent être déterminés par l'algorithme de BENTLEY-OTTMAN. Cette méthode est utilisée dans [SZIL 84]. Mais le constat empirique effectué en II.8 peut peut-être faire douter de l'intérêt du balayage.

Ces techniques satisfont souvent les besoins pratiques de la synthèse d'images ou du "*solid modeling*", mais ne sont pas vraiment satisfaisantes d'un point de vue théorique. La section suivante présente une méthode qui détermine efficacement les K couples ($a \in A$, $b \in B$) de boîtes intersectantes. Cette méthode est directement inspirée des techniques du "*Multi-Dimensionnal Searching*" de la géométrie algorithmique; curieusement, ces dernières ont été très peu utilisées en CAO et en synthèse d'images (je ne connais pas d'exemple d'utilisation).

6.2 LES TECHNIQUES DU "MULTI-DIMENSIONNAL SEARCHING"

6.2.1 Un algorithme efficace

Les techniques du "*Multi-Dimensionnal Searching*" sont le prolongement naturel de l'algorithmique maintenant classique [KNUTH 81] des tris et des recherches dans un ensemble d'enregistrements ordonnés sur une clé unique, ensemble qui peut être vu comme un espace à une seule dimension. Ces techniques ont été élaborées dans de très nombreux articles, mais une synthèse exaustive de l'état de l'art est effectuée dans [PREP 85] et [MELH 84] : y sont notamment présentées les structures de données comme le "*segment-tree*", le "*range-tree*", l'"*interval-tree*", le "*priority-search-tree*", et comment ils permettent de résoudre les six problèmes suivants dans R^d. Le sixième est proche du nôtre, et sa solution fait appel à la solution des cinq problèmes précédents :

"***Static Range Searching Problem***" : dans un ensemble donné de N points de R^d, trouver les K points contenus dans un d-rectangle requis (un d-rectangle est en dimension d l'analogue en 2D d'un rectangle parallèle aux axes). De nombreuses requêtes sont effectuées; y répondre rapidement justifie un prétraitement qui organise l'ensemble des N points dans une structure de "*range-tree*", ou de "*layered-range-tree*". La construction du "*range-tree*" et sa place mémoire sont en $O(N\log^{d-1}N)$; la recherche des K points satisfaisant une requête s'effectue en $O(K+\log^{d}N)$. Le "*layered-range-tree*" a les mêmes performances, sauf la recherche qui s'effectue en $O(K+\log^{d-1}N)$. La recherche triviale, par contre, n'exige aucun prétraitement à part la lecture des données en O(N), et satisfait une requête en O(N) temps et espace.

"***Static Interval Searching Problem***" : dans un ensemble donné de N d-rectangles, trouver les K d-rectangles contenant un point requis. De nombreuses requêtes sont effectuées, ce qui justifie un prétraitement qui organise l'ensemble des N d-rectangles dans un arbre récurrent à base de "*segment-tree*" et de "*range-tree*". Le prétraitement et l'espace mémoire sont en $O(N\log^d N)$, la recherche en $O(K+\log^d N)$. La recherche naïve satisferait cette requête en O(N) temps et espace.

"***Dynamic Range Searching Problem***" et "***Dynamic Interval Searching Problem***" : la version dynamique des deux problèmes précédents; l'ensemble des points (ou des d-rectangles) subit des insertions et des suppressions. Cependant, chacune des coordonnées des points (ou des d-rectangles) ne peut prendre que N valeurs possibles, préalablement fixées. Ainsi le "squelette" des structures précédentes n'est pas modifié, et permet de résoudre la variante dynamique des problèmes.

"***Static Searching Problem***": dans un ensemble donné de N d-rectangles, trouver les K d-rectangles intersectant un d-rectangle requis. Ce problème est réduit au "*Static Range Searching Problem*" et au "*Static Interval Searching Problem*". Le prétraitement est en $O(N \log^d N)$, l'espace occupé par la structure de données en $O(N \log^{d-1} N)$, et la recherche en $O(K+\log^d N)$. L'algorithme naïf est en O(N) temps et espace.

"***Dynamic Searching Problem***" : la version dynamique du problème précédent, résolu par l'emploi de la variante dynamique des structures précédentes; la recherche est en $O(K+\log^d N)$, l'insertion et la suppression en $O(\log^d N)$, l'espace mémoire en $O(N \log^{d-1} N)$.

"***All Pair Intersection Problem***" ou le problème des paires; soit un ensemble A de N d-rectangles. Trouver les K paires ($a \in A$, $b \in A$) telles que $a \cap b \neq \emptyset$. Ce problème est très proche du nôtre, et sa solution est exposée pour le cas familier d=3 (la généralisation est triviale et sans intérêt pour nous).

R^3 est balayé de bas en haut par un plan horizontal P; l'intersection de deux 3-rectangles est détectée une fois et une seule lorsque le plus haut commence à intersecter P. Les d-rectangles intersectant P sont dits actifs. L'ensemble des d-rectangles actifs est représenté par une structure S résolvant le "*Dynamic Searching Problem*" en dimension 2.

```
problème des paires =
{
 ordonner les zmin et zmax des 3-rectangles;
 S ← ∅ ;
 pour chaque z en ordre croissant faire
  { selon que z est le
     • zmin d'un 3-rectangle a :
       { chercher dans S les 3-rectangles intersectant a;
         S ← S ∪ a ; }
     • zmax d'un 3-rectangle a : S ← S - a ;
  }
}
```

Le tri initial est en O(NlogN) temps. Les N insertions et suppressions dans S se font chacune en $O(\log^2 N)$; la recherche dans S des k 2-rectangles intersectant un 3-rectangle naissant est en $O(k+\log^2 N)$. Le problème est donc résolu en $O(K+N\log^2 N)$; la structure S occupe O(NlogN) espace. L'algorithme naïf exige $O(N^2)$ temps et O(N) espace.

Notre problème est le "**problème des couples**" : soit A et B deux ensembles statiques de 3-rectangles; trouver tous les couples ($a \in A$, $b \in B$) tels que $a \cap b \neq \emptyset$. A ma connaissance, il n'est traité par aucun article de "*Multi-Dimensionnal Searching*", mais il se résoud trivialement par un dédoublement des structures de données employées pour résoudre le problème des paires :

L'espace est balayé de bas en haut par un plan horizontal P. L'intersection de A et de P est décrite par la même structure de données que pour le problème des paires; soit S_A cette structure. De même S_B décrit l'intersection de P et de B. Le pseudocode suivant décrit l'algorithme :

```
Problème des couples (A , B ) =
{ ordonner les zmin et zmax de A et B;
  S_A ← ∅; S_B ← ∅;
  pour chaque z par ordre croissant faire
  { selon que z est le
     • zmin de a∈A :
        { chercher les éléments b∈B intersectant a dans S_B ;
          S_A ← S_A ∪ a ; }
     • zmax de a∈A : S_A ← S_A - a ;
     • zmin de b∈B :
        { chercher les éléments a∈A intersectant b dans S_A ;
          S_B ← S_B ∪ b ; }
     • zmax de b∈B : S_B ← S_B - b ;
  }
}
```

Soit N = |A|+|B|; l'algorithme détermine les K couples intersectants en $O(K+N\log^2 N)$ temps et O(NlogN) espace. L'espace mémoire reste quasi linéaire, et la recherche des couples intersectants est bien plus performante que par la méthode naïve.

Une remarque qui va de soi : les requêtes multi-dimensionnelles ne sont pas vraiment des problèmes géométriques, numériques : seul compte l'ordre des nombres, et aucun calcul réel ou flottant (à part les comparaisons < > =) n'est effectué. Les algorithmes du "*Multi-Dimensionnal Searching*" ne sont donc pas perturbés par l'imprécision numérique, bien que les données puissent souffrir du recalage sur l'ensemble des flottants. De toutes façons, dans notre cas, les coordonnées sont des nombres rationnels.

6.2.2 Le "Static Range Searching Problem" revisité

A propos des algorithmes satisfaisant les requêtes du "*Static Range Searching Problem*", [PREP 85 pg71] s'interroge sur l'écart entre la borne inférieure théorique connue : K+dlogN (d>1) et les performances du meilleur algorithme connu, qui est en $O(K+N\log^{d-1}N)$ temps (on rappelle en passant que cet algorithme utilise un "*layered-range-tree*", dont la construction exige $O(N\log^{d-1}N)$ temps et espace).

Rappelons briévement la démonstration de la borne inférieure théorique : [PREP 85] suppose que les N sommets se trouvent sur les axes, N/d sommets par axe, symétriquement par rapport à l'origine. Avec cette configuration, il existe de l'ordre de $(N/2/d)^{2d}$ réponses distinctes possibles à une requête. Un ordinateur idéalisé, capable de diviser en deux l'espace de recherche à chaque top d'horloge, a besoin de T unités de temps pour trouver la bonne réponse, avec $2^T \geq (N/2/d)^{2d}$. Avec cette configuration, la réponse à une requête ne peut donc être trouvée en moins de d logN unités de temps, et K unités de temps supplémentaires permettent d'énumérer la solution.

La configuration est-elle trop simpliste, ou le meilleur algorithme connu n'est-il pas optimal ? [PREP 85 pg71] s'interroge sans pouvoir conclure. On rappelle que dans le cas voisin du "*Dynamic Range Searching Problem*", FREDMAN [FRED 81] a prouvé la borne inférieure théorique de $K+\log^d N$, qui correspond aux performances des meilleurs algorithmes connus.

Le paragraphe suivant prouve que, dans le cas statique, la borne inférieure théorique de K+dlogN est bien correcte, autrement dit les algorithmes de recherche connus ne sont pas optimaux. Un algorithme de recherche en O(K+dlogN) est exhibé en exclusivité :

Un prétraitement détermine toutes les réponses possibles aux requêtes; il peut sembler que le nombre de d-rectangles $[min_1, max_1] \times [min_2, max_2] \ldots \times [min_d, max_d]$ soit infini, mais en fait seul compte le rang de min_1 et max_1 dans l'ensemble des N abscisses, le rang de min_2 et max_2 dans l'ensemble des N ordonnées, etc. Ainsi, les d-rectangles requis peuvent toujours être "recalés" sur les coordonnées des N sommets; par exemple, si $x_j < min_1 \leq x_k$, et si aucun sommet n'a une abscisse incluse dans $]x_j, x_k[$, alors min_1 peut être recalé sans danger sur x_k; si $x_p \leq max_1 < x_q$, et si aucun sommet n'a une abscisse incluse dans $]x_p, x_q[$, alors max_1 peut être recalé sans danger sur x_p.

Un prétraitement calcule donc les réponses pour tous les d-rectangles recalés possibles. Appelons rang de l'abscisse d'un sommet le rang de cette abscisse dans l'ensemble ordonné des N abscisses; appelons rang de l'ordonnée d'un sommet le rang de cette ordonnée dans l'ensemble ordonné des N ordonnées; et ainsi de suite pour les d coordonnées des sommets; les d rangs sont accrochés aux sommets lors du prétraitement. Avec ces notations, la réponse pour le d-rectangle recalé $[rmin_1, rmax_1] \times [rmin_2, rmax_2] \ldots \times [rmin_d, rmax_d]$, est stockée dans REP $[rang(rmin_1)]$ $[rang(rmax_1)] \ldots [rang(rmin_d)]$ $[rang(max_d)]$; REP est un tableau à 2d dimensions de listes des sommets solutions.

Après ce prétraitement, la recherche des sommets solutions pour une requête s'effectue trivialement : le d-rectangle [min, max] requis doit être recalé en [rmin, rmax], ce qui s'effectue bien en O(d logN). Ensuite, un simple accès dans le tableau $REP[rang(rmin_1)] \ldots [rang(rmax_d)]$ donne la tête de liste de la réponse (l'accés à la tête de liste exige O(d) opérations). Enfin, l'énumération des sommets solutions se fait en O(K), K étant le nombre de solutions.

Bien sûr, cet algorithme est totalement inutilisable : l'espace mémoire utilisé est en $O(N^{2d})$ et le prétraitement ne peut évidemment être meilleur (puisqu'il doit initialiser cet espace mémoire). C'est vraisemblablement de par sa colossale stupidité que cette méthode a pu échapper à la sagacité de SHAMOS, PREPARATA et de leurs prédécesseurs; pourtant, [PREP 85 pg77-80] envisagent des méthodes similaires, utilisant des précalculs brutaux.

L'existence de cet algorithme prouve cependant que : en dimension d, la recherche des sommets inclus dans un d-rectangle requis, pour un ensemble statique de N points, peut s'effectuer de façon optimale en O(d logN); l'espace mémoire est en $O(N^{2d})$.

Il me parait inutile de préciser davantage l'ordre de grandeur du prétraitement, manifestement exponentiel. Par contre, je ne peux résister au plaisir d'optimiser encore l'algorithme de recherche, ...qui est pourtant optimal. Le paragraphe suivant propose une méthode de recherche plus qu'optimale, en O(d) :

Le paradoxe n'est qu'apparent : la borne inférieure théorique : d logN, suppose la continuité de l'espace de recherche, R^d. Mais les machines réelles, finies, n'utilisent que des approximations des nombres réels : les flottants; un flottant est représenté par un nombre fini de bits, par exemple 32 bits; il n'y a donc que 2^{32} flottants distincts, ce qui est bien loin de la puissance du dénombrable, sans parler de celle du continu. Il est donc possible, pour chacune des d dimensions, de stocker dans deux tables RMIN[] et RMAX[] de 2^{32} entrées les valeurs recalées rmin et rmax pour les 2^{32} flottants possibles. Envisageons, par exemple, le cas d=1, et l'intervalle flottant [g,d]; la représentation binaire de g (resp. d) est aussi la représentation binaire d'un entier g' (resp. d'). L'intervalle recalé sur les N abscisses est [RMIN[g'], RMAX[d']]; il se détermine manifestement en O(1) après la construction des tables, et non plus en logN. En d dimensions, il existe donc une méthode de recherche en O(d), qui exploite la nature foncièrement discrète des machines réelles. Peut-on encore améliorer l'algorithme ? Non : le d-rectangle est décrit par 2d valeurs, que doit bien lire la procédure de recherche, et qui servent en quelque sorte d'indices dans une table gigantesque contenant toutes les solutions.

Ces algorithmes de recherche, bien qu'optimaux, doivent manifestement être écartés; mais ni [PREP 85], ni [MELH 84] (pour prendre les deux livres de référence) n'explicitent clairement les critères théoriques justifiant cette attitude. A leur décharge, ils n'ont pas envisagé ces deux algorithmes.

7 CONCLUSION

Un algorithme et une structure de données simples et généraux ont été proposés pour effectuer les opérations booléennes sur des polyèdres rationnels. La BREP employée conserve de façon implicite toutes les informations contenues dans des structures plus complexes; ces informations implicites peuvent être regénérées facilement.

Cette nouvelle approche n'a cependant pas que des avantages : ainsi une arithmétique rationnelle est indispensable, et son coût est loin d'être négligeable. De plus, cette solution exploite le caractère affine des polyèdres et ne peut donc pas être généralisée à des solides dont les surfaces d'enveloppe ont un degré supérieur.

Chapitre 4

LA REPRESENTATION EXACTE DES FRONTIERES DES SOLIDES ALGEBRIQUES

1 INTRODUCTION

Les auteurs du "*solid modeling*" semblent penser que la modélisation exacte des solides plus complexes que les quadriques (ellipsoïdes, cônes, cylindres...) est impossible. Citons par exemple MANTYLA ([MANT 83]) :

"*Exact intersections of other types of surfaces do not seem to be possible, howewer. To manipulate objects with parametric surfaces, faceting seems to be necessary ([CARL 82])*"

A part les polyèdres, et les quadriques dont la modélisation est étudiée dans [LEVI 79], [LEVI 80] et [MILL 86], le "*solid modeling*" a jusqu'ici utilisé surtout les courbes et les surfaces paramétriques : les carreaux de surfaces bicubiques ou les Bsplines rationnelles par exemple ([BEZI 86], [CAST 85]). Les avantages des formes paramétrées sont bien connus, et présentés par exemple dans [ROSS 86] ou en II.1. Mais la modélisation par des formes paramétriques a aussi quelques inconvénients : ces formes ne décrivent pas spontanément un solide, au contraire des formes implicites $f(x,y,z)<0$; enfin, dans le cas général, l'intersection de deux surfaces paramétriques n'est pas une courbe paramétrique, ce qui oblige ce type d'approche à recourir à des représentations approximatives, dont [CASA 87] et [CARL 82] sont des exemples typiques.

Ce chapitre veut montrer que la modélisation exacte des "solides algébriques" est possible, grâce à l'emploi des formes implicites des surfaces algébriques.

On rappelle que l'implicitisation des formes paramétriques polynomiales ou rationelles est toujours possible : les techniques nécessaires sont présentées dans [SEDE 84]. L'opération inverse, la paramétrisation rationnelle des formes implicites, est le plus souvent impossible; les coniques et les quadriques sont une des exceptions les plus notables.

Dans ce chapitre, on appelle arbre CSG, ou arbre de construction, un arbre dont les noeuds portent les opérateurs booléens régularisés et dont les feuilles portent des inéquations algébriques $f_i(x,y,z)<0$ de degré fini quelconque; les f_i sont des polynômes multivariés en x, y et z, à coefficients rationnels ou algébriques; ils sont supposées "*squarefree*" (sans diviseurs carrés). $f_i(x,y,z)<0$ définit un solide; dans certains cas particuliers, ce solide peut être vide ou égal à R^3; dans tous les cas, il est régularisé au sens de TILOVE, grâce au caractère strict de l'inéquation : le solide ne contient pas de surface ou de courbe n'adhérant pas à son intérieur ([TILO 80]). C'est l'objet défini par la racine d'un tel arbre qui est appelé "solide algébrique" ou "objet algébrique".

Ce chapitre définit pour les objets algébriques une BREP exacte, qui décrit explicitement les frontières des objets. Puis il expose un algorithme qui produit la BREP d'un objet initialement décrit par un arbre CSG. La méthode proposée utilise un prolongement des cartes planaires du chapitre I, et quelques techniques des chapitres II et III que le lecteur reconnaîtra au passage. Cet algorithme sacrifie délibérément toutes les préoccupations d'efficacité au dépend de la simplicité et de la lisibilité : il est surtout une preuve constructive. Cet algorithme réduit le passage du CSG à la BREP à une liste de problèmes mathématiques qui ont tous été déjà résolus en calcul algébrique symbolique (quoique pas forcément de façon efficace); citons, entre autres, la dérivation symbolique des expressions polynômiales, l'élimination d'une variable dans un système d'équations algébriques, le calcul de résultants, l'intersection entre deux courbes planes algébriques, la détermination d'encadrements rationnels isolant les zéros réels d'un polynôme, le calcul de PGCD entre polynômes, le calcul des séquences de STURM, le calcul du nombre de zéros d'un polynôme dans un encadrement réel par les séquences de STURM, etc; ces problèmes ne sont pas tous indépendants.

Certaines des techniques nécessaires (élimination, résultants, séquences de STURM) ont déjà été utilisées en synthèse d'images, par KAJYIA ([KAJY 82]), WIJK ([WIJK 84]), et surtout par SEDERBERG et al. : [SEDE 84] [SEDE 85] et [SEDE 86] présentent notamment les techniques d'implicitisation des courbes et des surfaces paramétrées, et le calcul des points d'intersection entre courbes algébriques. Mais ils n'appliquent pas encore ces techniques à la modélisation des solides : ce que tente de faire ce chapitre.

♦ **Remarque** Les chapitres précédents ont montré les difficultés que pouvait causer l'imprécision numérique. Pour les éviter, nous supposerons qu'est employée une arithmétique algébrique, qui représente fidèlement tous les nombres algébriques. De telles arithmétiques existent ([LOOS 82] [DICR 86]), bien que jamais employées en "*solid modeling*".

L'approche proposée ici est plus proche du calcul symbolique ([PAVE 85]) que des approches classiques du "*solid modeling*". Le principal inconvénient des techniques du calcul symbolique est la puissance de calcul nécessaire : à titre d'exemple, l'inversion numérique d'une matrice NxN est en $O(N^3)$; l'inversion symbolique est en O(N!).

2 DESCRIPTION DE LA BREP

Les frontières du solide algébrique sont décrites en considérant une projection dans un plan de référence, disons une projection d'axe Oz sur Oxy.

On note SQRF(p) le polynôme "*squarefree*" ayant les mêmes diviseurs que le polynôme p. Des algorithmes déterminant SQRF(p), p donné, sont rappelés et comparés dans [YUN 76] et [YUN 77]. Posons $f(x,y,z) = SQRF(\Pi_i f_i(x,y,z))$: $f(x,y,z)=0$ est la plus petite surface contenant toutes les surfaces $f_i(x,y,z)=0$. La suite considère la projection de $f(x,y,z)=0$ sur Oxy.

Deux notations qui seront utilisées dans la suite de ce chapitre : on note g_x la dérivée relativement à x du polynôme g, et on note $R_x[g,h]$ le résultant de BEZOUT (ou déterminant de SYLVESTER) des deux polynômes g et h considérés comme des polynômes en x; des méthodes de calcul de $R_x[g,h]$ sont rappelées dans [SEDE 84]. La courbe du contour apparent dans Oxy de f (et de la projection sur Oxy des points d'auto-intersection de f) est donnée par : $f(x,y,z) = f_z(x,y,z) = 0$, c'est à dire, sauf exception : $R_z[f, f_z] = g(x,y) = 0$. Dans certains cas particuliers, g est identiquement nul : cela survient avec des surfaces cylindriques (au sens général) d'axe Oz. Ce cas particulier est ignoré : on peut remarquer que f ne peut contenir qu'un nombre fini de surfaces cylindriques, et il existe donc une infinité d'axes de projection non particuliers.

Pour simplifier, l'exposé ne détaille le traitement d'aucun cas particulier; ils peuvent tous être évités moyennant le choix d'un axe de projection "suffisamment quelconque". On suppose donc que chaque point de Oxy n'est la projection que d'un nombre fini de points de la surface $f(x,y,z)=0$: g ne peut être identiquement nul; on suppose aussi que g est "*squarefree*" : géométriquement, un arc de $g(x,y)=0$ est la projection d'un arc unique de contour (ou d'auto-intersection) de la surface $f(x,y,z)=0$. On suppose enfin que g ne contient aucune droite d'axe Ox ou Oy, et que ces droites ne sont pas des asymptotes de g : si tel n'est pas le cas, une simple rotation du repère du plan de projection suffit.

On décrit maintenant la BREP :

La BREP est une liste de faces (le mot "faces" ne signifie pas "faces planes").

Une face est une liste de bouts de face.

Un bout de face est défini par une région du plan Oxy, et par un entier strictement positif appelé : profondeur. On se contentera provisoirement de l'acceptation intuitive du mot "région". La profondeur se définit ainsi :

L'intersection entre un bout de face et une droite d'axe Oz traversant l'intérieur de la région du bout en (x_v, y_v) compte toujours un point et un seul; soit z_v la cote de ce point : z_v est toujours le k-ième zéro réel du polynôme "*squarefree*" $f(x_v,y_v,z) = q(z)$, k étant la profondeur du bout de face, et les zéros étant classés par ordre croissant.

Par exemple, la BREP d'une sphère compte deux bouts de face : l'hémisphère supérieur, de profondeur 2, et l'hémisphère inférieur, de profondeur 1. Les deux bouts de face se projettent sur la même région, délimitée par le cercle équatorial de la sphère.

Toute liste de bouts de face n'est pas forcément une face. Intuitivement, une face est une partie continue et connexe de la surface f=0, dont l'intersection avec toute droite verticale passant par l'une des régions des bouts de la face compte un point et un seul. Le procédé de construction des faces les décrit d'une façon à la fois plus intuitive et plus précise; il est exposé ultérieurement.

Un point M appartient-il strictement à un bout de face ? Il faut et il suffit que M appartienne strictement à la région, et que z_M soit le p-ième zéro de f(x,y,z)=0, p étant la profondeur du bout de face. Quand un point appartient-il strictement à une face ? Cette question se réduit trivialement à la précédente.

Il devient nécessaire de définir les régions délimitées par la courbe g(x,y)=0, et comment elles sont construites.

3 ARCS D'UNE COURBE, REGIONS

La courbe g(x,y)=0 crée une partition de Oxy en régions (le chapitre II emploie aussi le terme de "zones"); informatiquement, une région peut être représentée par la liste de ses arcs frontières. Mais comment définir les arcs ?

Deux points définissent un segment et un seul, et le test d'appartenance d'un point à un segment est trivial. Les choses ne sont plus aussi simples avec des arcs d'une courbe algébrique g(x,y)=0 : la donnée de deux points sur la courbe ne définit pas toujours un seul arc, et le test d'appartenance d'un point à un arc n'est pas trivial. Les considérations suivantes ont pour but de résoudre ces problémes.

Un arc est défini par une liste de bouts d'arc. Les bouts d'arc sont l'analogue 2D des bouts de face :

Informatiquement, un bout d'arc est défini par deux abscisses extrêmes x_g et x_d qui peuvent être infinies, avec $x_g < x_d$, et par une profondeur. L'intersection entre une droite d'équation $x-x_v=0$, avec $x_g < x_v < x_d$, compte toujours un point d'intersection et un seul, dont l'ordonnée y_v est le p-ième zéro du polynôme "*squarefree*" $g(x_v,y)=q(y)$, p étant la profondeur du bout d'arc.

Toute liste de bout d'arcs n'est pas forcément un arc : un arc est continu et connexe, ne contient aucun point singulier de la courbe g=0, et l'intersection entre l'arc et une droite d'équation $x-x_v=0$ passant strictement entre les abscisses extrêmes d'un quelconque de ses bouts d'arc compte un point et un seul. Un arc est défini plus rigoureusement par son procédé de construction :

Comment construire les bouts d'arc et les arcs de la courbe g(x,y)=0 ?

Les points ordinaires à normale parallèle à Ox, et les points singuliers de la courbe, sont déterminés en résolvant $g(x,y)=g_y(x,y)=0$; schématiquement, les x sont solutions de $R_y[g,g_y](x)=0$; soit x^* une solution; on résoud ensuite $PGCD(g(x^*,y), g_y(x^*,y)) = q(y)=0$. Des méthodes permettent d'obtenir une représentation exacte des solutions, par exemple [LOOS 82], [COLL 82]. Ces points, appelés "sommets" de la courbe, sont classés par abscisses croissantes.

Pour simplifier l'exposé, on suppose que deux points distincts ne peuvent avoir même abscisse; après tout, il existe une infinité de repères Oxy tels que cette condition soit vraie, et seulement un nombre fini (O donné) tels que cette condition soit fausse. Soit $((x_1,y_1), (x_2,y_2)..., (x_n,y_n))$ la liste des points obtenus. On les appelle "sommets" de la courbe. Avec la contrainte précédente, si le polynôme $g(x_k,y)=q(y)$ n'est pas "*squarefree*", y_k est le seul zéro multiple.

♦ **Remarque** Cet ensemble de points est le "contour apparent" de g sur Ox. Tout comme g(x,y)=0 est le contour apparent de f(x,y,z)=0 sur Oxyz. Il semble possible de raisonner récursivement sur la dimension, mais cet exposé se contente du cas tridimensionnel.

Une étude locale en chacun de ces points S détermine le nombre N(S) de parties continues de la courbe g(x,y)=0 qui "naissent" en S, et le nombre M(S) de parties qui "meurent" en S : naissance et mort se définissent par analogie avec le chapitre I. On rappelle brièvement que l'équation, dite mnémonique, de toutes les droites tangentes en un point multiple S d'ordre p (qui annulent toutes les dérivées partielles de l'ordre 1 à p exclu) est : $((x-x_S)g_x + (y-y_S)g_y)^p = 0$; avec cette notation, g_x^k désigne la k-ième dérivée en x de g; se reporter à [WALK 50] et [CASA 65] pour plus de précision. Un sommet S tel que M(S)=0 est appelé sommet de nais-

sance. Ce réemploi de la terminologie du chapitre I ne doit rien au hasard.

La courbe est ensuite balayée de gauche à droite par une droite d'axe Oy, représentée par une liste ordonnée de jetons; chaque jeton représente un bout d'arc actif (qui coupe la droite de balayage). La barre de balayage, B, est initialisée avec autant de jetons qu'il y a de solutions réelles à $g(x_0,y)=q(y)=0$: x_0 est un nombre quelconque inférieur à x_1, et chaque jeton représente une branche infinie active pour $x \to -\infty$. Sur chaque jeton est marqué son sommet gauche : ce sont ici des points à l'infini de la courbe g(x,y)=0.

Ensuite, pour chaque point $S=(x_i, y_i)$, de la gauche vers la droite, on fait :

Si $M(S) \neq 0$, on détermine le rang dans B du jeton représentant le bout d'arc le plus bas qui meurt en S. Les séquences de STURM [KNUT 81] [WIJK 84] permettent de déterminer le nombre k de zéros réels du polynôme $SQRF(g(x_S, y)) = q(y)$ dans $]-\infty, y_S[$. Les M(S) bouts d'arc mourant en S sont représentés dans B par les k+1, k+2, ...k+M(S)-ièmes jetons. Ces jetons sont enlevés de B; la liste des bouts d'arcs de l'arc représenté par chacun de ces jetons est alors connu; le sommet gauche de l'arc est connu : il a été marqué sur le jeton lors de sa création; le sommet droit est connu aussi : S bien sûr.

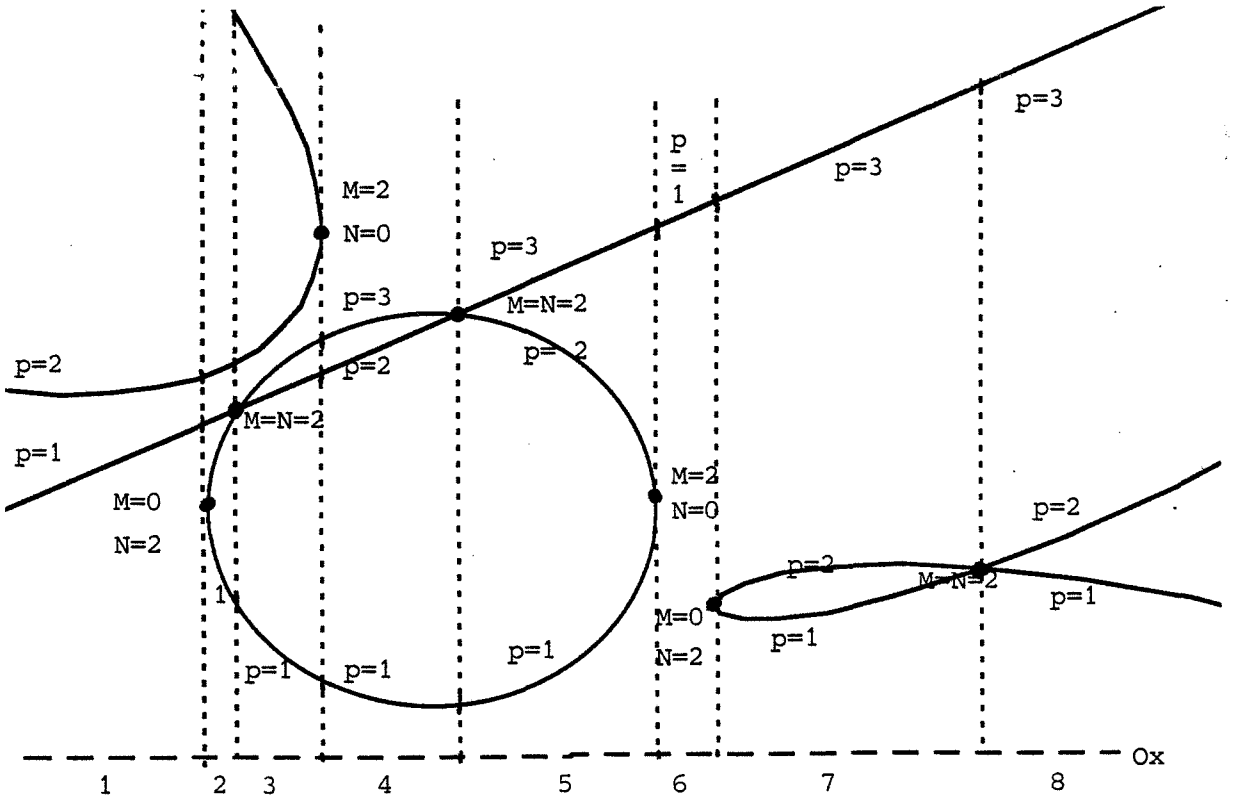


Fig. 1 : Les sommets, bouts d'arc et arcs d'une courbe algébrique.

Si N(S) ≠ 0, on détermine, comme au paragraphe précédent, le rang dans B du jeton représentant le bout d'arc naissant le plus bas qui naît en S. Les N(S) jetons sont insérés dans B à la k+1, k+2, ...k+N(S)-ièmes positions; leur sommet gauche, S bien sûr, est marqué sur les jetons.

Si M(S)=0 (S est un sommet de naissance), il est possible de déterminer l'arc de "butée" de S, par analogie avec le chapitre I. La butée de S est l'arc du bout d'arc représenté par le k-ième jeton (cet arc vaut nil quand k=0).

♦ **Remarque** Le cas des sommets S isolés : M(S)=N(S)=0, ne pose pas de difficultés particulières; ce cas survient par exemple pour $f(x,y,z)=x^2+y^2-z^2=0$, un cône d'axe Oz; alors $g(x,y)=x^2+y^2=0$ se réduit au sommet isolé (0,0).

Soit un jeton J qui n'a été ni inséré ni supprimé dans B lors du traitement de S : alors le bout d'arc que représentait J juste avant S, et le bout d'arc que représente J juste après S, appartiennent au même arc. La liste des bout d'arc de l'arc de J est mise à jour; *c'est cette construction qui définit les arcs, sans ambiguïté*; il est nécessaire de connaître le rang dans B de J, pour noter la profondeur du bout d'arc, ce qui est trivial.

Le traitement de S est achevé : la méthode traite le sommet suivant. Quand $x\rightarrow\infty$, B ne contient plus que des jetons représentant des bouts d'arc infinis. A la fin du balayage, les arcs et leur liste de bouts d'arcs sont connus.

Cette représentation des arcs permet bien de résoudre les problèmes cités au début de ce paragraphe, et quelques autres :

Un point M appartient-il à un bout d'arc a ? M∈a si et seulement si M est une des extrêmités de a, ou bien si x_M est inclus entre les abscisses extrêmes du bout d'arc, si g(M)=0, et si y_M est le i-ième zéro du polynôme "*squarefree*" $g(x_M,y)=q(y)$, i étant la profondeur de l'arc a.

Un point M appartient-il à un arc A ? Ce problème se réduit trivialement au précédent.

Cette représentation permet aussi d'échantillonner (par exemple pour le tracer) les bouts d'arc, et donc les arcs : soit un échantillon de nombres inclus entre les abscisses extrêmes d'un bout d'arc : x_1, x_2, ...x_n; les points correspondants sur le bout d'arc, de profondeur k, sont les k-ièmes zéros réels des polynômes $g(x_1,y)=q_1(y)$, $g(x_2,y)=q_2(y)$, ...$g(x_n,y)=q_n(y)$. (Bien sûr, on peut imaginer des méthodes incrémentales plus efficaces).

Cette représentation permet aussi de trouver le bout d'arc (et donc l'arc) situé immédiatement sous un point donné : c'est une variante du problème précédent. Connaissant l'arc et la région située au dessus, on peut donc déterminer la région contenant un point donné. Mais la représentation des régions n'est pas encore acquise. Comment faire ?

La technique des cartes planaires paraît toute indiquée. Supposons, pour commencer, que g(x,y)=0 ne contienne pas de branche infinie (voir [CASA 65] [WALK 50] pour la déterminations des branches infinies). Il suffit alors de remplacer les arêtes du chapitre I par des arcs, les sommets et les points d'intersection du chapitre I par les extrêmités des arcs. L'x-ordre, l'α-ordre, les butées, les brins ou demi-arcs se définissent comme en I. La CP-locale s'obtient par un algorithme identique; elle est tout de suite cohérente, puisque les arcs ne se coupent que sur leurs extrémités. La méthode construisant la CP-globale à partir de la CP-locale peut être réutilisée telle quelle, sans modification aucune. Seul phénomène vraiment nouveau : il peut exister des bords internes comptant seulement deux arcs (ou plutôt deux demi-arcs), ce qui ne perturbe pas le fonctionnement des algorithmes de carte planaire.

Que faire quand la courbe g(x,y)=0 contient des branches infinies ? On se cantonne ici à la solution la plus simple : la courbe est fenêtrée ("*clipped*") par un très grand carré [-Ω, Ω], qui contient tous les points à distance finie intéressants; c'est à dire toutes les extrêmités à distance finie des arcs, tous les points ordinaires à normale de direction Oy (obtenus en résolvant $g(x,y)=g_x(x,y)=0$), et tous les points d'intersection entre les butées des sommets de naissance et la verticale passant par ces sommets. L'intersection entre les branches infinies et le cadre du carré se réduit à la résolution de quatre équations algébriques à une seule inconnue. On ajoute à la courbe ainsi fenétrée les arcs (de simples arêtes) frontières du carré [-Ω, Ω]. Il faut corriger les butées valant nil : il suffit de trouver l'arête sur la droite $y+\Omega=0$ qui se trouve juste sous le sommet de naissance. Les CP-locale et globale sont ensuite construites.

Cette carte planaire permet de lister les régions, ou zones, et leurs arcs frontières; la notion de brin, ou demi-arc, permet de savoir si une région se trouve au dessus (vers les y positifs) ou au dessous (vers les y négatifs) d'un arc qui la limite. Enfin, il n'est pas difficile de reconnaître, au vol, les arcs infinis et les arêtes du cadre.

4 DE L'ARBRE CSG A LA BREP

La représentation des arcs et des régions du contour apparent de f est donc acquise. Il reste à reconstituer les informations 3D sur cette projection Oxy :

Une première méthode est présentée; elle ne sera pas retenue mais ses insuffisances permettent de bien comprendre le problème :

Pour chaque région, le nombre de faces actives (qui se projettent dans la région) est calculé comme suit : on détermine pour un point M quelconque strictement intérieur à la région le nombre de zéros réels du polynôme "*squarefree*" $f(x_M,y_M,z)=q(z)$ contenus dans $]-\infty, \infty[$, grâce aux séquences de STURM.

Les bouts de face sont alors connus : un bout de face est parfaitement décrit par sa région et sa profondeur.

Cependant, cette méthode est insuffisante, car des informations topologiques essentielles restent implicites : considérons deux régions G et D situées à gauche et à droite d'un arc a. Comme g est "*squarefree*" par hypothèse, a est la projection d'un seul arc de contour apparent (ou d'auto-intersection) de f(x,y,z)=0; soit A cet arc; certains des bouts de face actifs en G ou en D sont incidents en A, d'autres non; pour tout bout de face u actif en G et non incident en A, il existe un bout de face actif en D, et un seul : v, tel que u et v se "raccordent" de façon continue; u et v passent "en dessous" ou "en dessus" de A (tout comme deux bouts d'arcs successifs et appartenant au même arc passent "au dessus" ou "au dessous" d'un sommet). *Par définition, u et v appartiennent à la même face.* Comment trouver les bouts de face incidents en A, et comment trouver la liste des bouts de chaque face ?

La deuxième méthode est la bonne :

On considère successivement chaque arc a de la courbe g(x,y)=0, dans un ordre quelconque. Soit A l'arc dont a est la projection. Soit $M_a=(x_M,y_M)$, un point ordinaire de a, différent des extrêmités de a. M_a est la projection du point M_A de A, dont la cote z_M sera déterminée ultérieurement. On suppose de plus que la normale à la courbe g(x,y)=0 en M_a n'est pas de direction Oy : il existe toujours une infinité de tels points M, puisque nous avons interdit les droites de direction Ox dans g. La méthode utilise un plan auxilliaire de coupe passant par M_A; l'exposé suppose qu'il s'agit du plan vertical P parallèle à Oxz d'équation $y-y_M=0$. La courbe d'intersection entre P et la surface f(x,y,z)=0 a pour équation $f(x,y_M,z)=h(x,z)=0$; l'expression de h est déterminée; dans le cas général, h(x,y) est non identiquement nul et "*squarefree*"; on ne traitera que ce cas.

La méthode exploite les idées suivantes :

La courbe h(x,z)=0 dans P peut s'étudier comme la courbe h(x,y)=0 dans Oxy; les arcs de h(x,z)=0 sont déterminés par la méthode du balayage.

La courbe h(x,z)=0 a un seul sommet M_P d'abscisse x_M : $M_P=(x_M,z_M)$; on en déduit la cote de $M_A=(x_M,y_M,z_M)$. La cote z_M du point M_A peut aussi être calculée en remarquant que z_M est le seul zéro multiple du polynôme $f(x_M,y_M,z)=t(z)=0$; ou encore : z_M est le zéro du polynôme $PGCD(t(z),t_z(z))$ qui, par construction, est du premier degré. Cette méthode permet de trouver la cote de tous les points M_A pour M_a donné.

Le k-ième bout de face actif en G (respectivement en D) "est", dans P, le k-ième bout d'arc actif à gauche (respectivement à droite) de M_P. En effet, le k-ième bout d'arc juste à gauche (respectivement à droite) de M_P dans P est la coupe du k-ième bout de face actif en G (respectivement en D).

Si le k-ième bout d'arc actif à la gauche de M_P et si le j-ième bout d'arc actif à droite de M_P appartiennent au même arc de la courbe h(x,y)=0, alors le k-ième bout de face actif en G et le j-ième bout de face actif en D appartiennent à la même face. Cette condition est nécessaire et suffisante.

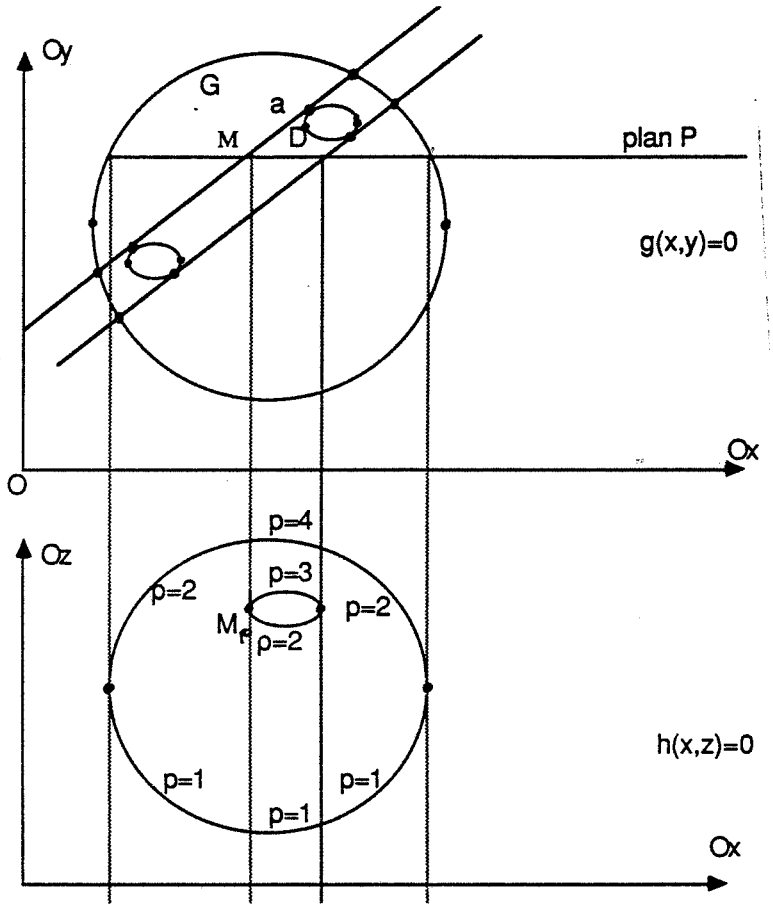
Si le i-ième bout d'arc actif à gauche de M_P meurt en M_P, alors le i-ième bout de face actif en G est incident à l'arc A, qui se projette en a. Cette condition est nécessaire et suffisante. De même, si le j-ième ème bout d'arc actif à droite de M_P naît en M_P, alors le j-ième bout de face actif en G est incident en A.

Déterminer les faces de la surface f(x,y,z)=0 se réduit donc à un problème 2D déjà traité, la détermination des arcs d'une courbe. Insistons sur le fait que la méthode précédente a été retenue pour sa simplicité et sa lisibilité, et non pour son efficacité.

A ce stade, la représentation des faces de la surface f(x,y,z)=0 est acquise. Certaines faces sont utiles : elles séparent l'intérieur de l'extérieur de l'objet décrit par l'arbre CSG; les autres faces sont inutiles. Pour achever la construction de la BREP, il faut reconnaître les faces utiles. La définition précédente est exploitée directement par la méthode naïve suivante :

Pour chaque face, il suffit de considérer un de ses bouts de face, n'importe lequel. Soient R et p la région et la profondeur du bout. Un point $M=(x_M,y_M)$ strictement intérieur à R est généré. On sait que la cote z_M de l'unique point d'intersection entre le bout de face et la droite verticale passant par M est le p-ième zèro du polynôme "*squarefree*" $f(x_M,y_M,z)=q(z)$. Les séquences de STURM permettent de générer un nombre z_b situé strictement entre le (p-1)-ième et le p-ième zéros, et un

Fig. 2



Aucun bout d'arc à gauche de M_p n'est incident en M_p. <-> aucun bout de face de G n'est incident en A

Le 2 ième et le 3 ième bout d'arc à droite de M_p sont incidents en M_p <-> le 2 ième et le 3 ième bout de face en D sont incidents en A.

Le 1 er bout d'arc à gauche de M_p et le 1 er bout d'arc à droite de M_p appartiennent au même arc <-> le 1 er bout de face dans G et le 1 er bout de face en D appartiennent à la même face.

Le 2 ième bout d'arc à gauche de M_p et le 4 ième bout d'arc à droite de M_p appartiennent au même arc <-> le 2 ième bout de face dans G et le 4 ième bout de face dans D appartiennent à la même face.

second nombre z_h situé strictement entre le p-ième et le (p+1)-ième zéros; la procédure récursive classique [ROSS 86] détermine si les points (x_M, y_M, z_b) et (x_M, y_M, z_h) appartiennent ou non à l'objet décrit par l'arbre CSG. La face est utile si et seulement un seul de ces deux points est intérieur. Du même coup, il est possible de savoir de quel côté (vers les z positifs, ou vers les z négatifs ?) de la face se trou-

vent l'intérieur et l'extérieur du solide.

On dispose maintenant de la description des faces utiles de la surface f(x,y,z)=0, autrement dit des faces de l'objet décrit par l'arbre CSG. Les arcs frontières des faces sont connus.

Ces informations suffisent pour trouver, par exemple, les diverses composantes connexes de l'enveloppe de l'objet, et leur liste de faces : par définition, deux faces incidentes au même arc appartiennent à la même composante connexe. Situer ces diverses composantes connexes les unes par rapport aux autres est un problème déjà rencontré, et traité, en III, où on a vu qu'il suffit de savoir calculer l'intersection entre une face et une droite non particulière. Ce problème se réduit d'une part au calcul des points d'intersection entre la surface f(x,y,z)=0 et la droite (question classique en synthèse d'images, depuis l'emploi du lancer de rayons), d'autre part au test d'appartenance de chacun de ces points à la face. Ce dernier problème est résolu en testant l'appartenance du point à chaque bout de face de la face (l'emploi d'une arithmétique algébrique supprime les difficultés de l'imprécision).

Par certains points, cette méthode ressemble à la méthode de ARNON [ARNO 82]. Cependant, l'algorithme de ARNON construit une représentation par énumération spatiale ("*A Cellular Decomposition of Semi Algebraic Sets*") de l'objet décrit par l'arbre CSG, et non pas une BREP. De plus, ARNON ne fusionne pas les bouts d'arcs en arcs, et les bouts de face en faces; il ne détaille pas non plus la méthode déterminant les incidences entre sommets, arcs, et faces (ou plutôt ce qui en tient lieu : les "cellules" de dimension 0, 1 et 2).

5 ANNEXE : LA REPRESENTATION DES ALGEBRIQUES

Les calculs des résultants, des PGCD, des séquences de STURM ont déjà été employés en synthèse d'images ou en "*solid modeling*" : [KAJY 82], [WIJK 84], [SEDE 84], [SEDE 86]. Les arithmétiques algébriques sont moins familières. Le principe de deux arithmétiques algébriques possibles est très brièvement présenté.

5.1 LA REPRESENTATION POLYNOMIALE

La première représentation ([KNUT 81] [LOOS 82]) des nombres algébriques traite les nombres algébriques réels (les seuls qui nous soient nécessaires). Elle s'inspire de la définition première des nombres algébriques : un nombre algébrique est la solution d'une équation algébrique de degré fini à coefficients entiers relatifs (ou rationnels). Un nombre algébrique est donc représenté par l'équation dont il est solution. Pour différencier ce nombre des autres racines réelles de la même équation, on attache au nombre un encadrement rationnel qui l'isole des autres racines; les séquences de STURM permettent d'isoler les racines réelles d'un polynôme

dans des encadrements rationnels disjoints. Comment effectuer les opérations (+ - × /) sur ces représentations ? On considère le cas de l'addition : soient a une racine de $\alpha(x)=0$ et b une racine de $\beta(x)=0$; α et β représentent respectivement a et b. Le polynôme $\gamma(x)$ représentant c=a+b s'obtient en éliminant a et b du système : $\alpha(a)=0$ et $\beta(b)=0$. et $a+b-c=0 \rightarrow \alpha(a)=0$ et $\beta(c-a)=0$; donc c vérifie $\gamma(c)=R_a[\alpha(a),\beta(c-a)](c)=0$. L'encadrement de c est obtenu en effectuant l'addition sur les encadrements de a et b; c sera bien le seul zéro de γ dans cet encadrement si les encadrements de b et c sont assez étroits; ces derniers peuvent être rendus aussi étroits que nécessaires grâce aux séquences de STURM, par dichotomie.

On procède de même pour les autres opérations. Il est aussi possible d'extraire les racines k-ième d'un nombre algébrique.

La comparaison de deux nombres se réduit à la détermination du signe de leur différence. Un nombre est nul si et seulement si son encadrement contient 0 et si le coefficient du terme de degré 0 de son polynôme est nul. Sinon, il suffit que son encadrement soit précisé jusqu'à ne plus contenir 0, et alors son signe est connu.

Le résultant de deux polynômes de degré n et p est au plus de degré np (théorème de BEZOUT). Le degré des polynômes décrivant les nombres peut donc croître très rapidement si aucune factorisation n'est effectuée; mais la factorisation des polynômes est une opération lente.

5.2 LA REPRESENTATION VECTORIELLE

Une deuxième représentation possible des nombres algébriques exploite les propriétés des extensions algébriques, qui ont à la fois une structure de corps commutatif et d'espace vectoriel. On illustre le principe de cette représentation sur un exemple simple, l'extension de gêne $\sqrt{2}$, notée $Q(\sqrt{2})$:

A partir des nombres rationnels et de $\sqrt{2}$, en employant les opérations (+ - × /), on peut générer une infinité de nombres algébriques (c'est cet ensemble qui est noté $Q(\sqrt{2})$), mais tous s'écrivent sous la forme : $x + y\sqrt{2}$, avec x et y rationnels. En effet :

$(a + b\sqrt{2}) + (c + d\sqrt{2}) = (a+c) + (b+d)\sqrt{2}$
$(a + b\sqrt{2}) - (c + d\sqrt{2}) = (a-c) + (b-d)\sqrt{2}$
$(a + b\sqrt{2}) \times (c + d\sqrt{2}) = (ac + 2bd) + (ad + bc)\sqrt{2}$
$1 / (a + b\sqrt{2}) = (a-b\sqrt{2}) / ((a + b\sqrt{2})(a - b\sqrt{2})) = a/(a^2 - 2b^2) - (b/(a^2 - 2b^2))\sqrt{2}$

Ces nombres peuvent donc être représentés de façon exacte par des vecteurs à composantes rationnelles, avec la base (1, $\sqrt{2}$). Comme la multiplication par un de ces nombres est une transformation linéaire, on peut préférer les représenter par des matrices 2x2 : $a+b\sqrt{2}$ est "identifié" à la matrice ${}^t((a\ b)\ (2b\ a))$; la somme, la différence et le produit de deux nombres ont respectivement pour matrice la somme, la différence et le produit des deux matrices; la matrice pour $1/(a+b\sqrt{2})$ est l'inverse de la matrice pour $a+b\sqrt{2}$. L'équation caractéristique de la matrice est une puissance du polynôme minimal du nombre algébrique représenté.

On aurait pu prendre pour exemple $Q(\sqrt{-1})$: cette représentation est applicable indifféremment aux algébriques réels et complexes.

Ceci se généralise pour tout nombre algébrique u de polynôme minimal ϕ, de degré n. Les nombres x de Q(u) sont représentés par un vecteur à composantes rationnelles dans la base $(u^0, u^1, ...u^{n-1})$: x=X(u), X étant un polynôme de degré n-1 au plus, à coefficients rationnels ([SAMU 71], [MALL 85], [BORO 66]).

L'addition et la différence s'effectuent trivialement sur les composantes. Le produit de x=X(u) et de y=Y(u) s'effectue ainsi : P=XY est un polynôme de degré 2n-2 au plus; P est divisé par ϕ : $P=D\phi+R$, R de degré inférieur à n; $xy=P(u)=D(u)\phi(u)+R(u)=R(u)$. L'inverse de x=X(u) s'obtient ainsi : X et ϕ sont premiers (car ϕ est minimal, et le degré de X est inférieur au degré de ϕ); il existe donc deux polynômes A et B de BEZOUT, déterminés grâce à l'algorithme d'EUCLIDE ([KNUT 81]), tels que : $AX+B\phi=1$. Donc $1/X(u)=(A(u)X(u)+B(u)\phi(u))/X(u) = A(u)$; le degré de A est bien inférieur à n, par construction.

Tant que l'on calcule dans une extension Q(u), tout se passe bien; mais que faire quand survient un nombre algébrique v d'une autre source ? [LOOS 82] construit un nombre algébrique w, de polynôme minimal ω, tel que Q(w) contienne Q(u) et Q(v), puis il détermine U et V tels que u=U(w), v=V(w) : ce qui permet d'exprimer tout vecteur de Q(u) et de Q(v) dans Q(w). Cet algorithme est lent, et est le principal inconvénient de cette représentation des algébriques.

La représentation des nombres algébriques est l'objet de recherches actives en calcul symbolique; une variante plus efficace de la représentation vectorielle a été proposée dans [DICR 86].

6 CONCLUSION

Ce chapitre a proposé un algorithme qui construit une BREP exacte d'un objet décrit par un arbre CSG algébrique. L'exposé de l'algorithme ignore les cas particuliers; par contre, les méfaits de l'imprécision numérique sont pris en compte.

Théoriquement, le passage d'un arbre CSG algébrique à une BREP parfaitement exacte est donc possible. Pratiquement, l'algorithme proposé est d'une terrible inefficacité, ne serait-ce que par l'emploi d'une arithmétique algébrique, extrêmement pénalisante. Cependant, les progrès ultérieurs du calcul symbolique et des performances des matériels informatiques résoudront peut être ces difficultés : en 1968, la méthode du lancer de rayon était impraticable ...

BIBLIOGRAPHIE

[ALT 81] H. ALT, K. MELHORN, I. MUNRO, "On The Complexity of Partial Match Retrieval", MFCS, LNCS 118, 156-161, 1981.

[ANSA 85] S. ANSALDI, L. de FLORIANI, B. FALCIDIENO, "Geometric Modeling of Solid Objects By Using A Face Adjency Graph Representation", SIGGRAPH'85, ACM Computer Graphics 19(3), 131-139, July 1985.

[ARGE 88] J. ARGENCE, thèse, Ecole des Mines de St Etienne, en préparation,

[ARNO 82] D. ARNON, "A Cellular Decomposition Algorithm for Semialgebraic Sets", Symbolic and Algebraic Computation, EUROSAM'79, 301-315, Lecture Notes In Computer Science 72, Springer-Verlag 1979.

[ATHE 83] P. ATHERTON, "A Scan-Line Hidden Surface Removal Procedure for Constructive Solid Geometry", Computer Graphics 17(3), 73-82, July 1983.

[AYAL 85] D. AYALA, P. BRUNET, R. JUAN, I. NAVAZO, "Object Representation By Means of Nonminimal Division Quadtrees and Octrees", ACM Transactions on Graphics 4(1), January 1985, 41-59.

[BARR 84] A.H. BARR, "Global and Local Deformations of Solid Primitives", SIGGRAPH'84, ACM Computer Graphics 18(3), 21-30, July 1984.

[BAUM 75] B.G. BAUMGART, "A Polyhedron Representation for Computer Vision", National Computer Conference, 589-596, 1975.

[BEIG 87] M. BEIGBEDER, thèse, Ecole des Mines de St Etienne, en préparation,

[BENT 75] J.L. BENTLEY, "Multidimensional Binary Search Trees Used for Associated Searching", Communications of The ACM 18, 509-517, 1975.

[BENT 79] J.L. BENTLEY, T. OTTMAN, "Algorithms for Reporting and Counting Geometric Intersections", IEEE Transactions on

Computers, C-28(9), 1979, 643-647.

[BERG 70] C. BERGE, Graphes et hypergraphes, Dunod, Paris, 1970.

[BERG 86] P. BERGERON, "A General Version of Crow Shadow Volumes", IEEE Computer Graphics and Applications 6(9), 17-22, September 1986.

[BEZI 86] P. BEZIER, "Courbes et surfaces", Mathématiques et CAO (4), Hermès, 1986.

[BORO 66], Z.I. BOROVIC et I.R. SAFAREVIC, "Théorie des nombres", Gauthiers Villars, 1966.

[CARL 82] W.E. CARLSON, "An Algorithm and Data Structure for 3D Object Synthesis Using Surface Patch Intersections", SIGGRAPH'82, ACM Computer Graphics 16(3), 255-263, July 1982.

[CASA 87] M.S. CASALE, "Free-Form Solid Modeling with Trimmed Surface Patches", IEEE Computer Graphics and Applications, 7(1), 33-43, January 1987.

[CASA 65] G. CASANOVA, "Mathématiques spéciales", tome II : "Algèbre et Analyse"; tome III :"Géométrie analytique", Eugène Belin, Paris, 1965.

[CAST 85] P. de CASTELJAU, "Formes à pôles", Mathématiques et CAO (2), Hermès, 1985.

[CHAZ 80] B.M. CHAZELLE, "Computationnal Geometry and Convexity", Department of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvannia, July 1980.

[CHAZ 83] B.M. CHAZELLE, "Reporting and Counting Arbitrary Planar Intersections", Technical Report CS-83-16, Brown University.

[COLL 82] G.E. COLLINS, R. LOOS, "Real Zeros of Polynomials", Computer Algebra, Computing Supplementum 4, 83-94, Springer-Verlag 1982.

[COQU 84] S. COQUILLART, "Représentation de paysages et tracé de rayon", thèse, Ecole des Mines de St Etienne, 1984.

[CROW 77] F.C. CROW, "Shadow Algorithms for Computer Graphics", SIGGRAPH'77, ACM Computer Graphics 11(2), 242-248, Summer 1977.

[CROW 76] F.C. CROW, "The Aliasing Problem In Computer Synthetized Shaded Images", Ph.D Thesis, University of Utah, March 1976.

[CROW 81] F.C. CROW, "A Comparision of Antialiasing Techniques", IEEE Computer Graphics and Applications 1(1), 40-48, January 1981.

[CROW 84] F.C. CROW, "Summed-Area Tables for Texture Mapping", ACM Computer Graphics 18 (3), 207-212, July 1984.

[DAVE 79] J. DAVENPORT, "Integration of Algebraic Functions", Symbolic and Algebraic Computation, EUROSAM'79, 412-325, Lecture Notes In Computer Science 72, Springer-Verlag 1979.

[DICR 86] C. DICRESCENZO, D. DUVAL, "Algebraic Computation on Algebraic Numbers", in "Computers and Computing", 54-61, Masson-Wiley 1986.

[FEIB 80] E.A. FEIBUSH, M. LEVOY, R.L. COOK, "Synthetic Texturing Using Digital Filters", ACM Computer Graphics 14(3), July 1980, 295-301.

[FRED 81] M.L. FREDMAN, "A Lower Bound of The Complexity of Orthogonal Range Queries, ACM Journal, 28, 696-705, 1981.

[GANG 84] M. GANGNET, D. MICHELUCCI, "Un outil graphique interactif", Rapport 84-12, Ecole des Mines de St Etienne, octobre 1984.

[GANG 84b] M. GANGNET, D. GHAZANFARPOUR-KHOLENDJANY, "Comparaison de techniques de mise en perspective de textures planes", CESTA 84, 29-35, Biarritz, mai 1984.

[GANG 87] Communications personnelles.

[GHAZ 85] D. GHAZANFARPOUR-KHOLENDJANY, "Synthèse d'images et antialiassage", thèse, Ecole des Mines de St Etienne, 1985.

[GLAS 84] A.S. GLASNER, "Space Subdivision for Fast Ray Tracing", IEEE Computer Graphics and Applications, 4(10), 15-22, October 1984.

[GOLD 86] J. GOLDFEATHER, J.P.M. HULTQUIST, H. FUCHS, "Fast Constructive Solid Geometry Display In The Pixel-Powers Graphics System", SIGGRAPH'86, ACM Computer Graphics 20(4), 107-116, August 1986.

[GREE 86] D. H. GREENE, F. F. YAO, "Finite-Resolution Computational Geometry", 143-152, IEEE 1986.

[HARD 79] G.H. HARDY, E.M. WRIGHT, "An Introduction to the Theory of Numbers", Chap. XVIII, 268-269, Theorem 330, Oxford University Press, Fifth Edition, 1979.

[HOOK 86] T.V. HOOK, "Real Time Shaded NC Milling Display", SIGGRAPH'86, ACM Computer Graphics 20(4), 15-20, August 1986.

[HORO 84] E. HOROWITZ, S. SAHNI, "Fundamentals of Data Structures In Pascal", Pitman 1984.

[KAJY 82] J.T. KAJYIA, "Ray Tracing Parametric Patches", SIGGRAPH'82,

ACM Computer Graphics 16(3), 245-254, July 1982.

[KNUT 81] D. KNUTH, "The Art of Computer Programming", tome 2, second edition, Addison-Wesley 1981.

[KOIS 85] P. KOISTINEN, M. TAMMINEN, H. SAMMET, "Viewing Solid Models By Bintree Conversion", EUROGRAPHICS'85, 147-157, Nice, September 1985.

[KUNI 85] L.T. KUNII, G. WYVILL, "CSG and Ray tracing Using Functional Primitives", Computer Generated Images, Graphics Interface'85, 137-152, Springer-Verlag, 1985.

[LAID 86] D.H. LAIDLAW, J.F. HUGHES, "Constructive Solid Geometry for Polyhedral Objects", SIGGRAPH'86, ACM Computer Graphics 20(4), 161-168, August 18-22, 1966, Dallas, Texas.

[LAKA 84] I. LAKATOS, "Preuves et réfutations, essai sur la logique de la découverte mathématique", Hermann 1984.

[LAZA 78] D. LAZARD, "Résolution algébrique des systèmes d'équations algébriques", Congrès AFCET-SMF (Palaiseau 1978).

[LAZA 79] D. LAZARD, "Systems of Algebraic Equations", 1979.

[LEE 77] D.T. LEE, C.K. WONG, "Worst-Case Analysis for Region and Partial Region Searches In Multidimensional Binary Search Trees and Balanced Quad Trees", Acta Informatica, 9, 23-29, 1977.

[LEVI 79] J.Z. LEVIN, "Mathematical Models for Determining The Intersections of Quadric Surfaces", Computer Graphics and Image Processing 11, 73-87, 1979.

[LEVI 80] J.Z. LEVIN, "Quadril : A Computer Language for the Description of Quadric Surface Bodies", SIGGRAPH'80, ACM Computer Graphics 14(3), 86-92, July 1980.

[LOOS 82] R. LOOS, "Computing In Algebraic Extensions", Computer Algebra, Computing Supplementum 4, 173-182, Springer-Verlag 1982.

[MAIR 83] H.G. MAIRSON, J.STOLFI, "Reporting Line Segment Intersections In The Plane", Technical Report, Department of Computer Science, Stanford University, 1983.

[MALH 76] K. MALHER, "Lectures on Transcendental Numbers", Springer-Verlag, Lectures Notes on Mathematics, 1976.

[MALL 85] M.P. MALLIAVIN, "Algèbre commutative : applications en géométrie et théorie des nombres", Masson 1985.

[MANT 82] M. MANTYLA, R. SULONEN, "A Solid Modeler with Euler Operators", IEEE Computer Graphics and Applications 2(7), 17-31, 1982.

[MANT 83] M. MANTYLA, "Set Operations of GWB", Computer Graphics Forum 2(3), 122-134, August 1983.

[MANT 83] M. MANTYLA, M. TAMMIMEN, "Localized Set Operations for Solid Modeling", 279-288, SIGGRAPH'83, ACM Computer Graphics 17(3), July 25-29, Detroit, Michigan,

[MART 87] P. MARTIN, D. MARTIN, "Les difficultés et les erreurs possibles dans les algorithmes de calcul de l'intersection de solides définis par leur bord", MARI 87, 48-56, mai 1987.

[MELH 84] K. MELHORN, "Data Structures and Algorithms", Springer-Verlag 1984.

[MICH 84] D. MICHELUCCI, M. GANGNET, "Un outil graphique interactif", MICAD 84, Hermès, février-mars 1984, 95-110.

[MICH 86] D. MICHELUCCI, B. PEROCHE, "Boundary Representation Using Trees", Ecole des Mines de St Etienne, rapport interne 86-1, janvier 86.

[MICH 87] D. MICHELUCCI, B. PEROCHE, "Représentation interactive d'arbres CSG", MICAD 87, 535-548. Hermès, février-mars 1987,

[MIDD 85] A.E MIDDLETICH, K.H. SEARS, "Blend Surfaces for Set Theoretic Volume Modeling Systems", SIGGRAPH'85, ACM Computer Graphics 19(3), 161-169, July 1986.

[MILL 86] J.R. MILLER, "Analysis of Quadric Surfaces Based Solid Models", SIGGRAPH'86, Introduction To Solid Modeling, Courses Notes 8, August 18, 1986.

[MOIS 84] J.C. MOISSINAC, "Aides informatiques à la réalisation de dessins animés", thèse, Ecole des Mines de St Etienne, 1984.

[MULL 78] D.E. MULER, F.P. PREPARATA, "Finding The Intersection of Two Convex Polyhedra", Theoretical Computer Science 7(2), 217-236, October 1978.

[MUTA 81] C. MUTAFIAN, "Equations algébriques et théorie de Galois", Vuibert, 1981.

[OTTM 82] T. OTTMAN, P. WIDMEYER, D. WOOD, "A Fast Algorithm for Boolean Mask Operations", Report 112, Institut F. Angewandte Mathematik Und Formale Beschreibungsverfahren, 1982.

[OTTM 82b] T. OTTMAN, P. WIDMEYER, D. WOOD, "A Worst-Case Efficient Algorithm for Hidden Line Elimination", Report 119, Institut F. Angewandte Mathematik Und Formale Beschreibungsverfahren, 1982.

[OTTM 85] T. OTTMAN, P. WIDMEYER, D. WOOD, "A Fast Algorithm for Boolean Mask Operations", Computer Vision, Graphics, and Image Processing 30, 249-268, 1985.

[OVER 81] M.H. OVERMARS, J.V. LEUWEN, "Maintenance of Configurations In The Plane", Journal of Computer and System Science 23, 166-204, 1981.

[PAVE 85] R. PAVELLE, P.S. WANG, "Macsyma from F to G", Journal of Symbolic Computation 1(1), March 1985.

[PERO 87] B. PEROCHE, J. ARGENCE, D. GHAZANFARPOUR, D. MICHELUCCI, "La synthèse d'images", Hermès 1987.

[PREP 85] F.P. PREPARATA, M.I. SCHAMOS, "Computational Geometry, An Introduction", Springer-Verlag, 1985.

[PUEY 87] X. PUEYO, J.C. MENDOZA, "A New Scan Line Algorithm for The Rendering of CGS trees", EUROGRAPHICS'87, 347-361, August 1987.

[REQU 80] A.A.G. REQUICHA, H.B. VOELCKER, "Mathematical Foundations of Constructive Solid Geometry : General Topology of Closed Regular Sets", Production Automation Project Technical Memorandum TM-27a, November 1980.

[REQU 83] A.A.G. REQUICHA, H.B. VOELCKER, "Solid Modeling: Current Status and Research Directions", IEEE Computer Graphics and Applications, 25-36, October 1983.

[REQU 85] A.A.G. REQUICHA, H.B. VOELCKER, "Boolean Operations In Solid Modeling : Boundary Evaluation and Merging Algorithms", IEEE Proceedings, January 1985, 30-40.

[ROMM 87] E. ROMMEL, "Modélisation de terrains polyplissés et polyfaillés", Diplôme d'ingénieur, Département de Géologie, Ecole des Mines de St Etienne, Juin 1987.

[ROSS 86] J.R. ROSSIGNAC, A.A.G. REQUICHA, "Depth-Buffering Display Techniques for Constructive Solid Geometry", IEEE Computer Graphics and Applications, 29-39, September 1986.

[ROTH 82] S. ROTH, "Ray Casting for Modeling Solids", Computer Graphics and Image Processing 18, 1982, 109-144.

[SAME 84] H. SAMET, "The Quadtree and Related Hierarchical Data Structures", ACM Computing Surveys, 16 (2), 187-260, June 1984.

[SAME 86] H. SAMET, R.C. NELSON, "A Consistent Hierarchical Representation for Vector Data", SIGGRAPH'86, ACM Computer Graphics 20(4), 197-206, August 1986.

[SAMU 71] P. SAMUEL, "Théorie algébrique des nombres", Hermann, collection Méthodes, 1971.

[SEDE 84] T.W. SEDERBERG, D.C. ANDERSON, "Implicit Representation of Parametric Curves and Surfaces", Computer Vision, Graphics,

and Image Processing 28, 72-84, 1984.

[SEDE 85] T.W. SEDERBERG, D. ANDERSON, "Steiner Surface Patches", IEEE Computer Graphics and Applications, 5(1), 23-36, 1985.

[SEDE 86] T.W. SEDERBERG, "A Tutorial on Algebraic Geometry for Computer Graphics and Computer Aided Geometric Design", SIGGRAPH'86, Geometry for Computer Graphics and Computer Aided Design, Courses Notes 10, August 1986.

[SEDE 86] T.W. SEDERBERG, S.R. PARRY, "Free-Form Deformation of Solid Geometric Models", SIGGRAPH'86, ACM Computer Graphics 20(4), 151-160, August 1986.

[SZIL 84] M. SZILVASI-NAGY, "An Algorithm for Determining The Intersection of Two Simple Polyhedra", Computer Graphics Forum 3, 219-225, 1984.

[TAKA 86] T. TAKALA, "Geometric Boundary Modeling without Topological Data Structures" EUROGRAPHICS'86, 115-128, August 1986.

[TILO 80] R.B. TILOVE, "Set Membership Classification : A Unified Approach To Geometric Intersection Problems", IEEE Transactions on Computers, 29(10), 874-883.

[TURN 84] J.A. TURNER, "A Set-Operation Algorithm for Two and Three Dimensional Geometric Objects", Report, Architecture and Planning Research Laboratory, University of Michigan, August 1984.

[TURK 86] K. TURKOWSKI, "Anti-Aliasing In Topological Color Spaces", SIGGRAPH'86, ACM Computer Graphics 20(4), 307-314, August 18-22, 1986.

[VANT 85] J.M. VAN-THONG, "Techniques de balayage et élimination de parties cachées", thèse, Université Pierre et Marie Curie (Paris VI), 1985.

[VANT 87] J.M. VAN-THONG, Communications personnelles.

[WALK 50] R.J. WALKER, "Algebraic Curves", Dover Publications Inc., New York, 1950.

[WIJK 84] J.J.V. WIJK, "Ray Tracing Objects Defined By Sweeping A Sphere", EUROGRAPHICS'84, 73-82, September 1984.

[WILL 83] L. WILLIAMS, "Pyramidal Parametrics", SIGGRAPH'83, ACM Computer Graphics 17(3), 1-11, July 1983.

[WILL 78] L. WILLIAMS, "Casting Curved Shadows on Curved Surfaces", ACM Computer Graphics 12(3), 270-274, July 1978.

[YAMA 84] F. YAMAGUCHI, T. TOKIEDA, "A Unified Algorithm for Boolean Shape Operations", IEEE Computer Graphics and

Applications, 24-37, June 1984.

[YUN 76] D.Y.Y. YUN, "On Squarefree Decomposition Algorithms", ACM Symposium on Symbolic and Algebraic Computation, 26-35, New-York 1976.

[YUN 77] D.Y.Y. YUN, "Fast Algorithm for Rationnal Function Integration", IFIP 77, Gilchrist, North-Holland, 493-498.

SOMMAIRE

Résumé : *La synthèse d'images et la CAO utilisent diverses modélisations des solides. Les représentations par les frontières ("Boundary Representations") sont l'une d'elles. Leurs constructions se heurtent à plusieurs difficultés : l'imprécision numérique dont les conséquences néfastes ont peut-être été sous-estimées, les possibles incohérences (comment être sûr qu'une représentation par frontières décrit bien un solide, au sens physique du terme ?) provoquées par les imprécisions numériques et/ou la redondance des représentations par les frontières, et enfin le foisonnement des cas particuliers. Cette thèse détaille les difficultés et quelques solutions nouvelles.*

Mots clés : *représentations par frontières ("BREP"), opérations booléennes sur les solides, élimination des parties cachées, cartes planaires, imprécision numérique, modélisation des solides ("solid modeling").*