Aide géométrique à l'aménagement de satellites

Eelco de Lange

N° d'ordre : Année 1998

THÈSE de DOCTORAT de l'ÉCOLE DES MINES DE PARIS

Spécialité : informatique

préparée à l'**INRIA Sophia-Antipolis** et
MATRA MARCONI Space

présentée par

Eelco DE LANGE

pour obtenir le grade de
DOCTEUR de l'ÉCOLE DES MINES DE PARIS

AIDE GÉOMÉTRIQUE À L'AMÉNAGEMENT DE SATELLITES

Soutenue le 19 février 1998 devant le jury composé de :

Président :	M.	**Yves ROUCHALEAU**
Rapporteurs :	M.	**Bernard LACOLLE**
	M.	**Jean-Paul LAUMOND**
Examinateurs :	M.	**Jean-Daniel BOISSONNAT**
	M.	**Thierry DUHAMEL**
	M.	**André GASQUET**
	Mme	**Monique TEILLAUD**

Remerciements

Je tiens à remercier vivement mes deux *directeurs de thèse Jean-Daniel Boissonnat et Monique Teillaud. Leur appui et leur participation enthousiastes ont fait que cette thèse a pu voir le jour.*

J'exprime ma gratitude à Arnaud de Saint Vincent et à Bernard Foch, qui ont proposé le sujet de l'aménagement géométrique de satellites. Je suis très content d'avoir pu participer à l'introduction dans l'industrie de méthodes innovatrices provenant de la géométrie algorithmique. Merci de m'avoir permis de travailler dans un environnement de haute technologie passionnant.

Malgré leur charge importante, Bernard Lacolle et Jean-Paul Laumond ont accepté d'être rapporteurs. Qu'ils en soient remerciés sincèrement. Merci également à Yves Rouchaleau, qui a pris en charge les aspects administratifs de ma thèse et a accepté de présider ce jury.

Je suis reconnaissant à André Gasquet et à François Lecouat pour l'encadrement de l'activité de placement géométrique à Matra Marconi Space. Je leur exprime ma gratitude pour leur aide dans les démarches techniques et administratives.

Thierry Duhamel est expert scientifique à Matra Marconi Space et il n'est pas inconnu de l'équipe PRISME. Je suis très honoré qu'il ait accepté de participer à ce jury.

Merci à Patrick Noël et à Stéphane Garay de l'équipe de développement du logiciel SYSTEMA, pour leurs conseils techniques.

Je ne dois pas oublier de remercier mes collègues du bureau d'études de Matra Marconi Space pour m'avoir expliqué la problématique de l'aménagement de satellites.

Ecolo ? Peut être, mais ce n'est pas pour épargner de l'encre mais par peur d'oublier quelqu'un que je remercie "en vrac" aussi tous mes autres collègues de l'équipe PRISME, des autres équipes à l'INRIA et des équipes de Matra Marconi Space. Merci pour votre participation, d'une façon ou d'une autre, à mon travail et surtout pour votre convivialité !

Toulouse, février 1998

Notations, définitions et conventions

$\mathbb{N}$	l'ensemble des nombres entiers naturels
$\mathbb{R}^d$	l'espace euclidien de dimension d
$P^C = \{p \in \mathbb{R}^d \mid p \notin P\}$	le complémentaire de l'ensemble P
$\ominus P = \{-p \mid p \in P\}$	le symétrique de l'ensemble P
$A \oplus B = \{a+b \mid a \in A, b \in B\}$	la somme de Minkowski des ensembles A et B
$A \ominus B = A \oplus (\ominus B)$	la différence de Minkowski entre les ensembles A et B
S^2	la sphère unité de $\mathbb{R}^3$
$G(P)$	le diagramme gaussien du polyèdre P
∂P	la frontière du polygone ou du polyèdre P ou de la région polygonale $\mathcal{P}$
$\beta(P)$	nombre de bits nécessaires pour représenter la valeur du polynôme P à variables entières

Nous considérons uniquement des polygones simples. Une région polygonale est un sous-ensemble du plan $\mathbb{R}^2$ dont la frontière est composée d'un ensemble fini de chaînes

polygonales fermées qui ne se coupent pas mais qui peuvent avoir des sommets coincidents. Chacune de ces chaînes trace la frontière d'un polygone ou d'un trou polygonal.

Polygones et polyèdres sont notés en majuscules. Les ensembles de polygones et de polyèdres et les régions polygonales sont notés en caractères calligraphiques. Un polynôme est noté P.

Dans l'espace euclidien $\mathbb{R}^d$, on confond la notion de point et de vecteur. Ainsi, on ne fait pas de différence entre le point p et le vecteur Op, où O est l'origine du repère. On manipulera alors aussi bien des sommes de vecteurs, que des sommes de points, que des sommes de points et de vecteurs.

Table des matières

Liste des figures

Liste des tables

Chapitre 1

Introduction

1.1 Problématique de l'aménagement de satellite

1.1.1 Aménagement

L'aménagement d'un satellite est une tâche qui pose beaucoup de problèmes en raison de la complexité du satellite. Il ne faut pas uniquement trouver la place pour un grand nombre d'équipements et les connecter par des câbles, mais il faut aussi prendre en compte un certain nombre de contraintes de natures thermique, électromagnétique, mécanique et autres. On peut souvent géométriquement modéliser ces contraintes par la spécification de zones interdites et de distances minimales, ce qui permet l'application de méthodes provenant de la géométrie algorithmique.

L'aménagement est une tâche répétitive : pendant la phase de conception, les spécifications du satellite et des équipements changent souvent et on doit pouvoir répondre rapidement à un changement de ces spécifications.

Il y a donc un besoin fort d'outils efficaces aidant l'ingénieur du bureau d'études dans ses tâches d'aménagement répétitives. Il est rare qu'un aménagement soit commencé à partir de rien (on essaie de réutiliser des parties d'aménagement d'autres satellites) et que l'on puisse spécifier toutes les contraintes de la conception (si elles sont déjà toutes modélisables géométriquement). Un outil complètement automatique servira donc peu par rapport à des outils qui gèrent des petits ensembles d'équipements et qui permettent d'intervenir à chaque phase du processus.

1.1.2 Placement d'antennes

Un problème très répétitif et souvent difficile à résoudre est le placement d'antennes sur un satellite de télécommunication. Ce placement est très souvent refait pendant les différentes phases de la conception d'un satellite à cause de changements fréquents des spécifications : des équipements peuvent être rajoutés ou leurs dimensions changées. Les spécifications d'équipements sont même souvent inconnues au moment où l'aménagement est commencé. Dans ce cas, une estimation grossière est faite.

Dans le cas d'un changement des spécifications d'un équipement, on souhaite savoir le plus vite possible si le placement de l'équipement est toujours réalisable sans trop changer les placements des équipements autour, ou sinon comment on peut modifier l'aménagement avec un nombre de modifications modeste. Il faut donc des outils interactifs qui fournissent de l'information sur l'admissibilité de placements, mais on s'intéresse aussi à des facilités de placement automatique de petits nombres d'équipements (un aménagement partiel).

1.2 Etat de l'art

1.2.1 Placement

La géométrie algorithmique propose des méthodes de placement qui sont soit basées sur la notion d'*espace admissible* soit sur l'inspection des *double contacts* possibles entre les formes à placer. Par double contact on entend le placement d'une forme tel qu'elle touche deux fois l'environnement formé par le conteneur plus les formes déjà placées. L'espace admissible représente l'ensemble des configurations permises (aussi bien les contacts que les placements sans contact) pour une forme à placer, où une configuration est une translation et rotation pour chaque forme à placer.

L'état de l'art sur le placement de polygones est pour une grande partie tiré de la synthèse de Daniels [Dan95].

1.2.1.1 Polygones

Le problème du placement a été introduit par Chazelle [Cha83], où il examine le placement en translation et rotation d'un polygone P dans un autre polygone C de tailles m et n respectivement. Le problème de décision consiste à déterminer si le placement est faisable ou pas et ne demande pas de calculer une solution. Chazelle donne un algorithme en temps $O(mn^2)$ qui résoud le cas où C est convexe et un algorithme en temps $O(m+n)$ au cas C est convexe et P peut être translaté uniquement.

Un algorithme naïf pour le cas général tourne en temps $O(m^3n^3(m+n)\log(m+n))$. Un meilleur algorithme, qui produit toutes les solutions en temps $O(m^3n^3 \log mn)$, est donné par Avnaim [Avn89]. Dans la suite, nous considérons le placement en translation uniquement.

Nous utilisons la notation kXY pour dénoter le placement de k polygones de type "X" dans un polygone de type "Y" (le conteneur). Les types de polygones sont "N" pour non convexe, "C" pour polygone convexe, "R" pour rectangle et "P" pour parallélogramme. m dénote la somme des nombres de sommets de tous les polygones à placer et n le nombre de sommets du conteneur.

Il est connu que kNN est NP-complet [Dan95]. Un algorithme naïf qui itère sur chaque combinaison de contacts entre polygones et conteneur aura une complexité de $O(k^2(mn)^{2k} \log mn)$. La table suivante qui est copiée de [Dan95] liste quelques résultats démontrés pour les problèmes kCN et kNN par Fortune, Avnaim et Boissonnat, et Devillers :

k	kCN	kNN
1	$O(mn \log mn)$ [For85]	$O(m^2n^2 \log mn)$ [AB87, Avn89]
2	$O(m^2n^2 \log m)$ [Dev93]	$O(m^4n^4 \log mn)$ [AB87, Avn89]
3	$O(m^3n^3 \log m)$ [Dev93]	$O(m^{14}n^6 \log mn)$ [Avn89]

Avnaim et Boissonnat donnent aussi un algorithme en temps $O(m^{60} \log m)$ pour le problème 3NP : le placement dans un parallélogramme [AB87, Avn89]. L'algorithme est basé sur une formule en termes de sommes de Minkowski et unions et intersections de polygones. Avnaim démontre qu'une telle formule n'existe pas pour 3NN, ce qui veut dire qu'un algorithme pour kNN avec $k > 2$ ne peut pas être basé sur des opérations ensemblistes uniquement [Avn89].

Milenkovic et al. [DML94] offrent trois approches au problème de placement en translation. La première itère sur les lignes qui séparent des polygones convexes et améliore la solution pour 3CN : l'algorithme a une complexité de $O(m^3n \log mn)$. La deuxième utilise MIP (mixed integer programming) pour résoudre kNN et devient lente lorsque $k \geq 4$. La troisième approche pour résoudre kNN est approximée : une pénétration maximale de ϵ entre les polygones est permise. Cette approche a une complexité de $O((1/\epsilon)^k) \log(1/\epsilon) k^6 \log s)$, où s le nombre maximal de sommets dans un polygone qui peuvent être créés lors des opérations booléennes ensemblistes utilisées par l'algorithme.

Milenkovic et Daniels [DM95] donnent un algorithme exact pour kNN qui utilise la programmation linéaire et la subdivision récursive du problème. L'algorithme a une complexité de $O((mn)^{2k+1} LP(2k, O(kmn + k^2m^2)))$, où $LP(a, b)$ est le temps nécessaire pour résoudre un programme linéaire de a variables et de b contraintes.

Bien que exponentielle en k, l'implémentation de l'algorithme a un bon comportement pratique pour $k \leq 3$.

La méthode proposée par Daniels [Dan95] est basée sur :

- la *restriction* des espaces admissibles des polygones,
- l'*évaluation*, pour essayer de conclure *oui, il y a une solution* ou *non, il n'y a pas de solution possible*, sinon
- la *subdivision récursive de l'espace admissible*, qui divise le problème en deux sous-problèmes qui sont évalués séparément.

L'idée de cette approche est de pouvoir répondre rapidement s'il n'y a pas de solution au problème : un grand défaut des méthodes qui itèrent sur les contacts est qu'elles doivent inspecter *tous* les contacts pour conclure que le problème n'a pas de solution, ce qui conduit à une complexité exponentielle. Les problèmes pratiques étant souvent sans solution, ceci n'est pas souhaitable.

La méthode, qui est approximée, bien que exponentielle dans le cas le pire, s'est montrée assez efficace en pratique pour $k \leq 10$. Elle est appliquée pour l'aménagement et le compactage pour la découpe industrielle de tôle.

1.2.1.2 Polyèdres

Soient C et P des polyèdres de tailles n et m respectivement. Avnaim observe que l'ensemble des configurations de P dans le polyèdre convexe C est égal à l'intersection de mn demi-espaces. En conséquence, l'ensemble des configurations pour P est calculé en temps $O(mn \log mn)$ [Avn89].

La somme de Minkowski, comme pour les polygones, peut être utilisée pour calculer l'ensemble de configurations d'un polyèdre telles qu'il n'y a pas collision avec un autre polyèdre. Avnaim propose un algorithme qui calcule la somme de Minkowski des polyèdres non convexes C et P de complexité $O(m^3 n^3 \log mn)$ [Avn89]. Guibas et Seidel proposent une méthode qui calcule efficacement la somme de Minkowski de deux polyèdres convexes C et P [GS87]. La complexité est $O(m + n + K)$, où K est la taille de la somme qui peut atteindre $\Theta(mn)$.

1.2.2 Autres approches

1.2.2.1 Programmation par contraintes

Charman applique dans sans thèse [Cha95] la programmation par contraintes sur le placement de formes en étendant le modèle de l'approche par contraintes pour prendre en compte l'aspect géométrique du problème et en étudiant des heuristiques de placement. La thèse a résulté en l'implantation d'un prototype permettant le placement de rectangles isothétiques.

1.2.2.2 Recherche opérationelle & méthodes stochastiques

Enfin, beaucoup de travail a été fait aussi sur l'application de la recherche opérationnelle, la programmation mathématique et les méthodes stochastiques, surtout dans le domaine des circuits intégrés. Nous citons les articles de Dyckhoff [Dyc90], Dowsland et Dowsland [DD92], Pimont et Tollenaere [PT93] et la thèse de Daniels [Dan95] pour une typologie et une bibliographie exhaustives.

1.2.3 Notre contribution

Nous proposons dans cette thèse une méthode de placement de formes composées d'un ou plusieurs polyèdres convexes sur un plan donné dans un environnement composé de polyèdres convexes. La méthode est basée sur le calcul de sections planes de sommes de Minkowski de polyèdres convexes et le calcul de l'union d'ensembles de polygones. La méthode est appliquée à l'aménagement d'antennes sur un satellite de télécommunication.

Nous avons développé un algorithme optimal et sensible à la sortie qui calcule directement une section plane de la somme de Minkowski de deux polyèdres convexes A et B. L'algorithme est de complexité $O(m + n + K_s)$, où m et n sont les tailles des deux polyèdres et K_s la taille de la sortie. Bien que K_s puisse atteindre une taille de $\Theta(mn)$, il est beaucoup plus petit en pratique que la taille de la somme entière $A \oplus B$.

Nous avons analysé la méthode qui calcule l'union d'un ensemble de polygones par division et fusion. Bien que non sensible à la sortie, le schéma de division et fusion assure un bon comportement dans la plupart des cas pratiques et une bonne complexité théorique lorsqu'on impose certaines contraintes sur les polygones d'entrée.

Nous nous sommes intéressés au traitement des dégénérescences géométriques et aux problèmes de précision numérique qui peuvent apparaître dans le calcul d'une section d'une somme de Minkowski et dans le calcul de l'union d'un ensemble de

polygones.

Ensuite, nous avons vérifié expérimentalement l'efficacité de nos algorithmes sur des modèles réalistes de satellites et nous avons développé des stratégies incrémentales pour le placement de petits ensembles d'équipements.

1.3 Plan du mémoire

Le reste de ce mémoire est organisé de la façon suivante. Le chapitre 2 introduit le problème du placement d'antennes sur un satellite de télécommunication, présente notre méthode pour résoudre ce problème et introduit les outils géométriques et arithmétiques nécessaires pour la mettre en œuvre.

Le chapitre 3 présente un algorithme optimal pour calculer une section plane d'une somme de Minkowski de deux polyèdres convexes et discute sa complexité algébrique. Dans le chapitre 4, nous considérons l'algorithme de Kedem et al. [KLPS86] qui calcule l'union d'un ensemble de polygones par division et fusion et discutons sa complexité algébrique et son comportement pratique.

Le chapitre 5 présente la mise en pratique de la théorie présentée dans les chapitres précédents pour la résolution du problème d'aménagement d'antennes. Le logiciel de placement GEOTOOLS est présenté ainsi que l'expérimentation de ce logiciel. GEOTOOLS visualise les contraintes imposées par l'environnement pour aider au placement manuel, ainsi que quelques méthodes de placement simultané de plusieurs composants.

Chapitre 2

Définition du problème et modèle géométrique

2.1 Placement d'antennes sur une plate-forme

Le placement d'antennes et d'instruments d'observation sur les parois extérieures d'un satellite est un problème extrêmement contraint, car les satellites doivent être aussi compacts que possibles.

En général, une antenne est composée d'un ou plusieurs réflecteurs paraboliques et d'une source qui émet ou reçoit un signal, ainsi que de leurs structures supports. Elle a également un champ de vision (désormais noté CDV), qui a grossièrement la forme d'un cône. Le CDV ne doit pas être occulté, même partiellement. Voir la figure 2.1 pour un exemple d'antenne de télécommunication.

L'aménagement est effectué en ajoutant les instruments un par un sur la structure. A chaque étape, le problème consiste à placer un instrument dans un environnement polyédrique. Nous étudions uniquement des antennes non-mobiles. Une antenne mobile peut être prise en compte en calculant le volume de balayage lorsque l'antenne tourne autour de ses axes de rotation.

La configuration d'une antenne est déterminée par un vecteur tridimensionnel indiquant la position d'un point de référence de l'antenne et par un angle donnant la rotation autour de l'axe du réflecteur parabolique, qui est la direction de vision et est nécessairement fixe. Le placement a donc a priori quatre degrés de liberté.

Cependant, la rotation et la composante verticale du vecteur translationnel de la configuration sont moins importantes. En général, une discrétisation grossière de ces deux degrés de liberté suffit. En pratique, effectivement, la rotation d'une antenne

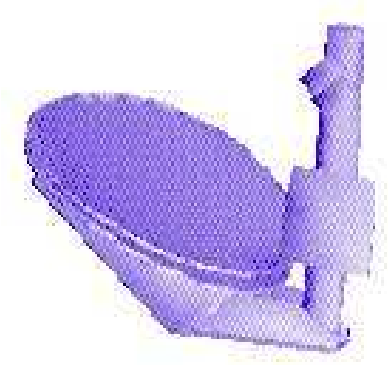


Figure 2.1: Une antenne de télécommunication.

permettant un bon placement est souvent connue par l'opérateur. Une translation dans la direction orthogonale à la paroi est considérée uniquement lorsque le placement de l'instrument sur la paroi s'avère impossible. Dans ce cas, un placement possible est cherché en s'éloignant progressivement de la paroi : à chaque pas, le placement en translation dans le plan correspondant est considéré.

A chaque étape, nous considérons donc le placement d'un instrument en translation dans un plan.

2.2 Polygones et polyèdres

La *somme de Minkowski* [GRS83] de deux ensembles A et B est définie comme

$$A \oplus B = \{p_a + p_b \mid p_a \in A, p_b \in B\}.$$

La somme de Minkowski est distributive et commutative sur l'union, donc pour trois ensembles A, B et C,

$$(A \cup B) \oplus C = C \oplus (A \cup B) = (A \oplus C) \cup (B \oplus C).$$

La *différence de Minkowski* $A \ominus B$ est définie comme $A \oplus (\ominus B)$, où $\ominus B$ est le symétrique de B.

2.2.1 Sommes de Minkowski de polygones convexes

Soient A et B deux polygones convexes de taille m et n respectivement. Supposons que A et B sont en position générale : aucune arête de A est parallèle avec une arête de B.

Soient v_A un sommet de A et $e_B = [v_B, v'_B]$ une arête de B. Soit l la droite qui porte e_B et l^+ le demi-plan défini par l et qui ne contient pas l'intérieur de B.

Soit t un vecteur tel que $l + t$ contienne v_A. $[v_A + v_B, v_A + v'_B]$ est une arête de $A \oplus B$ si et seulement si l'intersection entre $l^+ + t$ et l'intérieur de A est vide (voir figure 2.2).

Soient e_A, e'_A les arêtes incidentes à v_A dans l'ordre croissant d'angle polaire de leurs normales. L'intersection entre $l^+ + t$ et l'intérieur de A est vide si et seulement si la suite des angles polaires des normales à e_A, e_B, e'_A est croissante. Evidemment, ceci est aussi valable pour le cas symétrique d'une arête de A et un sommet de B.

Avnaim démontre comment cette observation est utilisée pour construire $A \oplus B$ en temps $O(m + n)$, en fusionnant les suites des normales des arêtes de A et de B ordonnées selon leurs angles polaires [Avn89].

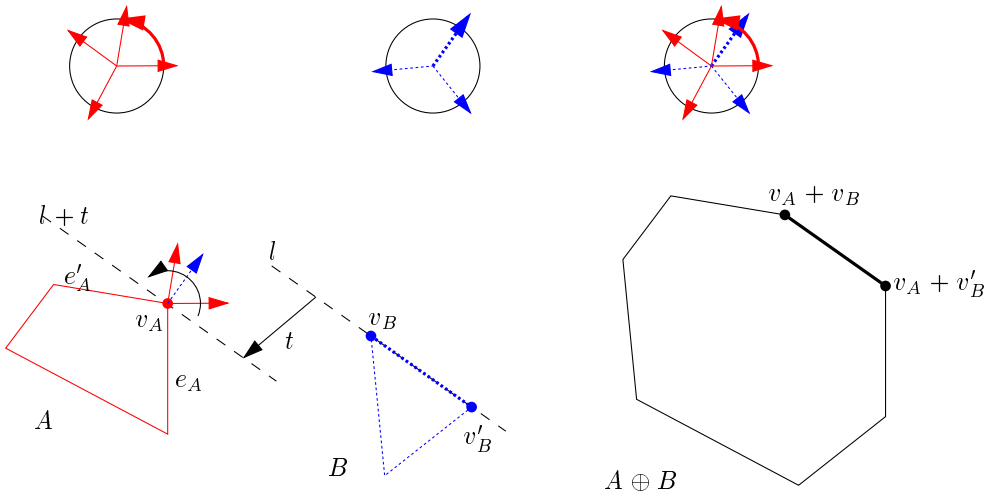


Figure 2.2: Caractérisation de la somme de Minkowski de deux polygones convexes.

2.2.2 Diagrammes gaussiens de polyèdres convexes

Considérons le polyèdre convexe P défini dans un repère orthogonal (O, X, Y, Z). Sur la sphère unité de $\mathbb{R}^3$ notée S^2, nous définissons dans ce paragraphe le *diagramme gaussien* $G(P)$ de P [LP83, GS87, AB87, AB89]. Le diagramme gaussien est utilisé pour calculer la somme de Minkowski de deux polyèdres convexes et pour démontrer la complexité de la somme dans le cas le pire.

Sur S^2, la direction *verticale* est déterminée par l'axe Z. Un plan parallèle à ce vecteur est dit *vertical* et un plan perpendiculaire à la direction verticale est dit *horizontal*.

Nous définissons le vecteur normal d'une face comme le vecteur unitaire orthogonal à la face et pointant vers l'extérieur de P. Le vecteur normal d'un plan support de P est défini comme le vecteur unitaire orthogonal à ce plan et qui pointe vers l'extérieur de P.

Le diagramme gaussien $G(P)$ sur S^2 est formé des éléments suivants (nous utilisons intentionnellement une terminologie différente pour éviter toute confusion entre éléments de P et éléments de $G(P)$) :

- Ses *nœuds* sont les vecteurs normaux des faces de P. Un nœud $n \in G(P)$ est dit *associé à* la face correspondante $f \in P$.
- Ses *arcs* sont des arcs de grand cercle, chacun étant le lieu des vecteurs normaux de tous les plans support de P en une arête. Un arc $a \in G(P)$ est dit *associé à* l'arête correspondante $e \in P$.
- Ses *cellules* sont des régions, chacune étant le lieu des vecteurs normaux de tous les plans support de P en un sommet. Une cellule $c \in G(P)$ est dite *associée au* sommet correspondant $v \in P$.

Etant donné un arc a, nous définissons $Pl(a)$ comme le plan qui contient a. Deux arcs a_1, a_2 sont coplanaires lorsque $Pl(a_1) = Pl(a_2)$. Un plan qui contient un arc contient toujours le centre de S^2.

Il y a une correspondance bijective entre les éléments de P et ceux de $G(P)$ (voir figure 2.3) pourvu que P n'ait pas de faces coplanaires.

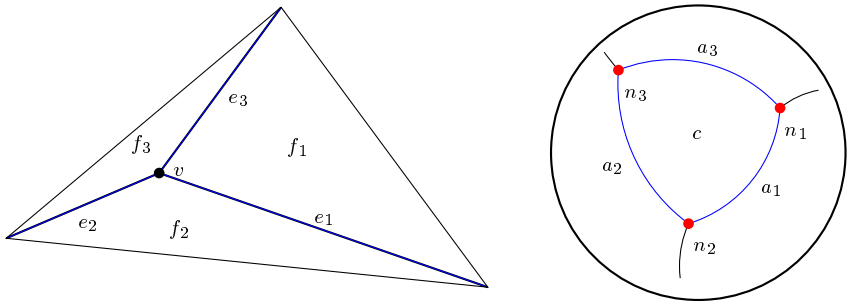


Figure 2.3: Un polyèdre convexe et son diagramme gaussien.

Lorsqu'un polyèdre convexe a des faces coplanaires, les nœuds associés à de telles faces coïncident et les arcs entre ces nœuds sont réduits à des points. Nous pourrions

éviter ce cas dégénéré en fusionnant les faces coplanaires dans un prétraitement du polyèdre.

Cependant, afin d'éviter un travail inutile sur des parties du polyèdre qui ne sont jamais visitées, nous préférons traiter le cas des faces coplanaires au moment où nous le rencontrons.

Nous supposerons dans un premier temps que les polyèdres convexes considérés n'ont pas de faces coplanaires ; on regardera ce cas dans la section 3.5.1.

2.2.3 Sommes de Minkowski de polyèdres convexes

Soient A et B deux polyèdres convexes et $G(M)$ la superposition de leurs diagrammes gaussiens. Il est connu que $G(M)$ est le diagramme gaussien de $M = A \oplus B$ [LP83, GS87]. Supposons que A et B sont en position générale : aucun nœud de $G(A)$ ne coïncide avec un nœud de $G(B)$, aucun arc de $G(A)$ ne contient un nœud de $G(B)$ et inversement.

Chaque face de M correspond alors à un nœud de $G(A)$, un nœud de $G(B)$ ou l'intersection d'un arc $a_A \in G(A)$ avec un arc $a_B \in G(B)$. Ce dernier type de face est parallèle à un plan support de A à l'arête associée avec a_A et parallèle à un plan support de B à l'arête associée avec a_B (voir figure 2.4).

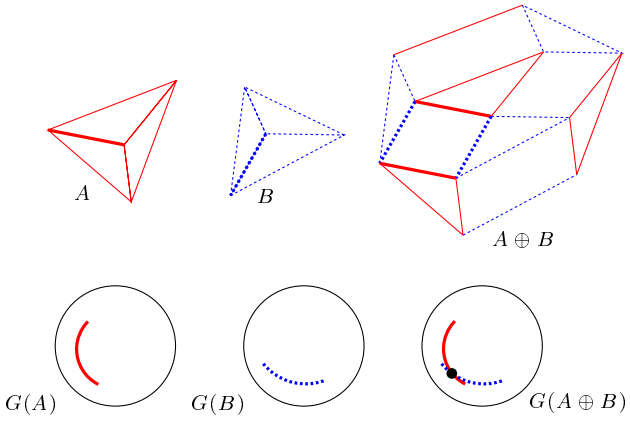


Figure 2.4: L'intersection de deux arcs des diagrammes gaussiens correspond à une face de la somme de Minkowski.

A et B étant en position générale, on distingue donc trois types de nœuds de $G(M)$ associés à des faces de M. Chaque nœud est uniquement défini par un élément de $G(A)$ et un élément de $G(B)$:

- Un nœud $n_A \in G(A)$ est contenu dans une cellule $c_B \in G(B)$. Nous appelons ceci un *nc*-nœud.

- Symétriquement, un nœud $n_B \in G(B)$ contenu dans une cellule $c_A \in G(A)$ est appelé un *cn*-nœud.

- L'intersection d'un arc $a_A \in G(A)$ avec un arc $a_B \in G(B)$ est un *aa*-nœud.

Il est bien connu que, étant donnés les éléments de A et de B associés avec les deux éléments définissant un nœud $n_M \in G(M)$, nous pouvons construire la face associée $f_M \in M$: elle est la somme de Minkowski de ces deux éléments de A et de B. Par exemple, la face de M associée avec un *aa*-nœud $n_M \in G(M)$ est le parallélogramme égal à la somme de Minkowski des deux arêtes de A et de B associées avec les deux arcs de $G(A)$ et de $G(B)$ qui se coupent dans n_M.

Lorsque A et B ne sont pas en position générale, on peut rencontrer les quatre cas dégénérés suivants :

- Le nœud $n_A \in G(A)$ est contenu dans l'arc $a_B \in G(B)$. Nous l'appelons un *na*-nœud.

- Symétriquement, un nœud $n_B \in G(B)$ contenu dans l'arc $a_A \in G(A)$ définit un *an*-nœud.

- Lorsque deux nœuds $n_A \in G(A)$ et $n_B \in G(B)$ coïncident, ils définissent un *nn*-nœud.

- Les arcs $a_A \in G(A)$ et $a_B \in G(B)$ se recouvrent.

2.3 Modèle géométrique

Nous considérons ici le placement d'un polyèdre et d'un groupe de polyèdres sur un plan, en translation dans un environnement composé d'obstacles polyédriques.

Soient E et P deux polyèdres et t un vecteur. La translation de P suivant t mettra P en collision avec E si et seulement s'il existe deux points $p_e \in E$ et $p_p \in P$ tels que $p_e = p_p + t$, donc $t = p_e - p_p$. En conséquence, l'ensemble de toutes les translations pour que P soit en collision avec E est donné par $\{p_e - p_p \,|\, p_e \in E, p_p \in P\}$, ce qui est exactement la différence de Minkowski $E \ominus P$. La frontière de $E \ominus P$ représente toutes les translations qui mettent P *en contact* avec E sans que leurs intérieurs se coupent, tandis que le complémentaire de $E \ominus P$ représente l'ensemble des translations *libres* pour P.

L'espace admissible est défini comme l'ensemble des translations libres et des translations qui mettent l'objet en contact avec l'environnement. On place donc P en choisissant une translation admissible $t \in (E \ominus P)^C$. Les contacts entre objets étant permis, la frontière fait partie de l'espace admissible. Si l'on place un ensemble $\mathcal{P} = \{P \in \mathcal{P}\}$ de polyèdres dans un environnement composé de l'ensemble $\mathcal{E} = \{E \in \mathcal{E}\}$ d'obstacles polyédriques, on calcule $(\mathcal{E} \ominus \mathcal{P})^C$, où $\mathcal{E} \ominus \mathcal{P}$ est définie par $(\cup_{E\in\mathcal{E}} E) \ominus (\cup_{P\in\mathcal{P}} P)$ (voir figure 2.5). En utilisant la distributivité de la somme de Minkowski sur l'union, cela revient à $\cup_{E\in\mathcal{E},P\in\mathcal{P}}(E \ominus P)$.

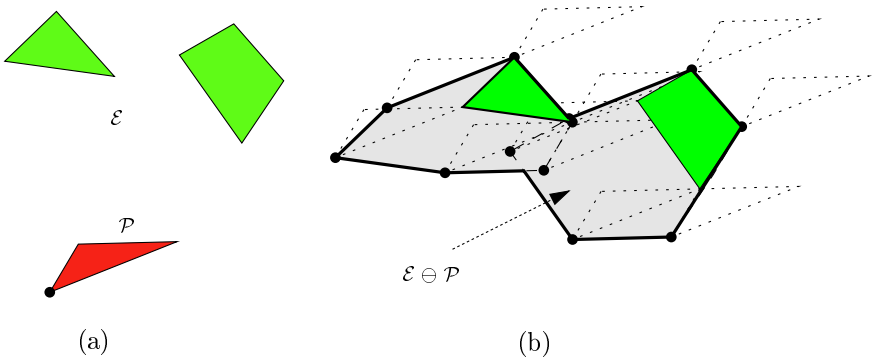


Figure 2.5: (a) La région polygonale $\mathcal{P}$ (composé d'un polygone) et l'environnement polygonal $\mathcal{E}$. (b) L'espace admissible est le complémentaire de la région $\mathcal{E} \ominus \mathcal{P}$ dessinée en gris clair.

Considérons maintenant le placement en translation d'un équipement sur une paroi d'un satellite. Supposons sans perdre en généralité que cette paroi est *horizontale* et définissons un repère (O, X, Y, Z) où l'axe Z est perpendiculaire à la paroi et les axes X et Y ainsi que O sont contenus dans la paroi. Un plan parallèle à l'axe Z est dit *vertical*. On veut positionner l'équipement $\mathcal{P}$ à une hauteur donnée z, donc on s'intéresse seulement aux translations qui positionnent $\mathcal{P}$ à cette hauteur. Toutes ces translations sont contenues dans un plan horizontal $\mathcal{Z}$ dans l'espace tridimensionnel qui représente toutes les translations de $\mathcal{P}$.

Les pièces des équipements, les CDV et la plate-forme sont modélisés comme des unions de polyèdres convexes. $\mathcal{E}$ est l'environnement, défini par la structure du satellite plus les équipements déjà placés. $\mathcal{P}$ est l'équipement à placer et $\mathcal{A}(\mathcal{E}, \mathcal{P})$ désigne l'ensemble de toutes les translations permises (l'espace admissible) qui placent $\mathcal{P}$ sur la plate-forme sans collision ou en contact avec $\mathcal{E}$. Soit $\mathcal{S}$ un équipement ou un obstacle. Les unions de toutes les pièces physiques de $\mathcal{S}$ et de tous ses CDV sont notées $\mathcal{S}_P$ et $\mathcal{S}_F$ respectivement.

Des pièces d'équipements différents ne peuvent pas se percer et un CDV ne peut percer aucune pièce (on dit que deux pièces sont en collision lorsqu'elles se percent). Deux CDV peuvent se croiser librement. Prenant en compte uniquement les pièces qui peuvent interférer, l'espace admissible $\mathcal{A}(\mathcal{E},\mathcal{P})$ pour l'équipement $\mathcal{P}$ sur la paroi en translation dans l'environnement $\mathcal{E}$ est bidimensionnel et est donné par :

$$\mathcal{A}(\mathcal{E},\mathcal{P}) = \left(\ (\mathcal{E} \ominus \mathcal{P}_P) \cup (\mathcal{E}_P \ominus \mathcal{P}_F) \ \right)^C \cap \mathcal{Z}. \tag{2.1}$$

$\mathcal{A}(\mathcal{E},\mathcal{P})$ contient sa frontière, qui représente les placements de $\mathcal{P}$ en contact avec $\mathcal{E}$. $\mathcal{A}(\mathcal{E},\mathcal{P})$ est égal au complémentaire de l'union de $(E \ominus P) \cap \mathcal{Z}$ pour toutes les paires $(E,P) \in (\mathcal{E} \times \mathcal{P}_P) \cup (\mathcal{E}_P \times \mathcal{P}_F)$. Voir la figure 2.6 pour un exemple de placement d'un équipement.

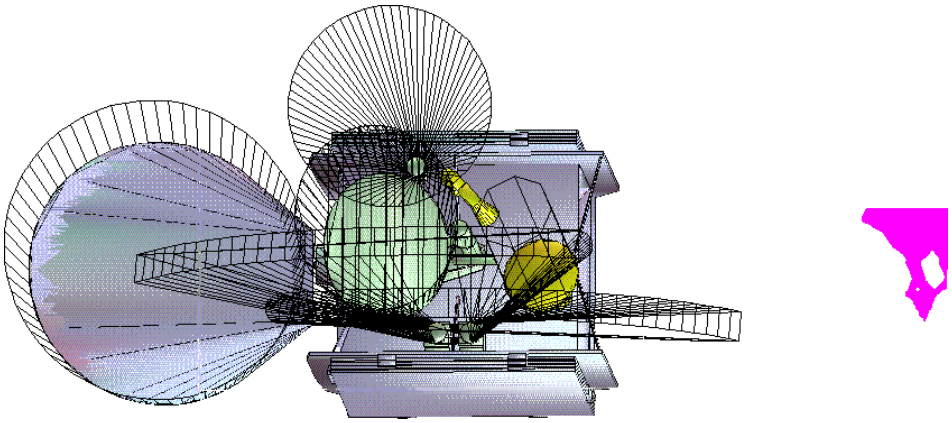


Figure 2.6: Exemple d'un placement d'équipement sur un satellite. L'équipement jaune est placé, $\mathcal{A}(\mathcal{E},\mathcal{P})$ est visualisé à droite.

En cas de dégénérescences, il est possible qu'une composante connexe de la frontière de $\mathcal{A}(\mathcal{E},\mathcal{P})$ ne puisse pas être décrite par un polygone simple :

- Un triple contact de $\mathcal{P}$ dans $\mathcal{E}$ qui immobilise $\mathcal{P}$ correspond à un point isolé dans l'espace admissible.

- Un double contact dégénéré ($\mathcal{P}$ peut être déplacé suivant une droite en restant en double contact avec $\mathcal{E}$) correspond à un passage d'épaisseur 0 (représenté par 2 arêtes qui se recouvrent) dans l'espace admissible.

Voir la figure 2.7 pour des exemples de placement dégénérés.

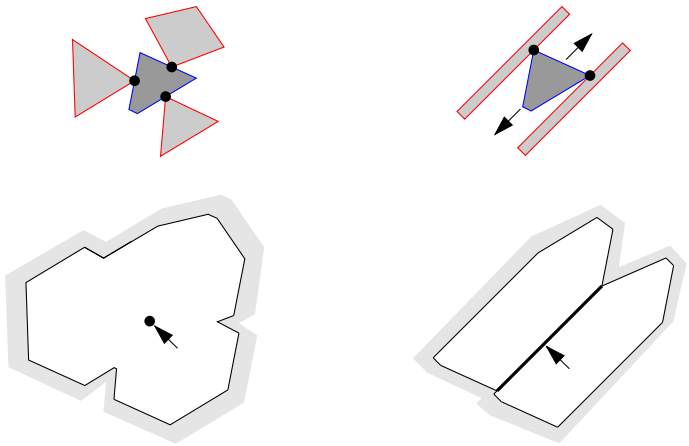


Figure 2.7: Exemples de placement en triple contact (à gauche) et en double contact dégénéré (à droite). En haut, le placement de l'objet foncé. En bas, son espace admissible.

Nous avons conçu un algorithme optimal qui calcule directement l'intersection d'une somme de Minkowski avec $\mathcal{Z}$ sans calculer la somme elle-même ; cet algorithme sera présenté dans le chapitre 3. Le chapitre 4 discute un algorithme efficace pour calculer l'union d'un ensemble de polygones. Le chapitre 5 revient au problème de placement d'équipements.

2.4 Tests et précision numérique

Un problème souvent rencontré dans l'implémentation d'algorithmes géométriques est le manque de précision numérique : les algorithmes géométriques sont pour la plupart basés sur le modèle de calcul exact et sont sensibles aux erreurs numériques causées par l'arithmétique de précision finie qui est réellement disponible.

Cette section présente les outils que nous utilisons pour mesurer la précision nécessaire pour évaluer correctement des tests géométriques, introduit le test **WhichSide** général qui est utilisé par nos algorithmes géométriques et présente la méthode d'Avnaim et al. pour évaluer le signe exact d'un déterminant [ABD+97], qui augmente la précision permise des entrées d'un test géométrique. Les définitions de ce chapitre sont basées sur celles de Liotta, Preparata et Tamassia [LPT96] et Boissonnat et Preparata [BP97].

2.4.1 Degré algébrique des tests géométriques

Afin de rendre nos algorithmes numériquement robustes, nous analysons le niveau de précision nécessaire pour évaluer exactement les prédicats. Nous examinons le *degré algébrique* d'un algorithme.

Un *prédicat élémentaire* est le signe $(+, -, 0)$ d'un polynôme homogène à plusieurs variables prenant comme arguments un sous-ensemble des variables d'entrée. Le *degré d'un prédicat élémentaire* est défini comme le degré maximal des facteurs irréductibles du polynôme correspondant et qui n'ont pas de signe constant. Un *prédicat* ou *test* est plus généralement une combinaison booléenne de prédicats élémentaires ; son degré est le degré maximal de ses prédicats élémentaires.

Le *degré d'un algorithme* A est défini comme le degré maximal de ses tests.

Le *degré d'un problème* P est défini comme le degré minimal des algorithmes qui résolvent P.

Les entrées sont des entiers représentés en b bits. Alors, si un polynôme est de degré algébrique d (et si son nombre de variables est constant), sa valeur peut être représentée en $\beta(P) = bd + c$ bits, où c est une constante.

2.4.2 Le test Which-Side

Considérons l'espace euclidien de dimension 2. Soient p_0, p_1, p_2 trois points où $p_i = (x_i, y_i)$. p_0, p_1 définissent la droite $ax + by = c$ orientée portant le vecteur p_0p_1, où

$$a = y_1 - y_0, b = x_0 - x_1, c = x_0y_1 - x_1y_0.$$

p_2 est respectivement *sur*, *à gauche de* et *à droite de* la droite lorsque $ax_2 + by_2 = c$, $ax_2 + by_2 < c$ et $ax_2 + by_2 > c$. Déterminer la position de p_2 par rapport à la droite définie par p_0, p_1 revient donc à évaluer le signe de

$$(y_1 - y_0)x_2 + (x_0 - x_1)y_2 - (x_0y_1 - x_1y_0),$$

ce qui est égal à :

$$\begin{vmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} x_0 - x_1 & x_2 - x_1 \\ y_0 - y_1 & y_2 - y_1 \end{vmatrix}.$$

Nous utiliserons les notations **Which-Side**(p_0, p_1, p_2) et **Which-Side**(l, p_2), où l est la droite passant par les points p_0, p_1. Le test **Which-Side** de dimension 2 est de degré 2.

De la même façon, le test **Which-Side** est défini dans l'espace euclidien de dimension 3 pour le plan $ax + by + cz = d$ défini par les points p_0, p_1, p_2, et le point p_3,

où $p_i = (x_i, y_i, z_i)$. Le test **Which-Side** de dimension 3 est de degré 3 et est le signe de

$$\begin{vmatrix} x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \\ z_0 & z_1 & z_2 & z_3 \\ 1 & 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} x_1 - x_0 & x_2 - x_0 & x_3 - x_0 \\ y_1 - y_0 & y_2 - y_0 & y_3 - y_0 \\ z_1 - z_0 & z_2 - z_0 & z_3 - z_0 \end{vmatrix}.$$

Nous utilisons les notations **Which-Side**(p_0, p_1, p_2, p_3) et **Which-Side**(P, p_3), où P est le plan qui contient les points p_0, p_1, p_2.

2.4.3 Déterminer le signe de déterminants

Comme le test **Which-Side** de la section précédente, la plupart des tests qui apparaîssent dans des algorithmes géométriques sont formulables en terme de signe d'un déterminant. Avnaim et al. proposent une méthode qui détermine le signe exact de déterminants 2×2 et 3×3 [ABD+97] sans calculer le déterminant même. La méthode prend des entrées entières. D'autres méthodes similaires sont l'arithmétique modulaire [BEPP97], la méthode proposée par Clarkson [Cla92] et la méthode de Shewchuk qui prend comme entrées des flottants [She97].

Lorsque les éléments de la matrice sont représentées sur e bits, la méthode peut évaluer le signe du déterminant en n'utilisant que $e + 2$ bits. Concrètement, si l'on utilise des variables flottantes de double précision du standard IEEE 754 (qui permet de représenter des entiers de 53 bits) et si les entrées apparaissant dans un déterminant nécessitent au plus $bd + c$ bits, une précision de $\lfloor (53 - 2 - c)/d \rfloor$ bits pour les entrées du test permet une évaluation exacte du signe du déterminant.

La méthode d'évaluation exacte de signe de déterminant prend plus de temps que le calcul du déterminant en arithmétique standard mais est néanmoins plus rapide que des méthodes qui calculent la valeur exacte du déterminant. En plus, l'utilisation de filtres numériques permet d'éviter son utilisation dans les cas où l'on sait que l'arithmétique standard fournira la bonne réponse. De bons filtres accélèrent beaucoup les calculs [FV93, MN94, DP97].

Chapitre 3

Sections de sommes de Minkowski

(Une version résumée de ce chapitre a été présentée au 13th Annual Symposium on Computational Geometry [BdLT97] et sera publiée sous le titre "Slicing Minkowski Sums for Satellite Antenna Layout" dans le *Journal of Computer-Aided Design* [BdLT98]).

3.1 Introduction

In chapter 2, we introduced our method for solving the layout problem of antennas on a telecommunication satellite. The method uses planar sections of Minkowski sums of polyhedra.

The size of the three-dimensional Minkowski sum is usually much greater than the size of the desired planar section (which we denote K_s), even though both their worst-case complexities are quadratic (Section 3.3). We present in Section 3.4 an algorithm that directly calculates this section in optimal output-sensitive $O(n + m + K_s)$ time, thus having a better complexity than an algorithm that calculates the entire Minkowski sum to determine the section. This is especially relevant when the polyhedra that are handled are very finely polygonalized approximations of curved shapes.

We also discuss how to handle the degeneracies (Section 3.5) and the numerical precision problems (Section 3.6) encountered in this algorithm. Finally, we present in Section 3.7 some experimental results that support our opinion that our algorithm is much more efficient in practice than a method that first calculates the entire Minkowski sum in $\mathbb{R}^3$ and then determines the wanted planar section.

In the sequel of this chapter, A and B are the convex polyhedra of whose Minkowski sum we calculate a section. n and m respectively denote the size of A and B. K

denotes the size of $A \oplus B$, whereas K_s denotes the size of the intersection $(A \oplus B) \cap \mathcal{Z}$ with a given horizontal plane $\mathcal{Z}$.

3.2 Planar sections of Minkowski sums

In this section, we characterize the intersection of the Minkowski sum $M = A \oplus B$ of two convex polyhedra A and B with a horizontal plane $\mathcal{Z}$. Since both M and $\mathcal{Z}$ are convex, $M \cap \mathcal{Z}$ is also convex. Suppose that no edge or facet of M is contained in $\mathcal{Z}$. Then, the intersections of a facet and an edge of M with $\mathcal{Z}$ correspond to an edge and a vertex of $M \cap \mathcal{Z}$ respectively.

The normal direction of an edge of $M \cap \mathcal{Z}$ contains the projection onto $\mathcal{Z}$ of the normal vector of the corresponding facet of M. Similarly, given a vertex v of $M \cap \mathcal{Z}$ and any supporting plane P of the corresponding edge of M, the projection onto $\mathcal{Z}$ of the normal vector of P is contained in the normal direction of a supporting line of $M \cap \mathcal{Z}$ at v.

The *azimuth* of a vector $\vec{v}$ is the polar angle of the projection of $\vec{v}$ in the horizontal plane. As $M \cap \mathcal{Z}$ is convex, the polar angles of the normal vectors of the sequence of supporting lines of $M \cap \mathcal{Z}$ form a monotonically increasing sequence.

This implies that, if no vertex of M is contained in $\mathcal{Z}$, $M \cap \mathcal{Z}$ is represented on $G(M)$ as a closed path π visiting arcs and nodes in the superposition of the Gaussian diagrams of A and B. π projects onto the horizontal plane exactly onto the sequence of normal directions we examined above. As π's sequence of polar angles is monotonic, π is monotonic with respect to the azimuth.

π is *strictly* monotonic: suppose that the walk visits an arc a contained in a vertical plane. a is associated with a horizontal edge e of M. When e is not contained in $\mathcal{Z}$, only one of the two facets f_1, f_2 incident to e can cut $\mathcal{Z}$. Consequently, a cannot be on π. When e is contained in $\mathcal{Z}$, we replace π by its upper envelope, which is strictly monotonic.

3.3 Combinatorics

3.3.1 Worst-case size of a section of a Minkowski sum

We prove a tight bound for the worst-case size of the intersection of a Minkowski sum with a plane.

It is known that, given two convex polyhedra A and B of sizes m and n respectively,

the size K of their Minkowski sum $A \oplus B$ is $\Theta(mn)$ in the worst case. This result is easily verified by considering the superposition of the Gaussian diagrams of the two polyhedra: arcs of $G(A)$ and $G(B)$ can intersect $\Theta(mn)$ times, which means that as many *aa*-facets exist in the Minkowski sum. A new theorem, more relevant for our application, shows that the size K_s of a planar section of $A \oplus B$ has the same asymptotical worst-case size as the size K of $A \oplus B$ itself.

Theorem 3.1 *Given two convex polyhedra A and B, the intersection of $A \oplus B$ with a plane has a size of $\Theta(mn)$ in the worst case.*

Proof It is clear that the upper bound is quadratic, so we only have to show an example that is actually quadratic. We construct a polyhedron A by intersecting two open polyhedral cones having the same vertical axis such that the intersection of their boundaries is a pattern of m zigzag edges. B is formed by taking the convex hull of a very small horizontal segment (the axis of B) and of n points on a small arc of a vertical circle centered at the midpoint of the segment. Then $A \oplus B$ has quadratic size (see figure 3.1).

Let us denote by u (resp. v) the topmost (resp. bottommost) vertex of B, and take the midpoint of the horizontal axis of B as the reference point on B. If any edge $[p, q]$ of the zigzag pattern on A is sufficiently big with respect to B, then point $p + v$ of $A \oplus B$ is above point $q + u$. So all the points of $[p, q] \oplus w$ defined as the sum $p + w$ of the upper endpoint p of $[p, q]$ and a vertex w of B (except the two vertices of the axis of B) are above all the points defined as the sum $q + w$ of the lower endpoint q of $[p, q]$ and w. This implies that a horizontal plane $\mathcal{Z}$ passing between $p + v$ and $q + u$ intersects all the $\Omega(mn)$ edges of the form $[p, q] \oplus w$ of $A \oplus B$. □

3.3.2 Practical linear cases

In practice, even when K is quadratic, K_s is linear. In this subsection, we show that any planar cross-section of the Minkowski sum of two bounded cylinders has linear size, even though K is quadratic in general. This is a case that often occurs in practice, when "thick discs" are used to approximate antenna reflectors. See figure 3.2.

Theorem 3.2 *Given two bounded cylinders A and B with both $O(n)$ facets, any plane cuts at most $O(n)$ facets of $A \oplus B$.*

Proof To prove the theorem, we show that the corresponding path π through $G(A \oplus B)$ visits at most a linear number of nodes. This is done by subdividing

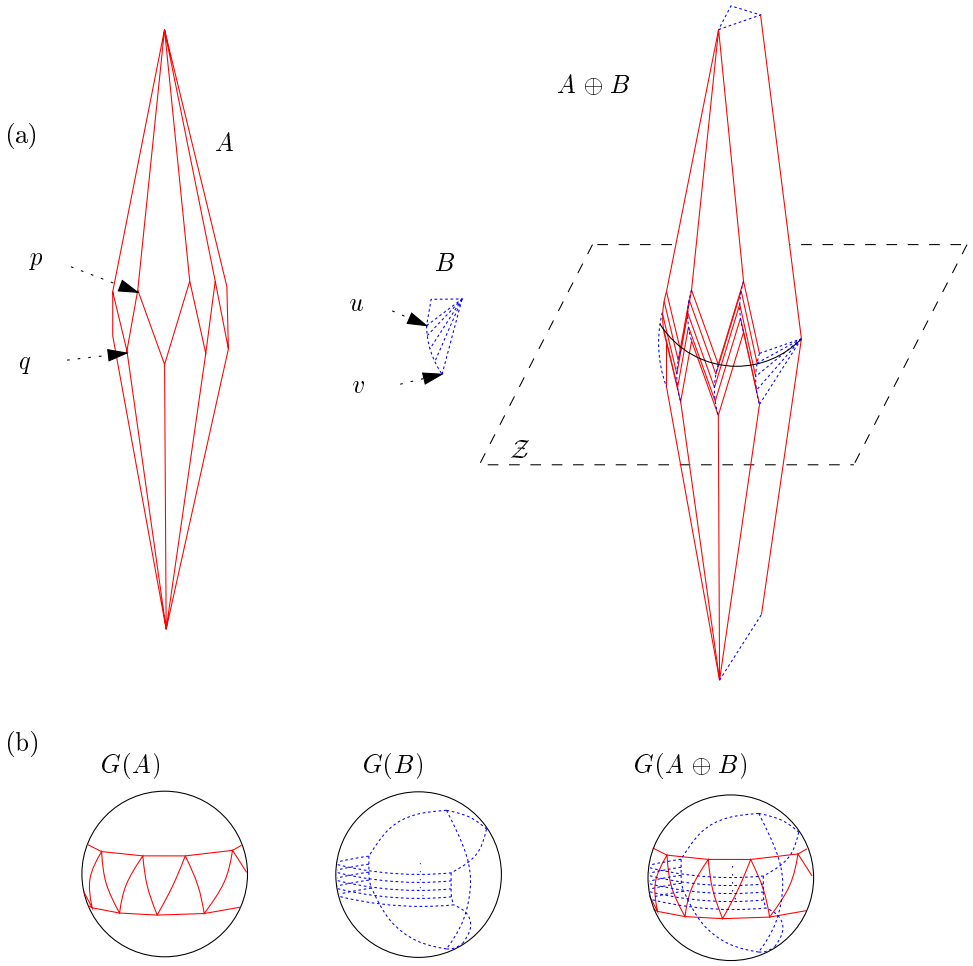


Figure 3.1: An example of a planar cross-section of a Minkowski sum with quadratic size. (a) the polyhedra, (b) their Gaussian diagrams.

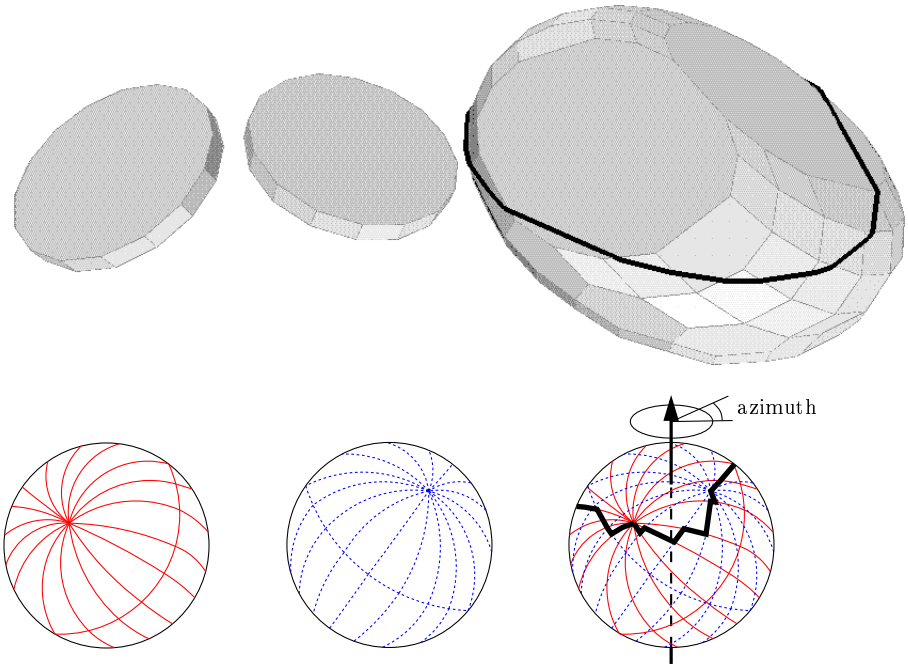


Figure 3.2: A practical case where K is quadratic while K_s is linear.

the Gaussian diagram in four quadrants and proving that for every quadrant, at most a linear number of nodes is visited.

In the Gaussian diagram of a cylinder, $O(n)$ arcs are incident to two nodes associated with the two facets that bound the cylinder. We call these nodes the *star nodes* of the cylinder's Gaussian diagram, and the other nodes the *side nodes*.

Without loss of generality, suppose that the plane that cuts $A \oplus B$ is horizontal. We define P_A and P_B to be the planes containing the vertical axis of S^2 and the star nodes of $G(A)$ and $G(B)$ respectively. P_A and P_B divide $G(A \oplus B)$ into four quadrants when we assume general position; that is, $P_A \neq P_B$ and no star node is contained in the vertical axis. If $P_A = P_B$, the diagram is divided into two hemispheres, and if the star nodes of $G(A)$ (resp. $G(B)$) are vertically aligned, P_A (resp. P_B) is not unique. At the end of the proof, we show that the theorem also holds for these degenerate cases; let us first assume general position.

We consider an individual quadrant Q. We project Q through the center of S^2 onto a vertical plane P_{pr} such that Q converts to a vertical slab of finite width extending towards infinity above and below. See figure 3.3.

By this projection, the portions inside Q of all arcs convert to line segments that are either finite or vertical. Azimuth-monotonicity converts to monotonicity

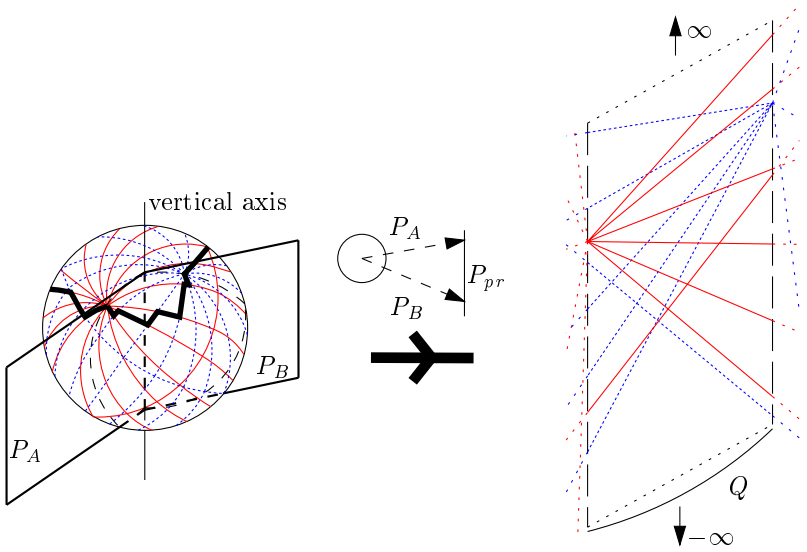


Figure 3.3: Projecting a quadrant of the Gaussian diagram through the center of the unit sphere onto a plane.

with respect to a horizontal line contained in P_{pr}. In the sequel, it is assumed that everything is projected onto P_{pr}.

Let A_* denote the set of all arcs incident to the star nodes of $G(A)$ and $G(B)$. A_* has size $O(n)$. First, we show that a monotonic path ϕ through Q walking only arcs from A_* visits at most $O(n)$ nodes of $G(A \oplus B)$.

Consider figure 3.4. Paths through Q visiting only arcs from A_* are either entirely above the line L_D through the two star nodes or entirely below L_D. Without loss of generality, we only consider the first set of paths.

Let I be the intersection between arcs $a_A \in G(A)$ and $a_B \in G(B)$ that is a node visited by ϕ. To the right of I, ϕ continues by walking either a_A or a_B. Assume without loss of generality that a_A is walked. We decide to charge a_B by I. It is easy to see that an intersection I' of any other arc a'_A with a_B to the right of I can never be reached by ϕ without breaking its monotonicity: the leftmost node where a'_A can be reached by a monotonic path that visits I is to the right of I'. Hence, since ϕ can never get back to a_B to the right of I, only a_B is charged by I. Since this holds for any intersection visited by ϕ, at most $O(n)$ nodes are visited.

Apart from A_*, there are arcs associated with the edges connecting the side facets of the cylinders. Let us call this set $A_\perp$, and consider the monotonic paths that can visit edges from both A_* and $A_\perp$. Notice that $A_\perp$ projects onto two lines on P_{pr}. Consider figure 3.5. $A_\perp$ subdivides Q in at most four cells. At most three of these

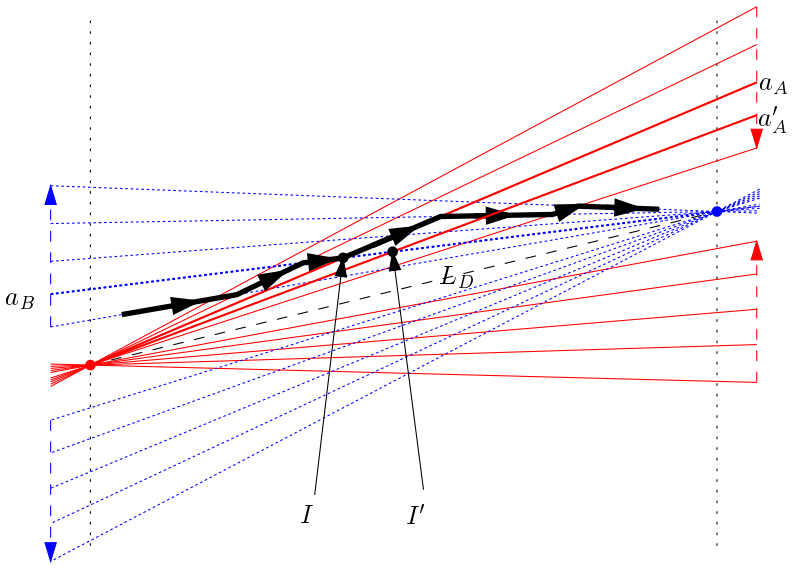


Figure 3.4: Any monotonic path visiting only arcs from A_* is linear.

cells can contain a part of a given arc. Consider one of the cells. The strict interior of the cell contains parts of arcs exclusively from A_*. Hence, for a cell, we use the same charging scheme as for a quadrant containing only A_*.

We charge arcs of A_* by intersections inside cells and we count the total number of intersections on the boundaries of the cells.

It is no longer true that an arc of A_* is charged by a unique intersection: monotonic paths exist that visit an arc $a_A \in A_*$ more than once by walking edges from $A_\perp$. Since the edges of $A_\perp$ form two lines, they cut a_A twice. Consequently, a path can get back to a_A twice, thereby entering another cell. So, any arc is charged by at most three intersections inside three distinct cells. Finally, since the arcs from $A_\perp$ project onto two straight lines, they intersect $O(n)$ times with arcs from A_*.

In Q, at most 3 intersections are charged on any arc from A_* and arcs from A_* intersect $O(n)$ times with arcs from $A_\perp$. Summing up for all four quadrants, we visit at most $O(n)$ nodes.

We come back to the two degenerate cases mentioned at the beginning of the proof. First consider the case where the vertical axis contains the star nodes of one of the diagrams. In that case, all arcs from A_* project to vertical arcs. Since π is strictly monotonic, no such arc can be walked. Hence a vertical arc a_v is charged by only one

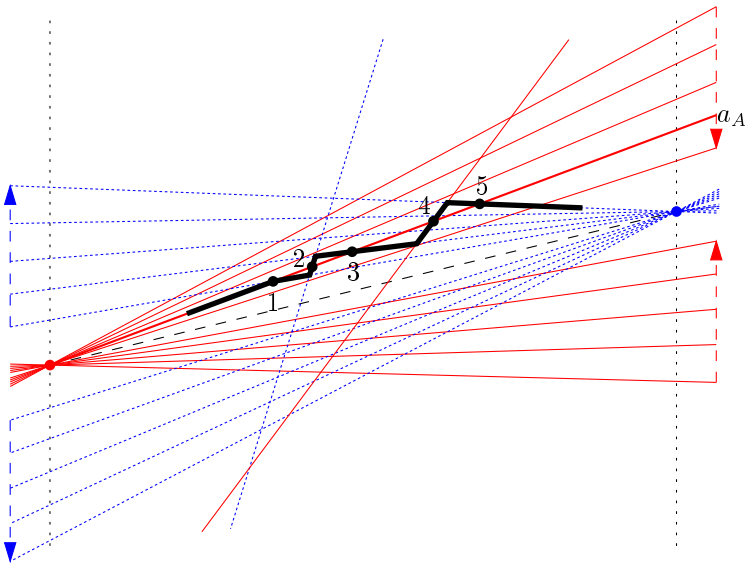


Figure 3.5: Introducing two lines allows a monotonic path to charge at most five intersections on an arc.

visited intersection.

If $P_A = P_B$, the diagram is divided in two hemispheres. P_{pr} is not defined for a hemisphere; both hemispheres project to an *entire plane*, with the star nodes at infinity. Instead of two pencils of lines through two points, we have two pencils of parallel lines (through two points at infinity). It is obvious that the counting argument also holds for this case. □

This theorem shows that K_s is often linear in practice while K is quadratic. We give here some practical results on the sizes of K and K_s. Table 3.1 allows us to compare some numeric values of K and K_s obtained when placing all the antennas on the wall in our model PRISMESAT. For each instrument, the average value of K ranges between three to more than six times the average value of K_s.

3.4 Directly computing a Minkowski slice

We present here an algorithm that directly calculates the intersection of the Minkowski sum of two convex polyhedra A and B with a given horizontal plane $\mathcal{Z}$ by calculating the monotonic path π through the superposition of their Gaussian diagrams (see Section 3.2). We denote $M = A \oplus B$. We assume for the moment that $\mathcal{Z}$ does not contain any vertices of M and that no degeneracies occur in the superposi-

Instrument	N_p	N_v	N_M	K	N_s	K_s
WALL	54	837	-	-	-	-
ESRI-1	3	32	162	30.8	44	11.0
ESRI-2	3	32	171	30.3	46	10.8
HORN-7-HST	7	188	420	51.3	146	11.9
SVS-ANTENNA	10	727	670	144.5	257	22.3
VOGO	10	296	770	91.5	161	17.7
TVGAVO-ENS	4	70	348	62.3	107	19.3
Total	*91*	*2082*	*2541*	*86.8*	*761*	*17.6*

Table 3.1: Characteristics of PRISMESAT. N_p is the number of polyhedra in the instrument and N_v is the total number of vertices in the instrument. N_M is the number of Minkowski sums involved when placing the instrument, N_s the number of Minkowski sums that cut $\mathcal{Z}$, K the average number of vertices of the Minkowski sums, and K_s the average number of vertices of the (nonempty) sections with $\mathcal{Z}$.

tion $G(M)$ of the Gaussian diagrams $G(A)$ and $G(B)$ of A and B. Degeneracies will be examined in Section 3.5.

The path π through $G(M)$ visits the arcs and nodes associated with all edges and facets of M that intersect $\mathcal{Z}$ (and *only* those elements). We don't calculate the entire superposition but, by walking $G(A)$ and $G(B)$ simultaneously, we only walk the arcs and nodes of $G(M)$ associated with elements of the output polygon $(A \oplus B) \cap \mathcal{Z}$ whose size is denoted by K_s.

The scheme of the algorithm that walks $G(M)$ while determining the path π corresponding to $(A \oplus B) \cap \mathcal{Z}$ is the following:

Algorithm MINKOWSKI-SUM-SLICE

---

find an initial arc to start the walk of π (Section 3.4.3)
`repeat`
  walk on the current arc to the next node on π (Section 3.4.2)
  find the next arc on the path (Section 3.4.1)
`until` the initial arc is again reached, which closes π.

As we saw, the fact that the path is monotonic is crucial.

3.4.1 How to get from a node to an arc

Assume that our walk has arrived at a certain node $n_M \in G(M)$, associated with facet $f_M \in M$. Because f_M cuts $\mathcal{Z}$ and is convex, exactly two edges of f_M cut $\mathcal{Z}$, one by whose associated arc we arrived at n_M and one by whose associated arc we will advance to the next node of the Gaussian diagram. Therefore we must determine the unique arc $a_M \in G(M)$ incident to n_M that must be followed in order to get to the next node of $G(M)$ (it is either an arc from $G(A)$ or from $G(B)$). Starting from the arc by which we arrived at $n_M \in G(M)$, we walk the arcs around $n_M \in M$ until we arrive at the searched arc.

Notice however that $G(M)$ does not contain the information needed to determine whether or not an edge associated with an arc incident to $n_M \in M$ cuts $\mathcal{Z}$. This information is obtained by inspecting A and B: the edge of M is the Minkowski sum of an edge of A and a vertex of B (or inversely), so we can compute the z-coordinate of its two endpoints and compare them with the height of $\mathcal{Z}$.

Let us first consider the case when n_M is a *cn*-node. n_M is given by a node of $G(B)$ contained in a cell of $G(A)$. To find the arc that must be followed, we have to test all arcs of $G(B)$ incident to the node, and it can take up to $O(n)$ time to identify the arc that must be walked. Since we know in which cell of $G(A)$ node n_M lies, we know that the arc we will be walking is (at least partially) contained in the cell n_M lies in when we start walking it. The case of an *nc*-node is symmetric and takes $O(m)$ time. We might visit all *cn*- and *nc*-nodes, but since arcs are incident to two nodes and the walk is monotonic, any arc from $G(A)$ or $G(B)$ is at most considered twice (once for both its end-nodes) on the visited *cn*- and *nc*-nodes during the whole walk.

The case of an *aa*-node n_M is easier to analyze since *aa*-nodes are incident to four arcs of $G(M)$ given by one arc of $G(A)$ and one arc of $G(B)$ intersecting at n_M: we examine them all to determine the next arc to walk. If it is contained in an arc of $G(A)$ (resp. $G(B)$), when we start following it, the cell of $G(B)$ (resp. $G(A)$) it lies in is one of the two incident cells of the arc of $G(B)$ (resp. $G(A)$) intersecting it. We visit at most $O(K_s)$ such nodes.

Consequently, $O(m + n + K_s)$ time is spent in total to determine which arc to walk next and in which cell this arc lies.

3.4.2 How to walk an arc to a node

Without loss of generality, suppose that we walk along arc a_M contained in $a_A \in A$ through cell $c_B \in B$. If a_A intersects an arc a_B incident to c_B, the next node n_M is an *aa*-node, defined by the intersection of a_A and a_B. Otherwise, we encounter an *nc*-node n_M, defined by c_B and the endpoint of a_A in the direction of the walk (see

figure 3.6). In both cases, we already know in which cell (or on which cell boundaries) of $G(A)$ or $G(B)$ node n_M lies.

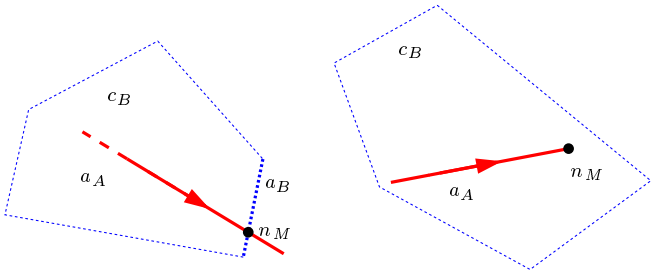


Figure 3.6: Identifying the next node $n_M \in M$. Arc a_A is walked on through cell c_B. (left) an *aa*-node. (right) an *nc*-node.

Let P_s, P_e be the planes that contains the vertical axis and the endpoints of a_A with the smallest and the largest azimuth respectively. Let P_s^+ and P_e^- be the halfspaces defined by P_s and P_e respectively and that contain a_A. We define $P_s^+ \cap P_e^-$ to be the *slab* of a_A (see figure 3.7). An arc a_B can intersect with a_A only if it cuts the slab of a_A.

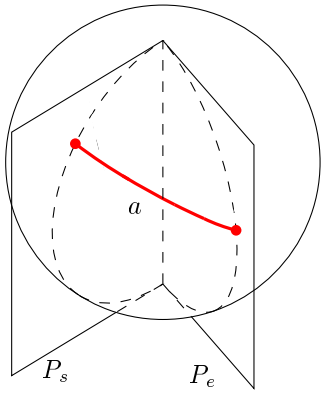


Figure 3.7: Definition of the slab of arc a.

Notice that the boundary of c_B can be decomposed in two subsequences of arcs, both being azimuth-monotonic. Consequently, while following arc a_A, we can follow at the same time the two sequences of arcs of the boundary of c_B that are respectively above and below a_A, until we reach either an intersection between a_A and an arc a_B or the endpoint of a_A.

In case we find an endpoint n_A, we will follow a new arc of $G(A)$ and at the same time keep on traversing, starting from the current arcs, the ordered subsequences of arcs of c_B. See figure 3.8a.

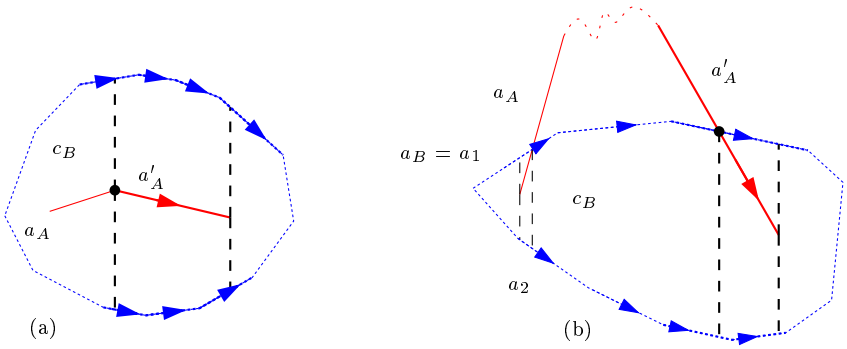


Figure 3.8: Following the boundary of c_B. (a) after an endpoint of a_A, (b) after an intersection between a_A and a_B.

In case we reach an intersection between a_A and a_B we either follow a_B through a cell c'_A and we go on symmetrically, or we keep on following a_A and we leave cell c_B to enter a cell c'_B. In this case, we memorize the arcs a_1 and a_2 of the two subsequences of the boundary of c_B at that point. Suppose that path π re-enters the same cell c_B later with another arc a'_A. Since π is monotonic, we examine the two subsequences, starting from a_1 and a_2, until we find the two arcs of the boundary of c_B that are just above and below a'_A, and then we go on as for a_A. See figure 3.8b.

$O(K_s)$ arcs of $G(M)$ are walked on π and $O(K_s)$ nodes are computed. By following the boundaries of the cells of $G(A)$ and $G(B)$ crossed by π, an arc of $G(A)$ or $G(B)$ can be examined once for each node of $G(M)$ that lies either in one of the two cells incident to this arc, or on the boundary of one of them, in which cases the intersection test can be charged to the cost of constructing the node of $G(M)$. An arc of $G(A)$ or $G(B)$ can in addition be examined at most twice to locate the entry point of π into each of its two incident cells by following their boundaries. So in total, $O(m+n+K_s)$ time is spent to find the nodes of the path.

3.4.3 Finding an initial arc

We determine a starting point for path π by calculating a *shadow* of M. The shadow of M is the boundary of the orthogonal projection of M onto a plane. We consider a vertical plane $\mathcal{V}$ (we still refer to the conventions given at the end of Section 2.3): this ensures that the shadow of M is cut by $\mathcal{Z}$.

As for the parts of the algorithm that were discussed previously, we assume that the polyhedra are in general position. We also assume general position with respect to the shadow: no facet, neither from A or B nor from M is parallel to the normal vector n_v of the shadow plane $\mathcal{V}$. These degeneracies will be discussed in Section 3.5.4.

On $G(M)$, the chosen shadow of M is associated with the sequence of arcs and cells of $G(M)$ cut by the vertical plane $\mathcal{V}'$ defined by two independent vectors of $\mathcal{V}$ and containing the center of S^2.

Let us consider the great circle Γ on S^2 contained in this vertical plane $\mathcal{V}'$. Starting with the cell of $G(A)$ associated with the highest vertex of A (whose associated cell necessarily cuts Γ), we can find the arcs of $G(A)$ intersected by Γ by following the arcs incident to all the cells cut by Γ (see figure 3.9a). In this way, we directly obtain the ordered sequence of the arcs of $G(A)$ cut by Γ. The same process is applied to B. Both arc-lists correspond to a sequence of connected edges from A and B respectively describing the shadows of A and B onto $\mathcal{V}$. The two lists are traversed at the same time, in such a way that the order in which the arcs cut Γ is preserved, so we get an ordered list of arcs of $G(M)$ corresponding to the shadow of M in $O(m+n)$ time.

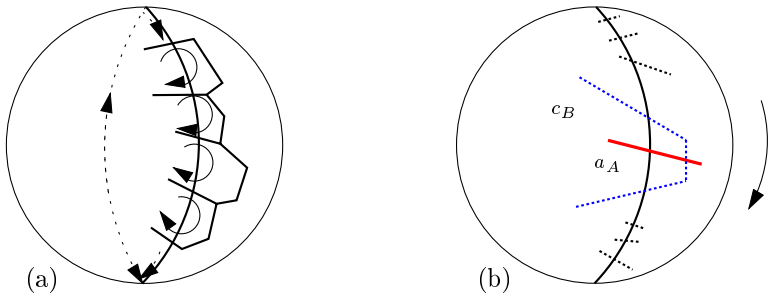


Figure 3.9: Finding an arc to start the walk with. (a) Constructing a sorted shadow-list in linear time. (b) Determining an arc whose associated edge cuts $\mathcal{Z}$.

In fact we can stop as soon as we encounter an arc associated with an edge of M intersected by plane $\mathcal{Z}$: without loss of generality, assume that this arc is an arc $a_A \in G(A)$ associated with edge $[p,q]$ of A (see figure 3.9b). It cuts or is contained in a cell of $G(B)$ associated with a vertex v_B that is on the shadow of B. The edge $[p,q] \oplus v_B$ intersects $\mathcal{Z}$ if and only if its endpoints $p+v_B$ and $q+v_B$ lie on opposite sides of $\mathcal{Z}$.

While traversing the arc-lists, we obviously know through which cells of $G(A)$ and $G(B)$ we are walking. Hence, if the initial arc belongs to $G(A)$ (resp. $G(B)$) we know in which cell of $G(B)$ (resp. $G(A)$) this arc lies at the point we start our walk.

3.4.4 Overall complexity

Summing up, a total time of $O(m + n + K_s)$ is needed to run the algorithm, where K_s is the size of the output polygon. This is better than any algorithm computing the 3D Minkowski sum and then its section by $\mathcal{Z}$, since K_s is at most equal to the size K of the entire Minkowski sum and usually much less.

3.5 Handling degeneracies

We explicitly handle the degeneracies. Nodes of a Gaussian diagram can coincide: this corresponds to coplanar facets in the polyhedron. We show in Section 3.5.1 how these are handled. Furthermore, as stated in Section 2.2.3, the following degeneracies can be encountered when the Gaussian diagrams of A and B are superposed:

- Node $n_A \in A$ is contained in arc $a_B \in B$ and symmetrically, node $n_B \in B$ is contained in arc $a_A \in A$.
- Two nodes $n_A \in A$ and $n_B \in B$ coincide.
- Arcs $a_A \in A$ and $a_B \in B$ overlap.

These degeneracies are described in Section 3.5.2. $\mathcal{Z}$ can contain vertices, edges and facets of $A \oplus B$: this is described in Section 3.5.3. Finally, In Section 3.5.4, we discuss how to handle facets that are perpendicular to the plane onto which the shadow is projected.

3.5.1 Coplanar facets in a polyhedron

A polyhedron might have coplanar facets. Such facets are associated with nodes in the Gaussian diagram that coincide on S^2. They have edges in common that are associated with zero-length arcs. Allowing this degeneracy introduces overlapping facets in the result.

An example: given the two coinciding nodes n_A and n'_A of $G(A)$, that are contained in arc $a_B \in G(B)$. f_A and f'_A are the facets associated with n_A and n'_A respectively, e_A is the edge that f_A and f'_A have in common and e_B the edge associated with a_B. The two facets $f_A \oplus e_B$ and $f'_A \oplus e_B$ overlap when e_A is not parallel to e_B (see figure 3.10).

It is unnecessary to preprocess the polyhedra to eliminate all coplanar facets since a walk through a Gaussian diagram might never encounter them. Instead, we generalize the methods we use to follow the adjacencies in order to handle groups of coplanar facets as one facet.

In fact, we only need a method that, given a non-zero length arc a incident to node n, finds the counterclockwise next non-zero length arc incident to the group of coinciding nodes containing n. This is the operation needed to determine the arc that is visited by π after the *cn*- or *nc*-node (or a degenerate node discussed in Section 3.5.2) it is currently on (Section 3.4.1) and to determine the shadow of a polyhedron (Section 3.4.3).

Notice that when we use a *"doubly-connected edge list"* structure to represent polyhedra, we find the next arc in constant time by inspecting a. We give the algorithm that, with respect to arc a, finds the counterclockwise next non-zero length arc incident to a node n (see also figure 3.11):

Algorithm NEXT-NONZERO-ARC(n, a)

$a' \longleftarrow$ NEXT-ARC(n, a)
`while` a' is a zero-length arc
  $n' \longleftarrow$ the other node incident to a'
  $a' \longleftarrow$ NEXT-ARC(n', a')
`endwhile`
`return` a'

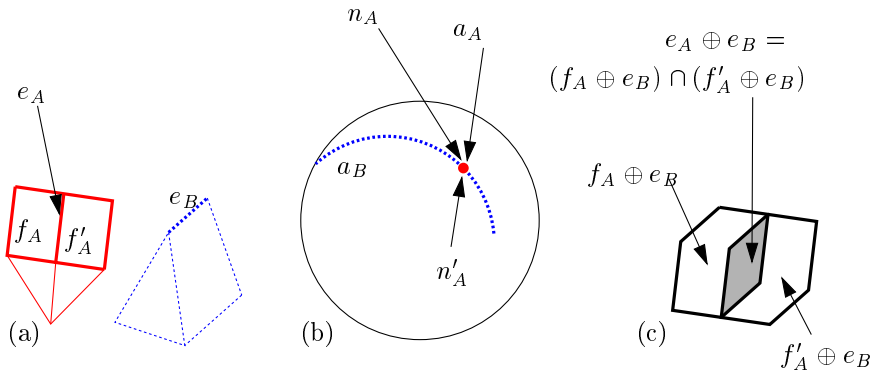


Figure 3.10: Coinciding nodes contained in an arc create overlapping facets. (a) the two polyhedra, with an edge e and two facets f_1, f_2 that are coplanar and have edge e_0 in common, (b) the superposition of their associated elements in the Gaussian diagrams, (c) the three associated overlapping facets: the overlapping part is the Minkowski sum of e and the edge e_0 common to f_1 and f_2.

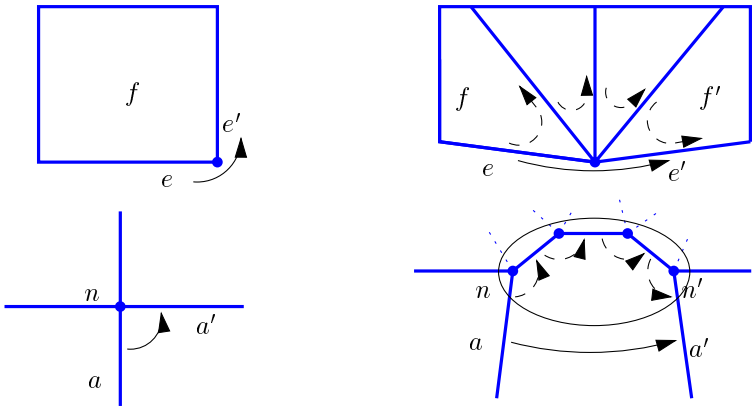


Figure 3.11: NEXT-NONZERO-ARC handles coinciding nodes as if they were one node. Above, the facets are drawn and below their associated nodes in the Gaussian diagram. (left) NEXT-ARC finds the counterclockwise next arc a' from arc a, (right) NEXT-NONZERO-ARC does likewise to find the next non-zero length arc a' from arc a (the nodes inside the ellipse coincide).

Notice that the input arc a must be of non-zero length; otherwise the algorithm might get in an infinite loop. Since, given an arc of non-zero length, NEXT-NONZERO-ARC always outputs an arc of non-zero length, we only have to assure that the initial arc of π is of non-zero length. Since the shadows of the polyhedra are also calculated using NEXT-NONZERO-ARC, the only case in which a zero-length arc can be encountered is when, at the beginning, the horizontal plane supporting the polyhedron at its topmost vertex contains coplanar facets of the polyhedron. In that case, all those facets are inspected in linear time in order to find a facet incident to an edge associated with an arc of non-zero length. That arc is used as the starting point to calculate the shadow.

3.5.2 Superposing Gaussian diagrams

When the Gaussian diagrams of A and B are not in general position, nodes of one diagram can be contained in arcs of the other diagram, nodes from $G(A)$ and $G(B)$ can coincide and arcs from $G(A)$ and $G(B)$ can overlap. This necessitates the following modifications.

Finding the next node. We are on an arc $a_M \in M$ that is either defined by a part of an arc from one diagram contained in a cell of the other diagram, or is the

overlapping part of two arcs from both diagrams.

Consider the first case. Without loss of generality, suppose that arc $a_A \in G(A)$ is walked through cell $c_B \in G(B)$. If an intersection is encountered, we detect whether the intersection coincides with a node incident to a_A and/or to c_B.

In the case of overlapping arcs a_A, a_B, the next node to visit is the first encountered endpoint of the overlapping part in the walking direction of π. This can be an endpoint of a_A contained inside a_B or inversely, or the endpoints of a_A and a_B that coincide.

Finding the next arc. Two cases are distinguished. The first case is where we are on a node n of π defined by the coinciding nodes $n_A \in G(A)$ and $n_B \in G(B)$. To find the next arc to walk, we walk around n inspecting the arcs incident to n_A and n_B in sorted order. While traversing the sorted lists of arcs, it is detected whether arcs e_A, e_B incident to n_A and n_B overlap: this is the case when e_A does not come before e_B and e_B does not come before e_A. If the arc to walk overlaps with an arc of the other diagram, both arcs are walked at the same time.

The second case is that of a node $n_A \in G(A)$ contained in arc $a_B \in G(B)$. We cut a_B in two at the position n_A and proceed as in the case of two coinciding nodes; the same is done for a node $n_B \in G(B)$ contained in an arc $a_A \in G(A)$. The same is done for the symmetric case of a node $n_B \in G(B)$ contained in arc $a_A \in G(A)$.

3.5.3 Vertices and edges in the section

$\mathcal{Z}$ can contain vertices, edges or facets of $M = A \oplus B$. Notice that when $\mathcal{Z}$ contains a facet, it also contains edges, and when it contains an edge, it also contains vertices.

Let us first look at the case where an entire facet f of M is contained in $\mathcal{Z}$. f is necessarily the topmost or bottommost facet of M. Consequently, it is the entire section. Suppose it is the topmost facet of M. Let el_A and el_B be the topmost elements of A and B respectively (both are either a facet, an edge or a vertex). The section of M with $\mathcal{Z}$ is equal to the (planar) Minkowski sum of the projections of el_A and el_B onto $\mathcal{Z}$. The bottommost facet of M is handled in the equivalent way. Notice that top- and bottommost facets of a polyhedron can consist of groups of coplanar facets: it is easy to construct one facet from such a group using the NEXT-NONZERO-ARC method.

When a vertex v of M is contained in $\mathcal{Z}$, the path π through $G(M)$ contains the cell associated with v. When an edge e of M is contained in $\mathcal{Z}$, π contains two cells, separated by the vertical arc associated with e. In both cases, π is not a path (see figure 3.12).

However, consider the facets associated with the nodes on the upper envelope of

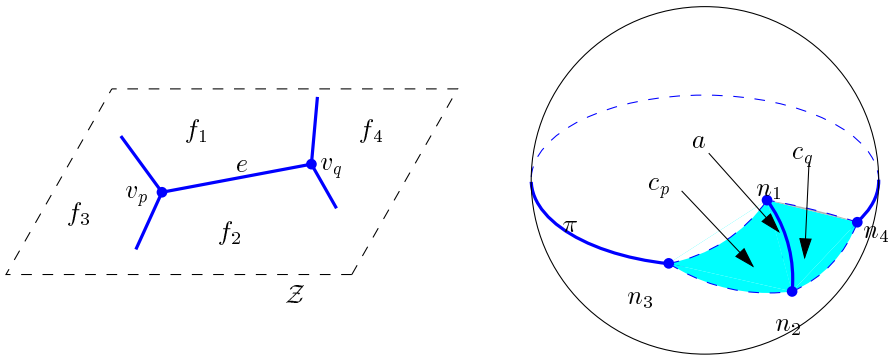


Figure 3.12: The path on $G(A \oplus B)$ is a pseudo-path when an edge or vertex of $A \oplus B$ is contained in $\mathcal{Z}$. Left, the horizontal edge e contained in $\mathcal{Z}$. Right, The pseudo-path π through $G(A \oplus B)$ contains an arc a contained in a vertical plane and two cells c_p, c_q.

that cell. Such a facet f either touches $\mathcal{Z}$ in a point or cuts it (in the latter case, the node associated with f is on π). In both cases, $f \cap \mathcal{Z}$ contains the vertex that is contained in $\mathcal{Z}$. The same holds for an edge e that is contained in $\mathcal{Z}$: such an edge is equal to the intersection with $\mathcal{Z}$ of a facet that is incident to e.

Consequently, the path π' that is the *upper envelope* of a pseudo-path π containing cells is associated with a sequence of facets and edges whose intersection with $\mathcal{Z}$ is equal to $M \cap \mathcal{Z}$, hence we can replace π by π'. In figure 3.12, for example, the sequence n_3, c_p, a, c_q, n_4 in π is replaced by n_3, n_1, n_4.

3.5.4 Degeneracies in the shadow

Nodes in the Gaussian diagram can be contained in the shadow plane. These nodes can be from one of the original diagrams or defined by intersections (strictly between arcs or the degenerate ones that are discussed in the next section). If the shadow plane contains a node, the shadow of the associated polyhedron contains a facet.

Given a facet f associated with the node n in the shadow plane. Instead of visiting n, we walk the arcs around it in the same way as is done for determining the next arc of π to walk (Section 3.4.1). This way, a sequence of arcs is determined that is associated with a connected sequence of edges on the shadow.

3.6 Tests and numerical stability

In this section, we closely look at the tests used and the numerical precision needed to evaluate them correctly. We show that all tests can be formulated in terms of **Which-Side** tests (Section 2.4.2).

3.6.1 Input data

The input of the algorithm are two polyhedra, both defined by a set of vertices with adjacencies describing the edges and facets of the polyhedron. The coordinates of an input vertex p are denoted (x_p, y_p, z_p) and are supposed to be represented by integers.

The normals of the facets of the input polyhedra are not given. We remind the reader that the node n associated with f is the normal of f pointing outwards the polyhedron. It is calculated from three non-collinear vertices on its boundary. If p, q, r are vertices appearing in counterclockwise order (as seen from the outside of the polyhedron) on the boundary of f, n is given by

$$\frac{pq \times qr}{||pq \times qr||} \tag{3.1}$$

However, by calculating this, we would introduce an error. We will see that it is not necessary for our tests to normalize. n is proportional to

$$pq \times qr = \left(\begin{vmatrix} y_q - y_p & z_q - z_p \\ y_r - y_q & z_r - z_q \end{vmatrix}, \begin{vmatrix} x_r - x_q & z_r - z_q \\ x_q - x_p & z_q - z_p \end{vmatrix}, \begin{vmatrix} x_q - x_p & y_q - y_p \\ x_r - x_q & y_r - y_q \end{vmatrix} \right).$$

The components of this vector are polynomials of degree 2 in the input data.

3.6.2 Tests

The following tests are necessary to implement the algorithm. The tests are given in the order they occur in the description of the algorithm. O denotes the center of S^2.

3.6.2.1 Getting from a node to an arc

The walk is currently on a node $n_M \in G(M)$ and has to determine the incident arc by which to continue.

Edge-Cuts-Z The arc of $G(M)$ we must walk is the one that is associated with an edge $[v_1, v_2]$ cutting $\mathcal{Z}$. This edge is defined by two vertices v_1, v_2, both being

the sum of a vertex of A and a vertex of B. It is determined whether v_1 and v_2 are at opposite sides of $\mathcal{Z}$ by **Which-Side**$(\mathcal{Z}, v_1)$ and **Which-Side**$(\mathcal{Z}, v_2)$. Since $\mathcal{Z}$ is a horizontal plane defined by $Z = z$, the test reduces to comparing the z-coordinates of v_1 and v_2 with z and is of degree 1.

Arc-Order If the node we are on is defined by an intersection of two arcs, by a node contained in an arc or by two coinciding nodes, arcs from both diagrams define arcs incident to n_M. We have to visit these arcs of $G(M)$ in the right order around n_M to determine which of them to walk.

We determine the order between two arcs $a_1, a_2 \in G(M)$ in the following way. Let $a_A = [n_A, n'_A] \in G(A)$ and $a_B = [n_B, n'_B] \in G(B)$ be the *oriented* arcs that define a_1, a_2 respectively, and let $a_2 = [n_M, n'_M]$. a_2 is counterclockwise of a_1 with respect to n_M if n'_M is above $Pl(a_1)$ (the plane containing a_1), which is equal to $Pl(a_A)$. Furthermore, since a_A and a_B are oriented, n'_M and n'_B are to the same side of $Pl(a_A)$.

Hence, the order is given by **Which-Side**(O, n_A, n'_A, n'_B) (see figure 3.13). Let $e_A = [p, q]$ be the edge associated with a_A, hence the edge common to the facets associated with n_A and n'_A. Since the two facets have e_A in common, n_A and n'_A have pq in common: this vector is perpendicular to $Pl(a_A)$. The test simplifies to the evaluation of the sign of $pq \times n'_B$. This polynomial is equal to

$$\begin{vmatrix} x_q - x_p & x_s - x_r & x_r - x_t \\ y_q - y_p & y_s - y_r & y_r - y_t \\ z_q - z_p & z_s - z_r & z_r - z_t \end{vmatrix}, \tag{3.2}$$

where r, s, t are vertices of the facet f'_B associated with n'_B. The test is of degree 3. If the sign is 0, a_A and a_B overlap.

Facets-Parallel Two facets f_1, f_2 are parallel if two non-parallel edges $[p, q]$ and $[q, r]$ on the boundary of f_2 are both parallel with f_1. Let n_1 be the normal of f_1. We test whether both $pq \times n_1$ and $qr \times n_1$ are zero, so **Facets-Parallel** is of degree 3 (see equation 3.2).

3.6.2.2 Getting from an arc to a node

The walk is on arc $a_M \in G(M)$ and has to determine the next node of $G(M)$ that will be encountered.

Arcs-Intersection If a_M is not defined by overlapping arcs from $G(A)$ and $G(B)$, we determine whether the arc defining a_M intersects with an arc from the other diagram. Without loss of generality, let $a_A = [n_A, n'_A]$ be the arc defining a_M. a_B traverses cell $c_B \in G(B)$; let $a_B = [n_B, n'_B]$ an arc on the boundary of c_B. We

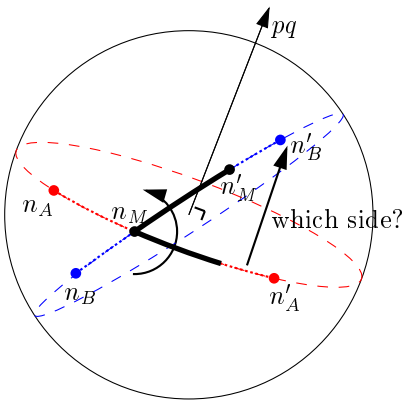


Figure 3.13: The **Arc-Order** test: determining the order between two arcs $a_A = [n_A, n'_A], a_B = [n_B, n'_B]$ around n_M. The great circles containing the arcs are drawn dashed; the vector pq is perpendicular to $Pl(a_A)$.

assume the arcs are oriented in increasing azimuth. We want to test whether a_A and a_B intersect.

We know that the endpoint of a_M that is already visited is not strictly outside c_B: it is either defined by n_A or by the intersection of a_A with another arc on the boundary of c_B.

Consequently, we determine whether a_A and a_B intersect by testing if:

- n_A and n'_A lie to opposite sides of a_B and
- n_B and n'_B lie to opposite sides of a_A.

See figure 3.14. Testing to which side lies a node n of the arc associated with edge $[p, q]$ is done by examining the sign of $pq \times n$. Hence, **Arcs-Intersect** is of degree 3 (equation 3.2).

Node-Order-On-Great-Circle The arc a_M is defined by the overlapping part of the arcs $a_A \in G(A)$ and $a_B \in G(B)$, where n_A and n_B are their respective endpoints with the highest azimuth. We know that the following node to visit is defined by n_A, n_B or both. To determine which one is the case, we determine the order between n_A and n_B on $Pl(a_M)$.

Let f_A be the facet associated with n_A, with vertices p, q, r on its boundary and f_B the facet associated with n_B, with vertices s, t, u on its boundary. Let $[p, q]$ and $[s, t]$ be the edges that are associated with a_A and a_B respectively. Since a_A and a_B overlap, these two edges are parallel. Notice that the vector pq is perpendicular to $Pl(a_A)$.

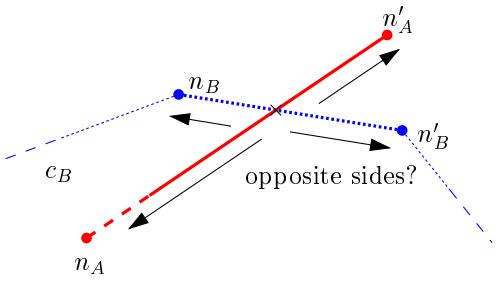


Figure 3.14: Testing whether the arcs $a_1 = [n_1, n'_1]$ and $a_2 = [n_2, n'_2]$ intersect.

Let P be the supporting plane of f_A, and let $P' = P + qt$ be the translated copy of P that contains $[s, t]$. P' contains the three point $s, t, r + qt$. n_A comes before n_B if u is behind P': this is the case when **Which-Side**$(s, t, r + qt, u)$ is negative (see figure 3.15). If u is on P' (the test returns zero), n_A and n_B coincide. **Node-Order-On-Great-Circle** is of degree 3.

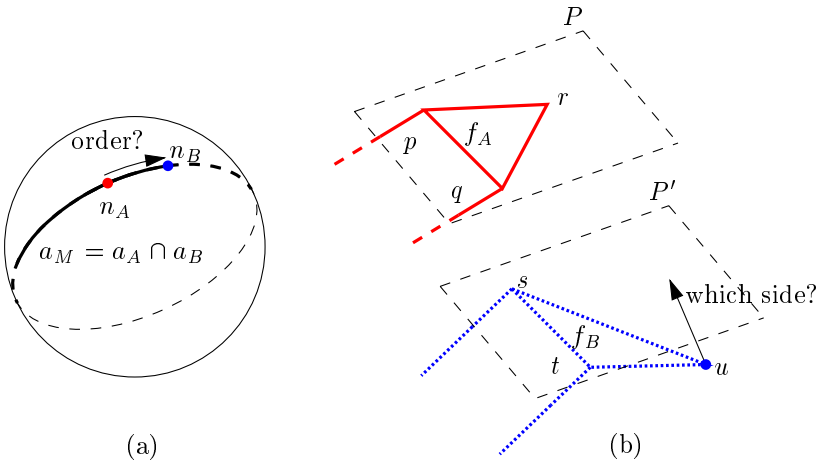


Figure 3.15: The **Node-Order-On-Great-Circle** test: determining the order between two nodes on a great circle. (a) nodes n_A and n_B on the great circle defined by $[p, q]$, (b) node n_B comes after n_A if u is behind $P' = P + qt$.

Azimuth-Order Arc a_B can intersect with arc a_A if it cuts the slab defined by a_A (see Section 3.4.2). If a_B is entirely beyond this slab in increasing azimuth, we know that a_A can not intersect with a_B nor with any arc further on the cell's

envelope than a_B. We determine this by checking whether n_B is behind the vertical plane containing the center of S^2 and n_A: **Which-Side**$(O, n_A, (0,0,1), n_B)$ (see figure 3.16). Let (x_A, y_A, z_A) and (x_B, y_B, z_B) be the unnormalized copies (as described in equation 3.1) of n_A and n_B respectively. The test simplifies to the sign of the following determinant:

$$\begin{vmatrix} x_A & x_B \\ y_A & y_B \end{vmatrix}.$$

Hence, **Azimuth-Order** is of degree 4.

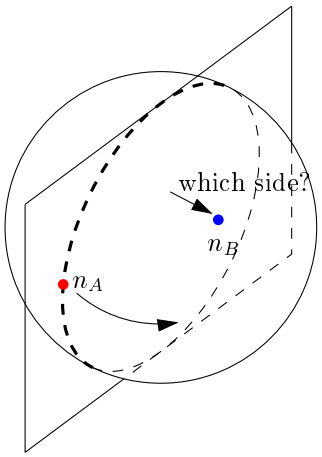


Figure 3.16: Determining the order between the azimuths of the two nodes n_A and n_B.

3.6.2.3 Finding an initial arc

Determining an arc of π to start the walk with.

Edge-On-Shadow $\mathcal{V}'$ is the plane defined by two independent vectors of $\mathcal{V}$ and containing the center of S^2. An edge e is on the shadow of the polyhedron if the nodes n, n' associated with the facets incident to e are at opposite sides of $\mathcal{V}'$: **Which-Side**$(\mathcal{V}', n)$ and **Which-Side**$(\mathcal{V}', n')$ verify this. We define $\mathcal{V}$ as $Y = 0$, so this simplifies to the evaluation of the signs of the y-coordinates of the unnormalized copies of n and n' (equation 3.1). Consequently, **Edge-On-Shadow** has degree 2.

Shadow-Arc-Order To traverse the arclists associated with the shadows of A and B, we must determine the order between arcs. Let $[p, q]$ and $[r, s]$ be the edges

associated with two arcs we compare. The order in which these arcs are cut by Γ is given by the comparison of the slopes of the projections of $[p, q]$ and $[r, s]$ onto the shadow plane $\mathcal{V}$. It is given by **Which-Side**$(O, pq, n_{\mathcal{V}}, rs)$, where $n_{\mathcal{V}}$ is the normal of the shadow plane. Since we defined $\mathcal{V}$ by $Y = 0$, this simplifies to

$$\begin{vmatrix} x_q - x_p & x_s - x_r \\ z_q - z_p & z_s - z_r \end{vmatrix},$$

which is of degree 2. See figure 3.17.

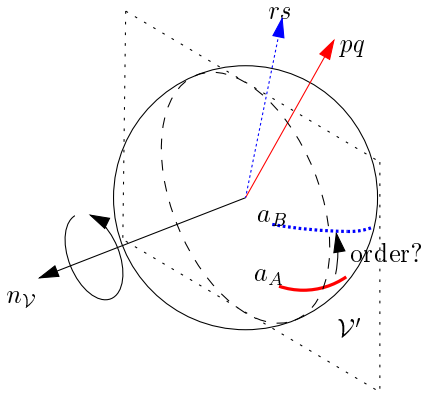


Figure 3.17: Determining the order between edges $[p, q]$ and $[r, s]$ on the shadow. The edges are associated with arcs a_A, a_B respectively.

Consequently, all tests needed to run the algorithm are of degree 4 or less. The only test that is actually of degree 4 is **Azimuth-Order**.

3.6.3 Exact determinant sign evaluation

We want to make Algorithm Minkowski-Sum-Slice numerically stable. To get rid of the numerical problems we encountered, we decided to implement those tests using the exact determinant sign evaluation test discussed in Section 2.4.3.

The coordinates of the vertices of the input polyhedra being represented by b bit integers, the entries of the 3×3 matrix in the **Which-Side** tests are represented by integers taking at most $2b + 3$ bits, since $2b + 3$ bits are taken for components of a calculated node. Avnaim et al.'s method needs a precision of $e + 2$ bits to evaluate the sign of a determinant with entries of e bits. Hence, since we use standard doubles, which allow exact representation of 53 bits integers, we can use 24-bit integers to represent our data, allowing a domain of $[0 \dots 16777215]$ for the coordinates of the polyhedron vertices, which is largely enough in practice.

3.7 Experimental results

We have tested the implementation of Algorithm MINKOWSKI-SUM-SLICE on a realistic although fictitious satellite model called PRISMESAT. The model is decomposed in convex polyhedra. Table 3.1 on page 27 presents the characteristics of the model (numbers of polyhedra and vertices). We present here only the performances of the algorithm per individual run of MINKOWSKI-SUM-SLICE and per placement of an instrument; the results of the layout are presented in chapter 5.

Table 3.2 gives the performance of Algorithm MINKOWSKI-SUM-SLICE for the slices computed at the different stages of the layout scenario. Time is measured in milliseconds CPU-time, and both total and average times are given, the average being calculated on all the Minkowski sums involved. The tests have been performed on a SUN SPARCstation20 running SunOS.[1]

Instrument	GUIBAS&SEIDEL [GS87]		MINKOWSKI-SUM-SLICE	
	Average	*Total*	*Average*	*Total*
WALL	-	-	-	-
ESRI-1	113.2	18339.1	26.6	4316.5
ESRI-2	121.5	20772.3	25.7	4399.8
HORN-7-HST	161.3	67738.7	39.5	16599.3
SVS-ANTENNA	648.3	434371.9	85.8	57514.4
VOGO	387.5	298412.4	63.0	48481.4
TVGAVO-ENS	250.9	87318.2	54.8	19065.9
Total	*364.8*	*926952.6*	*59.2*	*150377.3*

Table 3.2: Performance of the algorithms in milliseconds CPU-time.

We compare the practical performance of our algorithm to compute the slice of a Minkowski sum with the performance of the algorithm proposed by Guibas and Seidel [GS87]. This algorithm uses a topological sweepline algorithm to determine all the intersections of the superposed Gaussian diagrams of two convex polyhedra, constructing the Minkowski sum along the way. This is done in $O(m+n+K)$ time, where m and n are the sizes of the input polyhedra and K the size of the resulting Minkowski sum. The time to slice the Minkowski sums by the plane $\mathcal{Z}$ is added to the total time. Our implementation of this algorithm uses floating point arithmetic and is therefore not provably robust.

We also implemented a brute force method for computing the Minkowski sum. It consists in computing all vertices $v_A + v_B$ for all mn pairs of vertices $v_A \in A, v_B \in B$

[1] SPARCstation and SunOS are trademarks of Sun Microsystems, Inc.

and then calculating their convex hull [AB89] with the Quick Hull algorithm [BDH93]. This algorithm also uses floating point arithmetic. As expected, the performances were even poorer than Guibas and Seidel's algorithm because many points are taken into account that are not vertices of the Minkowski sum.

As can be seen, even though our implementation of Algorithm MINKOWSKI-SUM-SLICE uses the exact evaluation of signs of determinants which is more costly than floating point arithmetic, it is a lot faster than the other two methods tested.

3.8 Conclusions

We have proposed an algorithm to compute a section of a three-dimensional Minkowski sum by a plane without computing the whole sum. The algorithm runs in $O(n + m + K_s)$ time, where n and m denote the complexities of the input polyhedra and K_s denotes the size of the output section of the Minkowski sum. It is therefore optimal and output-sensitive. Degeneracies such as coplanar facets in the input polyhedra and overlapping arcs in the corresponding Gaussian diagrams have been handled. Numerical problems were solved by using exact computation when evaluating the geometric tests arising in the algorithm.

The algorithm was used to compute the admissible space for the placement of antennas on a satellite. The implementation was compared to two methods that calculate the entire Minkowski sum. Our method was shown to be very efficient in practice on realistic satellite models.

Chapitre 4

Calculer l'union de polygones

(Calculating the Union of a Set of Polygons)

4.1 Introduction

Calculating the union of a set of polygons is an operation that is of wide interest in computational geometry. For instance, it is used to calculate the admissible space for motion planning and layout problems [KLPS86, Avn89, Dan95], and for hidden surface removal [OWW85b, dB92, KOS92].

A polygon represents a region that is the interior of the polygon. If a polygon is not simple, its interior is not well-defined. Consequently, we only consider simple polygons. See figure 4.1 for some examples of simple and non-simple polygons.

Notice that the union of a set of polygons greatly depends on whether we consider the boundaries of the polygons to be part of the polygons or not. The number of connected components may differ and the boundary of the union can contain overlapping edges (we call these *zero-width passages*) and holes that are reduced to points. See for example figure 4.2.

It is necessary to preserve zero-width passages, for example, when the union represents the complement of an admissible space for an object and we want to allow contacts with obstacles. The boundaries of the input polygons are then not part of the forbidden regions these polygons define and a zero-width passage corresponds to a sequence of configurations of an object along a line in which it stays in contact with at least two obstacles. This occurs in our application to satellite layout.

The input sets we consider consist of polygons that either all include or exclude

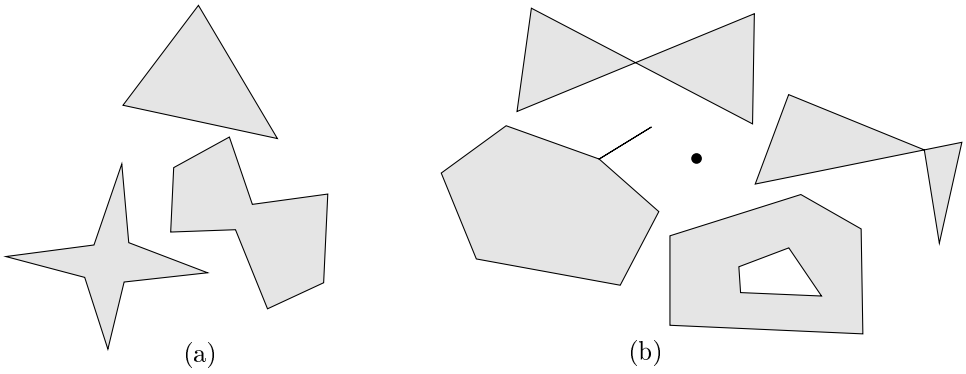


Figure 4.1: (a) some simple polygons, (b) some non-simple polygons.

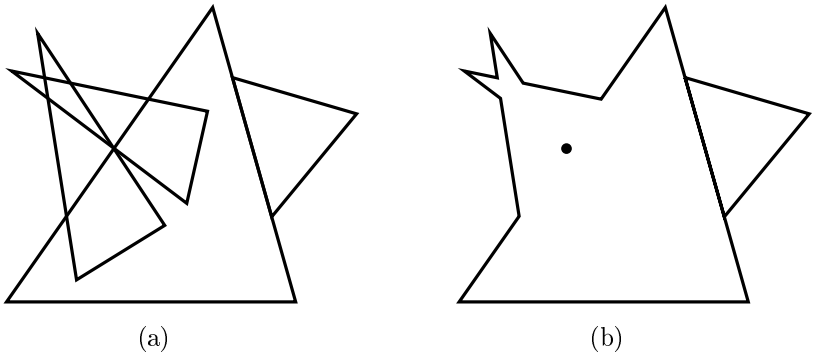


Figure 4.2: Example of a polygonal region with overlapping edges on its boundary and a hole reduced to a point. (a) A set of polygons not in general position and not including their boundaries. (b) Its union.

their boundaries: we disallow combinations of both types of inputs. If this were allowed, parts of the boundary of the union could be part of the union while other parts were not.

Though the complexity of the union of a set of polygons is $\Theta(n_e^2)$ where n_e is the total number of edges of the input polygons, it is linear for most practical examples. Even so, the set of edges of the input polygons might have $\Theta(n_e^2)$ intersections in the interior of the union (see for example figure 4.3). Gajentaan and Overmars define in [GO95] the class of 3SUM-hard problems for whose members no algorithm is known with complexity $o(n^2)$ (for input size n) if the output size is $o(n^2)$. The problem UNION-OF-POLYGONS is shown to be in this class so no output-sensitive algorithm is known.

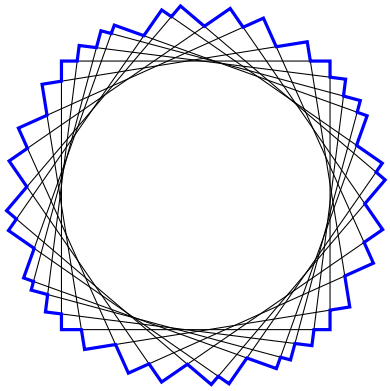


Figure 4.3: An example where the Divide & Conquer scheme only encounters $o(n^2)$ of the $O(n^2)$ intersections: the union of any subset of the rotated squares has linear size.

A Divide & Conquer approach is applied to avoid handling all intersections between polygon edges in order to determine the union. It takes advantage of the fact that the union of a subset of the polygons often hides many intersections between edges of polygons in that subset and edges of polygons in other subsets. We argue that when using a Divide & Conquer scheme, we encounter only a subquadratic number of intersections in the majority of practical examples because of this reason.

The rest of this chapter is organized as follows. In Section 4.2, we show that the problem UNION-OF-POLYGONS belongs to the class of 3SUM-hard problems. Section 4.3 presents an algorithm that calculates the union of a set of polygons by deducing it from the arrangement of the polygon edges. We analyze the theoretical complexity of the algorithm and the algebraic degree of two algorithms that calculate the arrangement.

In Section 4.4, we discuss the algorithm introduced by Kedem et al. [KLPS86] that calculates the union of a set of simple polygons using a Divide & Conquer strategy and we discuss its complexity. Then, Section 4.5 analyzes its algebraic degree and shows how we handled the numerical precision problems forthcoming from the high-degree tests used in the algorithm.

In Section 4.6, we give a lower bound on the algebraic degree of Problem UNION-OF-POLYGONS and show how the Divide & Conquer algorithm can be made to work using lower-degree tests. Finally, we will show some experimental results in Section 4.7.

In the sequel of this chapter, we use the following notations. $\mathcal{P} = \{P_1, P_2, \ldots, P_{n_p}\}$ denotes the set of n_p polygons of which we calculate the union $\mathcal{U}$. The boundary of $\mathcal{U}$ has s vertices. $\mathcal{E}$ is the set of n_e edges of all polygons of $\mathcal{P}$. $\mathcal{A}$ denotes the arrangement of $\mathcal{E}$ and $K_{\mathcal{A}}$ is the number of intersections between edges in $\mathcal{E}$.

4.2 Lower bound: 3SUM-hard problems

Gajentaan and Overmars introduce in [GO95] the class of 3SUM-hard problems, for which algorithms are not known that run in $o(n^2)$ time for inputs of size n with $o(n^2)$ output size. We show that UNION-OF-POLYGONS is in the class of 3SUM-hard problems by reduction to the STRIPS-COVER-BOX problem that is known to be 3SUM-hard.

Problem: STRIPS-COVER-BOX.

> *A strip is defined as the region between two parallel lines. Given a set of strips in the plane, does their union completely cover a given axis-parallel rectangle?*

Theorem 4.1 UNION-OF-POLYGONS *is 3*SUM*-hard.*

Proof Let S be a set of n strips whose union has size s and let R be a rectangle. Let $Q_1, \ldots, Q_4$ be four strips whose union contains a hole whose boundary is identical to the boundary of R (see figure 4.4). We assume that all strips include their boundaries. We also assume that all is in general position: every intersection is reduced to a point and the boundaries of no more than two strips intersect in one point.

Strips are not polygons. By truncating every strip using two lines perpendicular to the strip and outside R, we obtain a set of rectangles. Notice that all their vertices are outside R. Calculate the union $\mathcal{U}$ of the truncated strips of $S \cup \{Q_1, \ldots, Q_4\}$ (see

figure 4.4). All vertices of the boundary of $\mathcal{U}$ are defined by intersections of polygon edges.

Suppose that R is not entirely covered by the strips. This means that $\mathcal{U}$ contains at least one hole that is entirely contained in R. Since the boundary of such a hole is a closed connected component of the boundary of $\mathcal{U}$, it is a polygon and its boundary has at least 3 vertices. Obviously, these vertices are inside or on the boundary of R.

Suppose that R is entirely covered by the strips. Clearly, R is then completely contained in the union of the set of strips. Hence, no vertex of the boundary of the union can be contained in R or on the boundary of R.

Consequently, if the union of the truncated strips has a vertex that is inside R or on the boundary of R, which can be verified in $O(s)$ time (where s is the size of $\mathcal{U}$), then the rectangle is not entirely covered by the strips. We have therefore reduced in $O(s)$ time UNION-OF-POLYGONS to STRIPS-COVER-BOX. It follows that when $s = o(n^2)$, UNION-OF-POLYGONS is 3SUM-hard. Clearly, if $s = \Theta(n^2)$, the same result holds. □

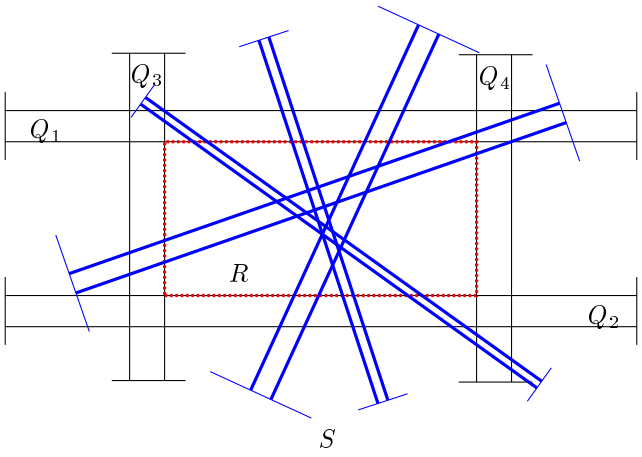


Figure 4.4: Illustration of the proof of the lower bound theorem.

This shows that an output-sensitive complexity for the calculation of the union of a set of polygons cannot be proven unless it is proven that any 3SUM-hard problem can be solved in $o(n^2)$ time.

4.3 Deducing the union from the arrangement of the polygon edges

The arrangement $\mathcal{A}$ of the edges of all polygons can be used to calculate the union of the polygons: the boundary of the union is defined by a subset of the edges of $\mathcal{A}$. We assume the polygons to be in general position: no three edges from $\mathcal{E}$ intersect in one point, edges from $\mathcal{E}$ only intersect in their interior except when they correspond with adjacent edges of the boundary of one of the polygons and all intersections between edges of $\mathcal{E}$ are reduced to points.

$|\mathcal{A}| = \Theta(n_e + K_\mathcal{A}) = \Theta(n_e^2)$. Using an algorithm proposed by Chazelle and Edelsbrunner, $\mathcal{A}$ is calculated in output-sensitive time $O(n_e \log n_e + K_\mathcal{A})$ [CE92]. This algorithm has algebraic degree 4 [BP97].

Another method to calculate $\mathcal{A}$ is by randomized incremental construction [CS89, BDS+92, Tei93]. Edges are inserted in the arrangement one by one in random order using the trapezoidal map of $\mathcal{A}$ that is incrementally constructed this way. Though the randomized algorithm runs slower in the worst case, a randomized analysis shows it to run in $O(n_e \log n_e + K_\mathcal{A})$ *expected time*. This means it is efficient in practice. A randomized algorithm that calculates $\mathcal{A}$ using its trapezoidal map has algebraic degree at least 5, since it must be able to determine the order between intersections (see Section 4.5.2.1) in order to construct the trapezoidal map.

We suppose that, when $\mathcal{A}$ is calculated, adjacency information is available. That is, given an edge $e_\mathcal{A} \in \mathcal{A}$, we can access the two cells it is incident to, and given $e_\mathcal{A}$ and one of the two cells $c_\mathcal{A}$ it is incident to, we can access the preceding and succeeding edge of $e_\mathcal{A}$ of the boundary of $c_\mathcal{A}$. We suppose also that given edge $e_\mathcal{A}$, we can directly obtain its supporting edge $e \in \mathcal{E}$ and the polygon $P_i \in \mathcal{P}$ that owns e.

Notice that no two cells contained in the exterior of $\mathcal{U}$ can be incident. A cell $c_\mathcal{A} \in \mathcal{A}$ is contained in $\mathcal{U}$ when there exists a polygon $P \in \mathcal{P}$ that has an edge that supports an edge of $c_\mathcal{A}$ and $c_\mathcal{A} \subseteq P$. Hence, we mark all cells that are contained in $\mathcal{U}$ in $O(K_\mathcal{A})$ time by inspecting (at most) all edges of $\mathcal{A}$. Then we remove every edge of $\mathcal{A}$ that is incident to two cells contained in $\mathcal{U}$. At the end, a set of edges is left defining one or more closed chains that describe the boundary of $\mathcal{U}$. This is done in $\Theta(K_\mathcal{A})$ time.

Consequently, an algorithm that calculates the arrangement of the edges of the polygons and uses it to construct the union runs in $O(n_e \log n_e + K_\mathcal{A})$ time. Notice that this is not an output-sensitive result: $K_\mathcal{A}$ can be $\Theta(n_e^2)$ even if $s = o(n_e^2)$.

4.4 The Divide & Conquer algorithm

In this section, we describe a Divide & Conquer algorithm that calculates the union of $\mathcal{P}$.

The algorithm was proposed in [KLPS86]. The Divide & Conquer strategy consists of solving a problem by cutting it up into smaller subproblems that are solved separately and then combining the results. We split the set of polygons in two (division), recursively calculate the unions of both subsets and then calculate the union of the two results (fusion). The fusions are performed by a sweepline algorithm, for example the one described in [OWW85a]. At the deepest level of recursion, we end up with a set of size 1 whose union is determined trivially.

The scheme of the algorithm can be seen as the postorder walk of a binary tree $\mathcal{T}$ of depth $\Theta(\log n_p)$, whose leaves are the input polygons P_i representing polygonal regions, and in which each internal node is a polygonal region representing the union of its two child nodes.

Consider the tree node N and the polygons P_1, P_2 that are leaves of the left and right subtree of N respectively. Let N_l, N_r be the left and right child of N respectively. Let $e_1 \in P_1$ and $e_2 \in P_2$ be edges that intersect in point I. If I is strictly contained in the polygonal region N_l or N_r, the intersection between e_1 and e_2 will not be encountered by the fusion step at N nor at any fusion step at a tree node that is an ancestor of N (see for example figure 4.5).

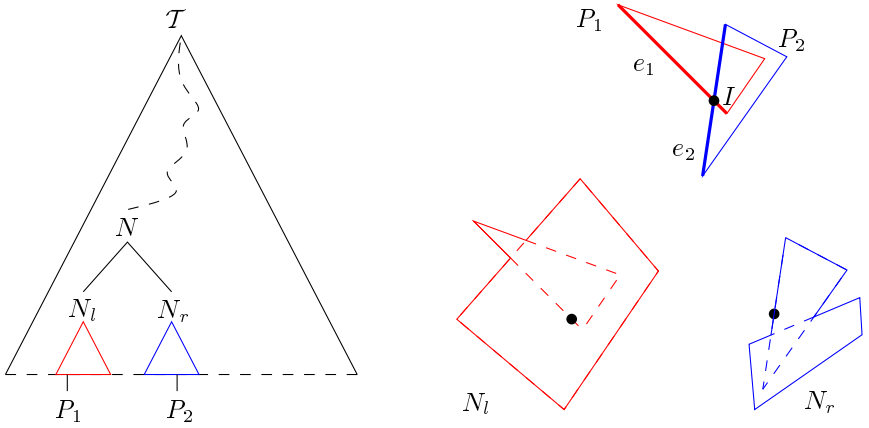


Figure 4.5: An example of an intersection between two polygonal edges that will not be encountered by the Divide & Conquer algorithm: the polygonal region N_l hides the intersection between edges e_1, e_2.

Since I is in the interior of the polygonal region N, it is also in the interior of the union $\mathcal{U}$ of $\mathcal{P}$, hence it is not a vertex of the boundary of $\mathcal{U}$. We argue that many such "internal" intersections are hidden this way for most practical examples and thus never encountered by the Divide & Conquer algorithm. Of course, this greatly depends on the nature of the input and on the way sets of polygons are split in two. We will show that for some classes of polygons, it can be proven that at most $o(n_e^2)$ intersections can be encountered and we will show some experimental results.

4.4.1 Sweeping two polygonal regions

The major part of the algorithm is the computation of the union of two polygonal regions (the fusion step), both being the result of the union of a subset of polygons. The union of the polygonal regions is determined by calculating the intersections of their boundaries. Notice that all intersections detected in this step appear on the boundary of the result of the sweep, so no unnecessary information is calculated in an individual fusion step.

We take advantage of the connectedness of the polygonal chains by using the sweepline algorithm proposed in [OWW85a] that calculates the union by moving a vertical line (the *cut*) from left to right.

$\mathcal{R}$ and $\mathcal{B}$ are the two polygonal regions that are swept; their boundaries consist of n_R and n_B edges respectively. We color $\mathcal{R}$ and $\mathcal{B}$ red and blue respectively, as well as their elements. $\mathcal{R} \cup \mathcal{B}$ is the union of $\mathcal{R}$ and $\mathcal{B}$ and $\partial\mathcal{R}, \partial\mathcal{B}, \partial(\mathcal{R} \cup \mathcal{B})$ denote the boundaries of $\mathcal{R}, \mathcal{B}, \mathcal{R} \cup \mathcal{B}$ respectively. We assume for the moment that $\partial\mathcal{R}$ and $\partial\mathcal{B}$ are in general position: no vertices coincide, no edges contain vertices and no more than two edges intersect in one point. Degeneracies will be discussed later on.

4.4.1.1 Polygonal regions

Though a simple polygon can be represented as one closed chain of vertices, this is not necessarily the case for the union of several polygons: it can consist of several connected components and can have holes in it (see for example figure 4.6).

Notice that the boundary of the polygonal region uniquely defines the polygonal region itself since we know it is finite: one can determine whether a point p is in its interior by checking whether a line segment connecting p with a point at infinity cuts the boundary an odd number of times.

The simplest way to represent the boundary of a polygonal region is by describing it as a set of closed simple chains, every chain representing a connected component of the boundary. Such a connected component of the boundary either encloses a connected component or a hole of the polygonal region. Notice that this simple structure

would not be sufficient when we would want to calculate the union of polygons that include their boundaries with other polygons that do not, which is why we disallow this.

The sweepline algorithm sweeps polygonal regions from left to right. To avoid sorting the vertices of the boundary of a polygonal region, we represent the boundary as the list of its vertices with their adjacencies stored in a table sorted in left to right order, the vertices describing one or more doubly connected circular lists (see figure 4.7). We call a vertex that is to the left of both its incident edges a *left extreme vertex*, a *right extreme vertex* if it is to the right of them and an *inbetween vertex* otherwise.

Notice that the execution of the sweepline algorithm at a given node of the Divide & Conquer tree $\mathcal{T}$ takes as input the polygonal regions attached to its two child nodes:

- If a child is an internal node of $\mathcal{T}$, the sweepline algorithm was executed to obtain the polygonal region. Because the sweepline algorithm works from left to right, the vertices of the boundary of that polygonal region were stored in such a table in the right order.

- If the child node is a leaf of $\mathcal{T}$, a table is constructed from the corresponding input polygon. If the polygon is monotonic in x (that is, any vertical line cuts its boundary at most twice), this is done in $O(n'_e)$ time by merging the sequences of vertices of its upper and lower envelope; the vertices of a general simple polygon are sorted in $O(n'_e \log n'_e)$ time, where n'_e is the number of edges of the polygon in both cases.

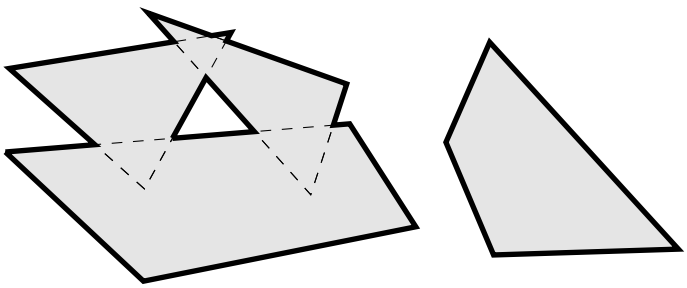


Figure 4.6: An example of a polygonal region that is the union of four polygons.

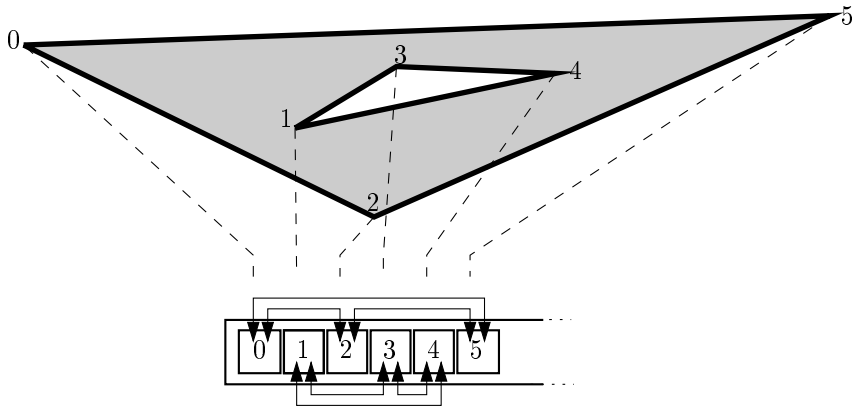


Figure 4.7: Storage of the boundary of a polygonal region.

4.4.1.2 The cut

The cut $\mathcal{C}$ is the sorted sequence of edges cut by the sweepline at some moment during the sweep. For each edge in the cut, we memorize

1. its color (i.e. the color of the polygonal region it belongs to),
2. whether the interior of the owning polygonal region is above or below it,
3. whether it is in the interior of the other polygonal region at the point where it is cut by the sweepline.

Obviously, we directly determine its color. Informations 2 and 3 are kept up to date as follows (see also figure 4.8):

- (a). When inserting a pair of edges e_1, e_2 incident to a leftmost vertex, inspect the edge $e_\wedge$ directly above them in the cut. Because we memorized informations 2 and 3 for $e_\wedge$, we can deduce them for e_1 and e_2. The informations for the edges directly above and below e_1, e_2 in the cut does not change.

  Obviously, when no edge is above e_1 and e_2 in the cut, we know that the interior of the polygonal region owning them is between e_1 and e_2 and that we are outside the other polygonal region: the polygonal regions we handle are finite.

- (b). When the cut steps over an inbetween vertex, an edge is replaced by another from the same polygonal region belonging to the same chain. Hence, the informations are simply copied from the edge that is removed from the cut and the information for the edges directly above and below it in the cut remains unchanged.

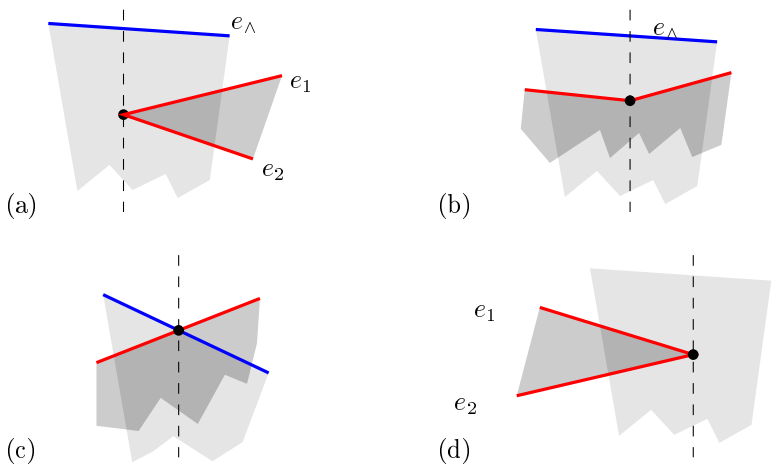


Figure 4.8: How to keep adjacency information up to date for edges in the cut.

- (c). When a red and a blue edge intersect, the red edge passes from the interior to the exterior of $\mathcal{B}$ or inversely and the same happens symmetrically for the blue edge with $\mathcal{R}$.

- (d). When the cut steps over a rightmost vertex and the two edges incident to it disappear from the cut, the information for the edges directly above and below them in the cut remains unchanged.

4.4.1.3 Complexity

Let $\mathcal{Q}$ denote the event queue and $\mathcal{C}$ the sequence of edges cut by the sweepline. Let $k = \Theta(n_R n_B)$ be the number of intersections between red and blue edges. $n_R + n_B + k$ events are to be handled.

Only intersections need to be inserted in $\mathcal{Q}$: since the vertices of $\partial\mathcal{R}$ and $\partial\mathcal{B}$ are already sorted, the next event is determined by comparing the leftmost as yet unswept vertices of $\partial\mathcal{R}$ and $\partial\mathcal{B}$ and the first following detected intersection. Determining the next event costs $O(\log k) \leq O(\log n_R + \log n_B)$.

For each event, locating its one or two incident edges and the neighbors above and below them in $\mathcal{C}$ is done in constant time if we keep pointers to the edges in the cut. Updating $\mathcal{C}$ by removing and/or inserting at most two edges costs $O(\log(n_R + n_B))$ time. Testing for at most two intersections is done in constant time and a detected intersection is inserted in $O(\log n_R + \log n_B)$ time in $\mathcal{Q}$. Output of the result (at most two edges and one vertex) is done in constant time.

So, in total, the sweepline algorithm takes $O((n_R + n_B + k)\log(n_R n_B))$ time to calculate $\mathcal{R} \cup \mathcal{B}$.

4.4.2 Complexity of the Divide & Conquer scheme

Calculating the union of a set of polygons is hard in the sense that, even when the output size is less than quadratic with respect to the input size, a quadratic number of intersections might be encountered during the computation (theorem 4.1). In this section, we prove an upper bound for the Divide & Conquer algorithm and show what happens when we make some special assumptions on the input polygons. Notice that the complexity of the Divide & Conquer algorithm is proven in [KLPS86] for the special case that the union of any subset of $\mathcal{P}$ has linear size in its input (in that case, when the numbers of edges of the polygons are bounded by a constant, the algorithm runs in time $O(n_p \log^2 n_p)$) and that our proof is the generalization of this result.

$K_{\mathcal{A}}$ denotes the number of intersections between all pairs of polygon edges. K is the number of distinct intersections encountered during the Divide & Conquer algorithm, s is the size of the output. Obviously, $s \leq n_e + K$ and $K \leq K_{\mathcal{A}}$. Notice that a detected intersection can reappear as a vertex of the boundary of a polygonal region in $\Theta(\log n_p)$ successive fusion steps. We distinguish between *polygon vertices*, which are defined by vertices from input polygons, and *intersection vertices* which are vertices of the boundaries of (partial) unions defined by intersections between edges of input polygons.

Theorem 4.2 *Given is a set of n_p polygons $\mathcal{P}$ with every polygon having at most a constant number c of edges (so $n_e \leq cn_p$). If the Divide & Conquer algorithm computes K intersections, it will run in $O((K + n_p)\log^2 n_p)$ time.*

Proof The algorithm subdivides $\mathcal{P}$ in two subsets $\mathcal{P}_1$ and $\mathcal{P}_2$ of size $\lceil n_p/2 \rceil$ and $\lfloor n_p/2 \rfloor$ respectively, calculates the unions $\mathcal{U}_1$ and $\mathcal{U}_2$ of $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively and then calculates the union of $\mathcal{U}_1$ and $\mathcal{U}_2$. This corresponds to the following recurrent equation describing the worst case:

$$T(n_p) \leq \begin{array}{ll} 2 * T(\lceil \frac{n_p}{2} \rceil) + O((n_p + k + K_1 + K_2)\log n_p) & , n_p > 1 \\ c & , n_p = 1 \end{array}$$

where K_1 and K_2 are the numbers of intersection vertices of $\partial\mathcal{U}_1$ and of $\partial\mathcal{U}_2$ respectively and k the number of intersections between $\partial\mathcal{U}_1$ and $\partial\mathcal{U}_2$. Obviously, $k + K_1 + K_2 \leq K$. Hence

$$T(n_p) \leq O((K + n_p)\log^2 n_p),$$

which concludes the proof. □

A tighter upper bound can be proven when we make special assumptions on the input data. For instance, it is shown that the size of the union of the entire set $\mathcal{P}$ is linear if no pair of polygons intersects in more than two points [KLPS86]. Since this means that the sizes of the intermediate results are linear in their input size also, the algorithm runs in $O(n_p \log^2 n_p)$. This is for example the case when $\mathcal{P} = \{A \oplus B \mid B \in \mathcal{B}\}$, where $\{A\} \cup \mathcal{B}$ is a set of convex polygons.

Matoušek et al. introduce the notion of δ-*fat* triangles: a triangle is δ-fat if its interior angles are at least δ [MPS+94]. They show that the union of a set of δ-fat triangles has size $O((n_p \log\log n_p)/\delta)$. van Kreveld generalizes this result to simple polygons by introducing the notion of δ-*wideness* of polygons: a polygon is δ-wide if the width/length ratio of any passage between two edges of the polygon is at least δ: the union of a set of δ-wide polygons has size $O((n_p \log\log n_p)/\delta)$. Hence, on inputs consisting of δ-wide polygons, the Divide & Conquer algorithm runs in time $O((n_p \log\log n_p)/\delta \log^2 n_p)$ [vK93].

4.4.3 Handling degeneracies during the sweep

In the previous section, we have assumed that $\partial\mathcal{R}$ and $\partial\mathcal{B}$ are in general position. We want the algorithm to be able to handle input in non-general position [BMS94]. The following degeneracies can occur:

1. If two vertices are contained in a vertical line, the two corresponding vertex events occur at the same moment. Furthermore, if the two vertices have the same color and are adjacent, the edge they are incident to will be contained in the cut $\mathcal{C}$ when the events are processed. This degeneracy disappears when we use a lexicographic order on events by defining an order along y between events with the same x-coordinate.

2. Two or more vertices coincide. The boundaries of polygonal regions consist of polygonal chains and every vertex is adjacent to exactly two other vertices. So, to be able to model a polygonal region with touching chains (the union of two polygons that touch in a point for example), we must allow coinciding vertices in a polygonal region.

3. A red edge contains a blue vertex or vice versa. In the result, we avoid generating edges that contain a vertex by splitting the edge in two at that point and adding a vertex. Consequently, we do not have to allow edges containing vertices of the same color.

4. Edges intersect along a line segment that is not reduced to a point. If the edges have the same color, this represents a zero-width passage in the corresponding input polygonal region (see for example figure 4.2 in the introduction of this chapter).

5. Holes in a polygonal region that are reduced to points (see figure 4.2 in the introduction of this chapter).

We define a type of event that handles all degenerate cases:

- The event may consist of more than one coinciding vertex (2).
- The event may be contained in an edge (3).
- When edges overlap (4), both endpoints of the overlapping part either coincide with two vertices or coincide with a vertex and are on the interior of an edge, so cases (2) and (3) handle it implicitly.
- A hole reduced to a point is entirely contained in an event (5).

One or more vertices at position p_e define a vertex event to be handled. Two sets of edges E_L, E_R are incident to p_e to the left and right side of the event respectively (see figure 4.9a).

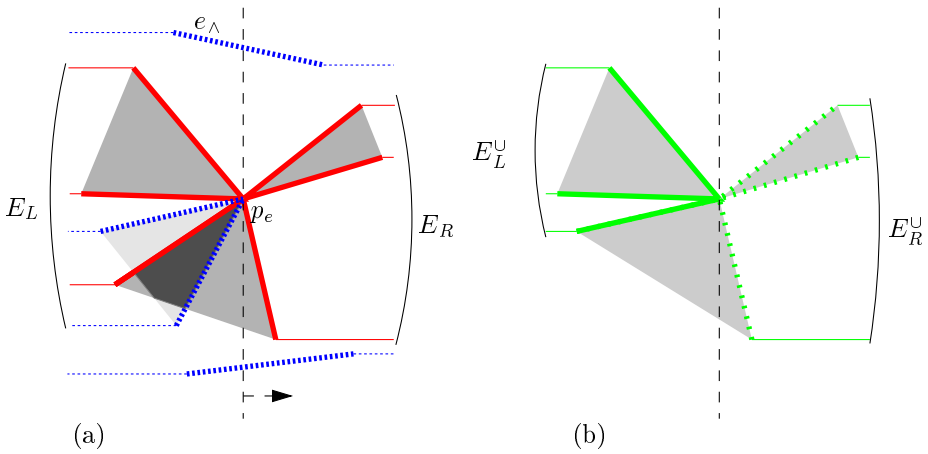


Figure 4.9: Handling a vertex event. (a) Sweeping over the event. (b) The edges of $\partial(\mathcal{R} \cup \mathcal{B})$ that are drawn solid are generated.

Notice that an intersection of the interior of two edges cannot coincide with a vertex event: two edges with the same color cannot intersect, so a red and a blue edge should then contain p_e. But in that case, neither a red nor a blue vertex can coincide with p_e. One edge might contain p_e: this is detected at the moment the vertex event is handled when E_L is determined from $\mathcal{C}$. We remind the reader that an edge containing a vertex is cut in two if it supports edges of the boundary of the result.

So, there are two types of events that are to be handled: *intersections between the interiors of two edges that are reduced to a point* and *vertex events that might contain degeneracies.*

Special care is taken for holes that are reduced to points: we remind the reader that these can occur when the input polygons do not include their boundaries. Such a hole of $\mathcal{R}$ (resp. $\mathcal{B}$) defines a hole reduced to a point of $\mathcal{R} \cup \mathcal{B}$ if it is *not strictly inside* $\mathcal{B}$ (resp. $\mathcal{R}$).

A vertex event p_e is swept as follows:

1. Using the ordered vertex tables of $\mathcal{R}$ and of $\mathcal{B}$, find the set of vertices V that coincide with p_e: walk both tables from left to right starting from the first as yet unswept vertex.

2. E_L is the contiguous sequence of edges in $\mathcal{C}$ incident to the vertices from V or containing p_e in their interior. Note that the implementation of $\mathcal{C}$ as a search tree allows an edge to be accessed in constant time from another edge it is adjacent to in $\mathcal{C}$. We obtain E_L from $\mathcal{C}$ as follows:

   (a) Locate in $\mathcal{C}$ an edge e incident to any vertex in V. If pointers are kept to the edges in the cut, this is done in constant time.

   (b) Starting from e, walk $\mathcal{C}$ upwards and downwards as long as the following edge encountered is incident to a vertex in V or contains p_e in its interior. Add all those edges in the correct order to E_L.

   (c) Call $e_\wedge$ the first edge encountered above that is not part of E_L (and thus terminates the upward walk through $\mathcal{C}$). Obtain from $e_\wedge$ the informations described in Section 4.4.1.2; these allow us to determine which parts of $\partial(\mathcal{R} \cup \mathcal{B})$ are to be generated.

   (d) If, during the walk upwards or downwards, an edge is encountered that contains p_e in its interior, call it $e_\times$.

3. Generation of vertices of $\partial(\mathcal{R} \cup \mathcal{B})$. Two cases are distinguished and handled separately:

   - p_e contains a red *hole* that is reduced to a point. A hole reduced to a point of $\partial(\mathcal{R} \cup \mathcal{B})$ is generated if p_e is outside $\mathcal{B}$, V contains at least one other vertex or $e_\times$ exists (notice that both are necessarily blue). Otherwise, nothing is generated. Blue isolated holes are handled symmetrically.
   - All other cases. By walking E_L and inspecting the informations kept with $e_\wedge$, determine which edges incident to p_e are part of $\partial(\mathcal{R} \cup \mathcal{B})$. We call this set $E_L^\cup$ (see figure 4.9b). If $\mathcal{R}$ and $\mathcal{B}$ do not include their boundaries, coinciding edges define a zero-width passage of $\partial(\mathcal{R} \cup \mathcal{B})$ when these edges have to same color or when the interiors of the owning polygonal regions

are at opposite sides of the edges. Generate the set of right extreme vertices and the adjacencies of $E_L^{\cup}$.

4. Remove E_L from $\mathcal{C}$.

5. Determine E_R from the left extreme vertices in V and, if existant, add $e_\times$ to E_R.

6. Sort E_R in vertical order and insert it in $\mathcal{C}$.

7. Determine the set of edges $E_R^{\cup}$ incident to p_e that are part of $\partial(\mathcal{R} \cup \mathcal{B})$ by walking E_R and inspecting $e_\wedge$ the same way as has been done for $E_L^{\cup}$. Generate the set of left extreme vertices incident to the edges in $E_R^{\cup}$.

It is easy to verify that the modified event handling scheme does not change the complexity of the algorithm.

4.5 Numerical precision issues

We want to make the algorithm robust, so we analyze the numerical precision needed to correctly evaluate the geometric tests (see Section 2.4). The Divide & Conquer scheme only splits sets of polygons in two, calls itself recursively twice and then calls the sweepline algorithm. Hence, we only have to analyze the sweepline algorithm that calculates the union of two polygonal regions that are the unions of subsets of the polygons. The tests and computations we describe are listed in [BP97].

4.5.1 Calculating intersections

Given the pair of intersecting edges $[p_0, p_1], [p_2, p_3]$, the x-coordinate of their intersection I is obtained by dividing the determinants of two 2×2 matrices:

$$x_I = \frac{A_I}{B_I}, A_I = \begin{vmatrix} x_1 - x_0 & x_0 y_1 - x_1 y_0 \\ x_3 - x_2 & x_2 y_3 - x_3 y_2 \end{vmatrix}, B_I = \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_3 - x_2 & y_3 - y_2 \end{vmatrix} \tag{4.1}$$

and $y_I = \frac{A'_I}{B_I}$ is obtained in an equivalent way. It is easy to see that A_I, A'_I and B_I have degree 3, 3 and 2 respectively.

Because of the division, we cannot expect to be able to calculate the coordinates of I accurately using standard arithmetics, so we cannot use them in tests without introducing an error. Even worse, if the input polygonal regions of the sweep were produced by another sweep in the Divide & Conquer scheme, their boundaries have

vertices whose coordinates are inaccurate, implying also that edges incident to such vertices are inaccurate.

This problem could be solved by rounding all generated intersection vertices but then the lexicographic order between the vertices might change and the polygonal can become nonsimple. Additional work would then be needed to sort the vertices and maintain simplicity as described in [GY86, GSS89, Mil90, GM95, Mil95, GGHT97]. Instead, we choose to use an exact representation using integer values:

- we suppose that the vertices $p_i = (x_i, y_i)$ of the input polygons are on an integer grid,
- represent an intersection vertex I exactly as the homogeneous point of triple precision (A_I, A'_I, B_I) whose coordinates are integers,
- represent an edge by the two vertices it is incident to (these may be intersection vertices) and the supporting line defined by two *vertices from an input polygon.*

The advantages of exact representation are

1. the edges constituing the boundary of the result will not deform,
2. the output of a sweep can be used directly as input for another sweep.

4.5.2 Tests

Two types of tests are needed for the algorithm. The first type determines the order between events and the second one determines to which side of an edge a vertex lies. We suppose the edges to be oriented. Notice that the tests handle intersection vertices (detected in a *previous* sweep) and intersections (detected during *this* sweep) in the same way, using homogeneous coordinates. See Section 2.4 for the definition of tests and their degree.

4.5.2.1 The order between events

We determine the order between two events by comparing their x-coordinates.

If two events are both defined by polygon vertices (**Event-Order**$_{pp}$), determining the order between them is done by simply comparing their x-coordinates. Let x_0, x_1 be the x-coordinates of the events we compare. The order is given by the sign of $x_1 - x_0$, which is a test of degree 1.

In an equivalent way, the order between two intersection events I and J is given by the sign of $\frac{A_I}{B_I} - \frac{A_J}{B_J}$. This sign is evaluated without divisions using the determinant of a 2×2 matrix:

$$\textbf{Event-Order}_{ii}(I, J) = sign \begin{vmatrix} A_I & B_I \\ A_J & B_J \end{vmatrix} sign(B_I) sign(B_J) \tag{4.2}$$

This test has degree 5. To compare an intersection event with an event defined by a polygon vertex, notice that the polygon vertex p can be written in the homogeneous form $(x, y, 1)$. Hence, the order is determined as follows:

$$\textbf{Event-Order}_{pi}(p, I) = sign \begin{vmatrix} x & 1 \\ A_I & B_I \end{vmatrix} sign(B_I) \tag{4.3}$$

and this test has degree 3.

A special case of $\textbf{Event-Order}_{ii}$ occurs when two intersections I and J appear on an edge e of an input polygon. Notice that for every edge, we have memorized the two polygon vertices defining that edge. Let x_0, x_1 be the x-coordinates of the polygon vertices that define e. $x_1 - x_0$ is a common factor in the determinant: it can be rewritten as a degree 4 polynomial. Let us call this test $\textbf{Event-Order}'_{ii}$.

4.5.2.2 Which side

The other type of test we need determines to which side of an edge a given vertex lies. It was shown in Section 2.4.2 how the test is defined as the sign of a determinant and that it is of degree 2. However, it takes as input the x and y coordinates of three points so intersection vertices cannot be directly used. For an intersection vertex $I = (A_I, A'_I, B_I)$ and an edge supported by the polygon edge $[p_0, p_1]$, the test is rewritten as follows:

$$\textbf{Which-Side}_i(p_0, p_1, I) = sign \begin{vmatrix} x_0 - x_1 & A_I - x_1 B_I \\ y_0 - y_1 & A'_I - y_1 B_I \end{vmatrix} sign(B_I) \tag{4.4}$$

$\textbf{Which-Side}_i$ has degree 4.

4.5.3 Exact evaluation of determinant signs

We store all expressions needed to evaluate the tests in standard double floating point variables. The IEEE standard 754 specifies that integers up to 53 bits are exactly representable in a double. $\textbf{Event-Order}_{ii}$, the most demanding test, evaluates the sign of a number that needs $5b + 6$ bits to be represented. If we want to represent those values exactly, the input data should be represented using at most 9 bits,

corresponding with an integer domain $[0 \ldots 511]$, which is too small for most practical applications.

As for Algorithm MINKOWSKI-SUM-SLICE, we choose to use the determinant sign evaluation test discussed in Section 2.4.3. The matrix entry demanding the largest number of bits is $A_I - xB_I$ from the **Which-Side**$_i$ test: it needs $3b + 4$ bits. Consequently, we can allow $b = 15$ when using double floating point variables from the IEEE 754 standard, corresponding with an integer domain of $[0 \ldots 32767]$ for the coordinates of the polygon vertices. This is largely sufficient for most applications and in particular for computing the admissible space for our satellite antenna layout application.

4.6 Lower degree algorithms

Are all high-degree tests needed for the sweep algorithm to run correctly? The original sweepline algorithm of Bentley and Ottmann [BO79] demands a precision of degree 5, because it needs to compare the order between intersections.

Boissonnat and Preparata give in [BP97] two algorithms of degree 3 and 2 that determine all the intersections of a set of line segments: they do not depend on the correct evaluation of the order between intersections.

4.6.1 Degree 4 is a lower bound

We cannot directly use these lower-degree algorithms to determine the intersections between the boundaries of two polygonal regions: the algorithms depend on the correct evaluation of the order between input vertices. Since in a recursion step, the boundary of a polygonal region can have intersection vertices, we cannot determine the order between two such vertices. So we are looking for an algorithm that does not impose the correct evaluation of the order between input vertices.

The correct evaluation of tests with degree 4 is a lower bound for any algorithm calculating the union of polygons or more generally an arrangement of line segments: if we cannot determine the order between vertices on a polygon edge (which demands the degree 4 **Event-Order**$'_{ii}$ test), the order between adjacent vertices of the boundaries of the cells in the arrangement cannot be guaranteed. The algorithm of Chazelle and Edelsbrunner (Section 4.3) actually is of degree 4.

Hence, an algorithm of degree 4 exists to calculate the union of a set of polygons. Notice that the order between intersection vertices of the upper envelope (as well as of the lower envelope) of a cell in the arrangement is guaranteed but not the order between arbitrary intersection vertices when a precision of degree 4 is available. We

remind the reader that calculating $\mathcal{A}$ for deducing $\mathcal{U}$ from it is expected to be slow in practice, since *all* intersections between polygon edges are calculated.

4.6.2 Divide & Conquer with a degree 4 fusion algorithm

To show that a Divide & Conquer algorithm of degree 4 exists that calculates the union of a set of polygons, we give here a degree 4 fusion algorithm that calculates the union of two polygonal regions $\mathcal{R}$ and $\mathcal{B}$ that are the unions of the sets of polygons $\mathcal{P}_\mathcal{R}$ and $\mathcal{P}_\mathcal{B}$ respectively.

- $n_{e_\mathcal{R}}, n_{e_\mathcal{B}} \leq n_e$ denote the total numbers of edges of the polygons in $\mathcal{P}_\mathcal{R}$ and $\mathcal{P}_\mathcal{B}$,
- $K_\mathcal{R}, K_\mathcal{B}$ denote the numbers of intersection vertices of $\mathcal{R}$ and $\mathcal{B}$ respectively,
- k denotes the number of times $\partial\mathcal{R}$ and $\partial\mathcal{B}$ intersect.

The boundaries of $\mathcal{R}$ and $\mathcal{B}$ have $O(n_{e_\mathcal{R}} + K_\mathcal{R})$ and $O(n_{e_\mathcal{B}} + K_\mathcal{B})$ vertices respectively. $\mathcal{R}$ and $\mathcal{B}$ are represented as the sets of the connected components of their boundaries. Every connected component is the oriented chain of its vertices, that is, given two consecutive vertices p_0, p_1 on a connected component, the interior of the polygonal region is to the left of the oriented edge $[p_0, p_1]$.

The goal is, given the connected components of the boundaries of $\mathcal{R}$ and $\mathcal{B}$, to determine the connected components of the boundary of $\mathcal{R} \cup \mathcal{B}$. We take advantage of the fact that $\mathcal{R}$ and $\mathcal{B}$ are themselves unions of sets of polygons to efficiently calculate the intersections between $\partial\mathcal{R}$ and $\partial\mathcal{B}$.

4.6.2.1 Supporting edges

$\partial\mathcal{R}$ might have $O(n_{e_\mathcal{R}}^2)$ edges, but these edges are supported by $O(n_{e_\mathcal{R}})$ edges of the input polygons from $\mathcal{P}_\mathcal{R}$. One such polygon edge can support $O(n_{e_\mathcal{R}})$ edges of $\partial\mathcal{R}$ since it can intersect with $O(n_{e_\mathcal{R}})$ other polygon edges. Obviously, the same holds for $\mathcal{B}$.

For a polygonal region $\mathcal{P}$, let $E(\mathcal{P})$ denote the set of polygon edges that support the edges of $\partial\mathcal{P}$. When we generate a polygonal region $\mathcal{P}$, we determine and store $E(\mathcal{P})$ and for each edge $e \in E(\mathcal{P})$, we store all edges of $\partial\mathcal{P}$ that it supports in the order they appear on e. This divides e in a contiguous sequence of intervals, each interval representing an edge of $\partial\mathcal{P}$ or the interior of $\mathcal{P}$. Let $Iv(e, \mathcal{P})$ denote the sequence of intervals on e defined by $\partial\mathcal{P}$ (see figure 4.10). An intersection I occurring on e is located in $Iv(e, \mathcal{P})$ using a binary search. We determine this way whether I is on $\partial\mathcal{P}$ or in the interior of $\mathcal{P}$.

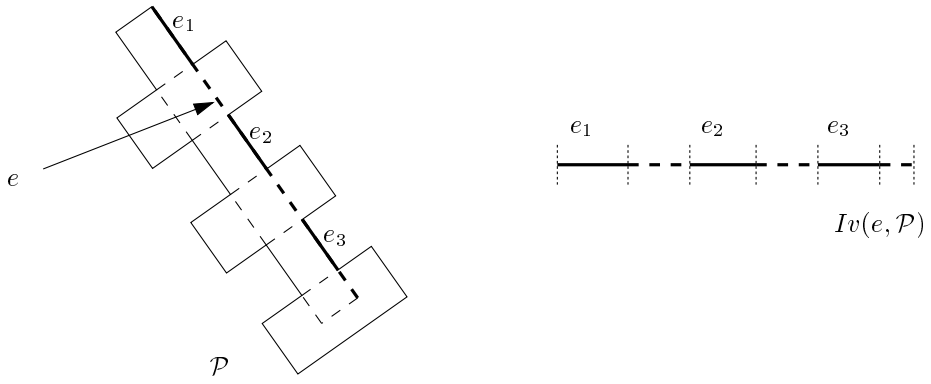


Figure 4.10: The edges of the boundary of the polygonal region $\mathcal{P}$ that are supported by the polygonal edge e divide e in the sequence of intervals $Iv(e, \mathcal{P})$.

4.6.2.2 Finding the intersections

We proceed as follows to detect all k intersections between $\partial\mathcal{R}$ and $\partial\mathcal{B}$. Calculate the intersections of all pairs of supporting edges $(e_\mathcal{R}, e_\mathcal{B}) \in E(\mathcal{R}) \times E(\mathcal{B})$. This takes at most $O(n_{e_\mathcal{R}} n_{e_\mathcal{B}})$ time and uses the **Which-Side** test of degree 2, since all these edges are defined by polygon vertices.

$E(\mathcal{R} \cup \mathcal{B})$ is a subset of $E(\mathcal{R}) \cup E(\mathcal{B})$. For each edge $e \in E(\mathcal{R} \cup \mathcal{B})$ contained in $E(\mathcal{R})$, $Iv(e, \mathcal{R} \cup \mathcal{B})$ is constructed incrementally from $Iv(e, \mathcal{R})$ by locating all detected intersections of e that occur on edges of $\partial\mathcal{R}$ and updating the intervals. The same is done for each edge of $E(\mathcal{R} \cup \mathcal{B})$ contained in $E(\mathcal{B})$.

Observe the intersection I of the supporting edges $e_\mathcal{R} \in E(\mathcal{R})$ and $e_\mathcal{B} \in E(\mathcal{B})$. We locate I on $Iv(e_\mathcal{R}, \mathcal{R} \cup \mathcal{B})$: I is either on an edge of $\partial\mathcal{R}$ or in the interior of $\mathcal{R}$. The same is done for $Iv(e_\mathcal{B}, \mathcal{R} \cup \mathcal{B})$. If I is located both on an edge $e'_\mathcal{R} \in \partial\mathcal{R}$ and on an edge $e'_\mathcal{B} \in \partial\mathcal{B}$, we conclude that $e'_\mathcal{R}$ and $e'_\mathcal{B}$ intersect in I and $Iv(e_\mathcal{R}, \mathcal{R} \cup \mathcal{B})$ and $Iv(e_\mathcal{B}, \mathcal{R} \cup \mathcal{B})$ are updated.

Both $Iv(e_\mathcal{R}, \mathcal{R} \cup \mathcal{B})$ and $Iv(e_\mathcal{B}, \mathcal{R} \cup \mathcal{B})$ consist of $O(n_{e_\mathcal{R}} + n_{e_\mathcal{B}})$ intervals. Hence, by binary searching, locating I is done in $O(\log(n_{e_\mathcal{R}} + n_{e_\mathcal{B}}))$ time using the **Event-Order**$'_{ii}$ test that is of degree 4. By incrementally constructing the intervals of the edges from $E(\mathcal{R} \cup \mathcal{B})$, we obtain the intersections in sorted order on the connected components of $\partial\mathcal{R}$ and $\partial\mathcal{B}$. Locating and sorting all k intersections takes $O(k \log(n_{e_\mathcal{R}} + n_{e_\mathcal{B}}))$ time.

4.6.2.3 Constructing the union

Knowing all intersections in the right order on the edges of $\partial\mathcal{R}$ and $\partial\mathcal{B}$ allows the construction of the connected components of $\partial(\mathcal{R}\cup\mathcal{B})$ that contain such intersections. Start on a not yet visited intersection and follow the red boundary in the direction that is not covered by $\mathcal{B}$. When an intersection is encountered, continue by following the blue boundary in the direction that is not covered by $\mathcal{R}$.

Continue this way, changing color at every intersection that is encountered. Mark all intersections that are encountered as visited, and mark all connected components encountered as being intersected. When the walk returns to the intersection that started it, this closes up the path that defines a connected component of $\partial(\mathcal{R}\cup\mathcal{B})$ (see figure 4.11). Do this for all yet unvisited intersections. This way, all intersecting red and blue components are handled. Since there are k intersections and $\partial\mathcal{R}$ and $\partial\mathcal{B}$ have size $O(n_{e_\mathcal{R}}+K_\mathcal{R})$ and $O(n_{e_\mathcal{B}}+K_\mathcal{B})$ respectively, constructing all these components is done in $O(k+n_{e_\mathcal{R}}+K_\mathcal{R}+n_{e_\mathcal{B}}+K_\mathcal{B})$ time.

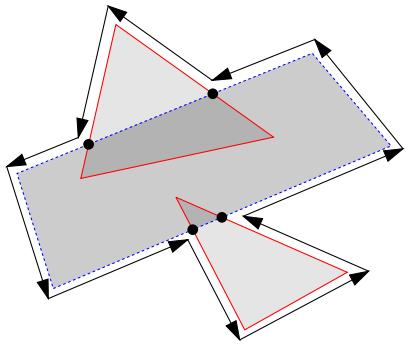


Figure 4.11: How a connected component of $\partial(\mathcal{R}\cup\mathcal{B})$ containing an intersection is constructed from $\partial\mathcal{R},\partial\mathcal{B}$ and the detected intersections.

A component of the boundary that does not intersect with the boundary of the other polygonal region is part of $\partial(\mathcal{R}\cup\mathcal{B})$ if it is outside the other polygonal region. For a red component C, this is determined as follows. Let v be a vertex of C and e a non-vertical edge incident to v. Notice that, since $\mathcal{B}$ is the union of the polygons in $\mathcal{P}_\mathcal{B}$, v is in the interior of $\mathcal{B}$ if it is in the interior of one of the polygons from $\mathcal{P}_\mathcal{B}$.

Consider the line l that supports e. For every polygon $P\in\mathcal{P}_\mathcal{B}$, we determine whether v is inside P by counting the number of intersections I of l with edges of P for which $I<_x v$. If this number is odd, v is in the interior of P. Since $\partial\mathcal{R}$ can consist of $O(n_{e_\mathcal{R}}+K_\mathcal{R})$ components, this takes $O((n_{e_\mathcal{R}}+K_\mathcal{R})n_{e_\mathcal{B}})$ time in total. The same is done in the symmetrical way for the non intersecting components of $\mathcal{B}$. Since

$\partial\mathcal{R}$ and $\partial\mathcal{B}$ have size $O(n_{e_\mathcal{R}} + K_\mathcal{R})$ and $O(n_{e_\mathcal{B}} + K_\mathcal{B})$ respectively, generating all non intersecting components is done in $O(n_{e_\mathcal{R}} + K_\mathcal{R} + n_{e_\mathcal{B}} + K_\mathcal{B})$ time.

4.6.2.4 Complexity

Summing up, the fusion algorithm runs in time

$$O(n_{e_\mathcal{R}} n_{e_\mathcal{B}} + k(\log n_{e_\mathcal{R}} + \log n_{e_\mathcal{B}}) + K_\mathcal{R} n_{e_\mathcal{B}} + K_\mathcal{B} n_{e_\mathcal{R}}).$$

Notice that for every fusion step, the cost to construct the union is smaller than the cost to localize and sort the intersections and to localize the non intersecting components.

Now consider the Divide & Conquer scheme. In total, K intersections are encountered. The following actions are counted globally:

- $O(n_e^2)$ pairs of polygon edges are tested for intersection, since, after a pair of edges is tested, both edges become part of the same polygonal region and the pair will not be tested again.
- Since any polygon edge can intersect with $O(n_e)$ other polygon edges, it is subdivided in $O(n_e)$ intervals in any node of the Divide & Conquer scheme. Hence, the K detected intersections are located and sorted in $O(K \log n_e)$ time on edges of the boundaries of polygonal regions.
- Each of the $O(n_e + K)$ non intersecting connected components is tested with at most $O(n_e)$ polygon edges.

Summing up, localizing and sorting all detected intersections and localizing all non intersecting components costs $O((n_e + K_\mathcal{A}) \log n_e)$. We have not counted the cost to construct the union in every fusion step. However, for every fusion step, the cost to construct the union is negligable, so it is also negligable in the cost of the global Divide & Conquer scheme. Hence, this degree 4 Divide & Conquer algorithm has an overall complexity of $O((n_e + K)n_e)$.

4.7 Results

The original version of the algorithm has been implemented and tested on some distinct types of input. The tests were performed on a SUN UltraSparc 1 running Solaris.[1] Tables 4.1, 4.2 and 4.3 and figures 4.13, 4.14, 4.15 and 4.16 give the results of the tests. The following types of polygons have been tested:

[1] UltraSparc and Solaris are trademarks of Sun Microsystems, Inc.

1. sets of squares generated by randomly rotating a centered square around the origin without scaling or translation. This is a reference test because *the total number of intersections between polygon edges is guaranteed to be quadratic* while *the output size is always linear.*

2. sets of polygons generated from randomly chosen pregenerated convex polygons that are scaled, rotated and translated randomly. The polygons have 4.5 vertices on average and 6 at most.

3. sets generated the same way from very nonconvex (comb-like) polygons (see figure 4.12). The polygons have 18.1 vertices on average and 25 at most.

4. sets of Minkowski sections of PrismeSat. These convex polygons have 40.7 vertices on average and 50 at most.

The figures use doubly logarithmic scale: this means that linear and quadratic curves are both straight lines and that they are distinguished by their slope: curves with the same complexity are parallel. n_e is added to the figures as a reference.

The vertices of all polygons are on a 50000×50000 integer grid. Unit size in case of the Minkowski sections is one millimeter. We have measured

- The output size s.
- $I = n_e(n_e - \frac{n_e}{n_p})$ is an approximation of the number of pairs of edges whose intersections would be tested by a naive algorithm that calculates all intersections.
- The number of intersections K encountered by the Divide & Conquer algorithm.

n_p	n_e	I	s	K	E	t_{DC}	$E_{\mathcal{A}}$
5	20	320	40	32	128	40	100
10	40	1440	80	104	**376**	110	**400**
20	80	6080	160	224	928	250	1600
50	200	39200	400	704	2992	800	10000
75	300	88800	600	1168	4944	1370	22500
100	400	158400	800	1640	7016	1920	40000
200	800	636800	1600	3640	15992	4640	160000
500	2000	$3.99 \cdot 10^6$	3940	10476	46298	12810	999965
750	3000	$8.99 \cdot 10^6$	5916	16268	73944	20670	$2.25 \cdot 10^6$
1000	4000	$1.6 \cdot 10^7$	7848	22969	102584	30199	$4.0 \cdot 10^6$
2000	8000	$6.4 \cdot 10^7$	15496	49927	224754	65336	$1.6 \cdot 10^7$
5000	20000	$4.0 \cdot 10^8$	36940	134786	624608	187304	$1.0 \cdot 10^8$
7500	30000	$9.0 \cdot 10^8$	53152	210599	974844	283260	$2.25 \cdot 10^8$
10000	40000	$1.6 \cdot 10^9$	67860	284570	$1.33 \cdot 10^6$	381130	$4.0 \cdot 10^8$
15000	60000	$3.6 \cdot 10^9$	94740	441284	$2.07 \cdot 10^6$	744700	$9.0 \cdot 10^8$
20000	80000	$6.4 \cdot 10^9$	116400	592709	$2.82 \cdot 10^6$	833110	$1.6 \cdot 10^9$

n_p	n_e	I	s	K	E	t_{DC}	$E_{\mathcal{A}}$
5	17	231.2	17	2	60	20	19
10	42	1587.6	48	8	196	50	50
20	90	7695	61	40	505	130	178
50	214	44880	70	176	**1422**	380	**1786**
75	320	101035	92	298	2357	670	3076
100	457	206761	101	372	3366	860	3883
200	882	774034	120	748	6579	1690	16782
500	2184	$4.76 \cdot 10^6$	220	2196	17702	4850	117302
750	3264	$1.06 \cdot 10^7$	234	3188	26953	7370	230204
1000	4401	$1.93 \cdot 10^7$	225	4192	35291	9250	434119
2000	8728	$7.61 \cdot 10^7$	297	8482	71042	18700	$1.66 \cdot 10^6$
5000	22109	$4.89 \cdot 10^8$	398	21772	183592	49090	$1.06 \cdot 10^7$
7500	32818	$1.08 \cdot 10^9$	399	32520	271998	72020	$2.40 \cdot 10^7$
10000	43767	$1.92 \cdot 10^9$	485	43848	365415	102030	$4.30 \cdot 10^7$
15000	65564	$4.3 \cdot 10^9$	504	65230	541923	142650	$9.63 \cdot 10^7$
20000	87256	$7.61 \cdot 10^9$	520	87705	730436	190810	$1.70 \cdot 10^8$

Table 4.1: Performances of the algorithms on sets of rotated squares (above) and pregenerated convex polygons (below) in terms of numbers of events and of calculation time. t_{DC} is the time in milliseconds CPU-time for the Divide & Conquer algorithm.

n_p	n_e	I	s	K	E	t_{DC}	$E_{\mathcal{A}}$
5	67	3591.2	103	46	280	90	117
10	167	25100	223	74	878	260	243
20	383	139355	564	476	2792	1080	1079
50	769	579534	1063	1742	8505	3280	5065
75	1442	$2.05 \cdot 10^6$	1094	3088	16068	6280	14358
100	1785	$3.15 \cdot 10^6$	1137	4256	**21380**	8380	**26749**
200	3859	$1.48 \cdot 10^7$	961	9549	48662	19050	110401
500	9326	$8.68 \cdot 10^7$	1088	22432	120263	49110	511347
750	13890	$1.93 \cdot 10^8$	733	34177	179029	72180	$1.34 \cdot 10^6$
1000	18539	$3.43 \cdot 10^8$	1043	48138	249467	104456	$2.30 \cdot 10^6$
2000	36781	$1.35 \cdot 10^9$	724	93566	485753	195100	$9.25 \cdot 10^6$
5000	90761	$8.24 \cdot 10^9$	865	229611	$1.21 \cdot 10^6$	482760	$5.46 \cdot 10^7$
7500	137052	$1.87 \cdot 10^{10}$	798	346551	$1.83 \cdot 10^6$	737610	-
10000	179864	$3.24 \cdot 10^{10}$	800	467034	$2.44 \cdot 10^6$	987500	-
15000	269664	$7.27 \cdot 10^{10}$	912	695787	$3.64 \cdot 10^6$	$1.45 \cdot 10^6$	-
20000	363132	$1.32 \cdot 10^{11}$	811	931013	$4.9 \cdot 10^6$	$1.94 \cdot 10^6$	-

Table 4.2: Performances of the algorithm on sets of pregenerated nonconvex polygons in terms of numbers of events and of calculation time.

Equipment	n_p	n_e	I	s	K	E	t_{DC}	$E_{\mathcal{A}}$
ESRI1	96	3649	$1.32 \cdot 10^7$	71	112	10898	1550	6603
ESRI2	103	3970	$1.57 \cdot 10^7$	87	157	11509	1890	7201
HORN7HST	204	5952	$3.53 \cdot 10^7$	574	418	22935	3800	13552
SVS	380	18368	$3.36 \cdot 10^8$	307	1099	68989	11110	65549
VOGO	341	12524	$1.56 \cdot 10^8$	39	987	48899	7990	44653
TVGAVO	196	9230	$8.48 \cdot 10^7$	40	423	31431	4860	22768

Table 4.3: Performances on the convex polygons representing the forbidden regions in the admissible spaces of the instruments of PrismeSat that are placed.

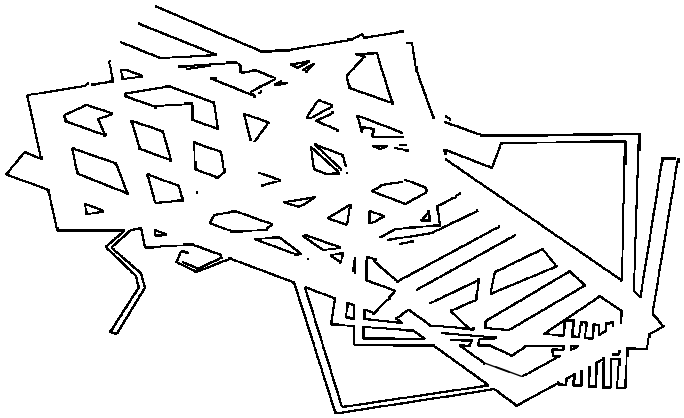


Figure 4.12: Example of the union of a set of nonconvex polygons.

- The sum of the numbers of events of all sweeps E, which is an indication of the running time of the Divide & Conquer algorithm. Notice that both a polygon and an intersection vertex can be encountered by $O(\log n_p)$ sweeps. We know that $O((K + n_p) \log^2 n_p)$ is an upper bound for E when the sizes of the polygons are bound by a constant.

- The number of events $E_{\mathcal{A}} = n_e + K_{\mathcal{A}}$ handled by a sweepline algorithm that would sweep the entire set of polygons at once, hence determining the arrangement. This is an estimation since it assumes that no intersections and polygon vertices coincide.

As can be seen from the figures, our algorithm behaves efficiently for the polygon inputs that were generated. It is more efficient than computing the entire arrangement of the edges of 10 or more rotated squares, of 50 or more convex polygons, and of 100 or more nonconvex polygons.

In the case of the rotated squares, the output size is linear, in the other two cases, it has a less than linear behaviour. The Divide & Conquer algorithm runs in $O(n_p \log n_p)$ time for the rotated squares (which supports Kedem et al.'s theoretical result for inputs where the union of any subset has linear size – see page 56) and even less for the other types of input.

The tests on sets of Minkowski slices suggest that the algorithm becomes efficient when the number of input polygons is big: E seems to have asymptotically slightly smaller behaviour than $E_{\mathcal{A}}$. The output size's behaviour is too chaotic to estimate its complexity but the Divide & Conquer algorithm runs in linear time for this input. However, the number of tested sets (6) is too small for any firm conclusions.

Anyhow, we conclude that the sizes of these sets are too small to justify the Divide & Conquer strategy: the arrangement algorithms of Section 4.3 would do better for all of the tested sets of Minkowski slices, because proportionally many of the intersections between edges are encountered by the Divide & Conquer scheme.

4.8 Conclusions

We have discussed Kedem et al.'s Divide & Conquer algorithm that calculates the union of a set of polygons. At each step, two polygonal regions are merged using a sweepline algorithm, that handles the degeneracies and numerical problems that can occur.

We have shown that any algorithm calculating the union of a set of polygons has degree 4 and we proposed a degree 4 algorithm to calculate the union of two partial unions. This algorithm has a slightly higher complexity than the sweepline algorithm.

The original algorithm was implemented and tested on sets of Minkowski slices, randomly generated convex and nonconvex polygons as well as on rotated squares, the last test serving as a reference for the others. It has shown to be very efficient for larger sets of polygons.

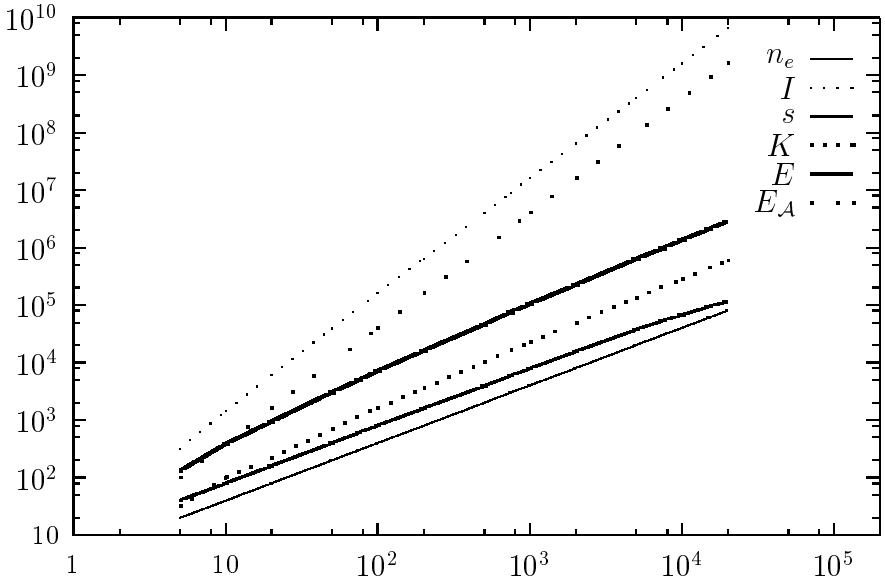


Figure 4.13: Performance of the Divide & Conquer algorithm on sets of rotated squares.

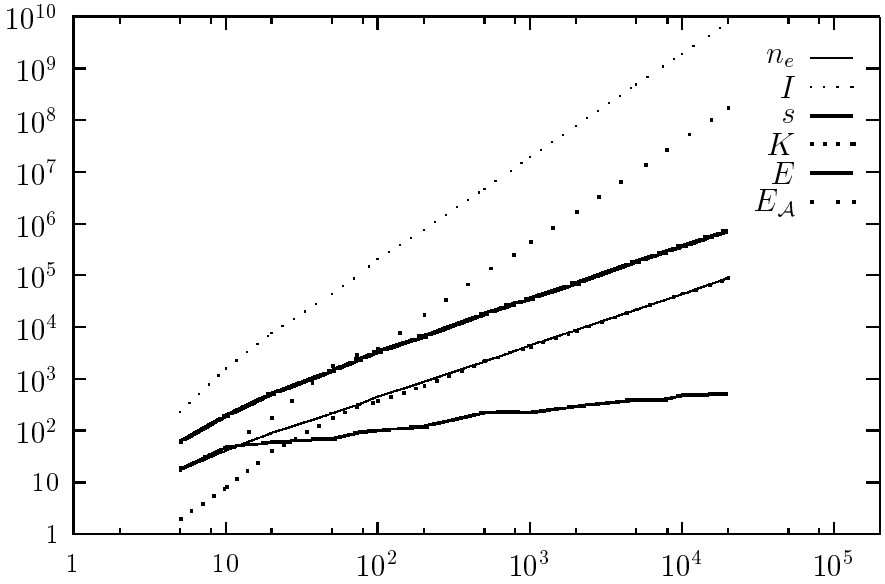


Figure 4.14: Performance of the Divide & Conquer algorithm on sets of convex polygons.

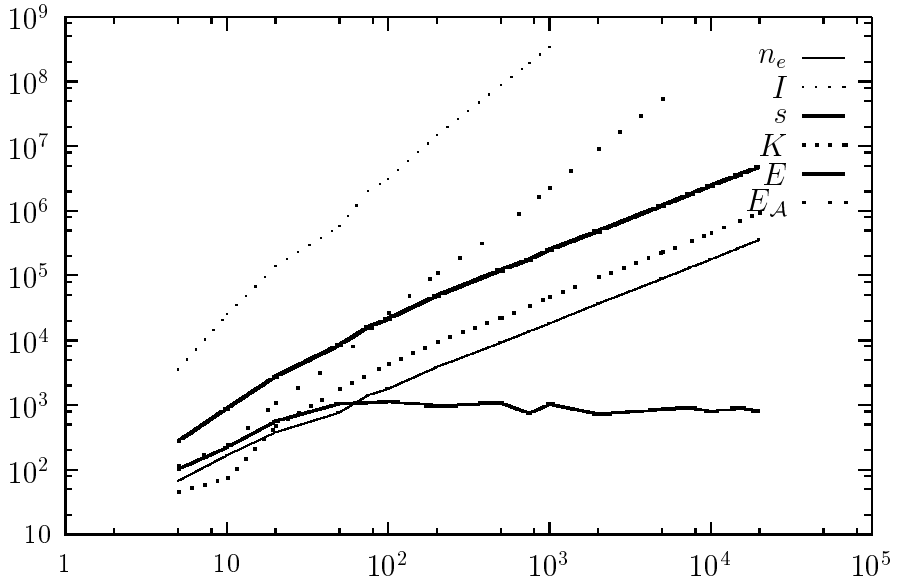


Figure 4.15: Performance of the Divide & Conquer algorithm on sets of nonconvex polygons.

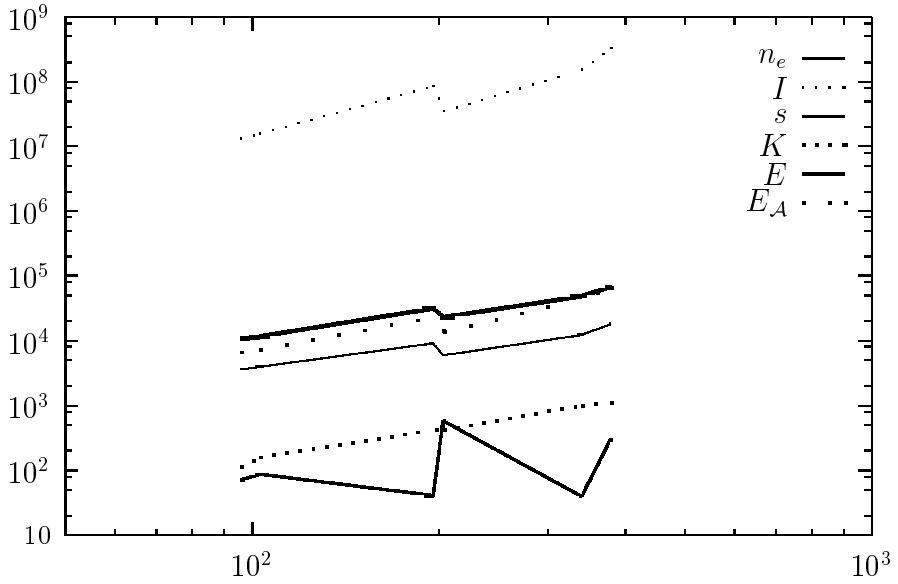


Figure 4.16: Performance of the Divide & Conquer algorithm on sets of Minkowski sections of PrismeSat.

Chapitre 5

Placement de plusieurs équipements

5.1 Introduction

Les algorithmes présentés dans les deux chapitres précédents forment la base de la méthode présentée dans le chapitre 2 qui calcule l'ensemble des placements en translation sur un plan horizontal d'un équipement dans un environnement, l'équipement et l'environnement étant composés de polyèdres convexes. La méthode permet l'aménagement interactif d'un ensemble d'équipements en les plaçant itérativement.

Ce chapitre décrit les expériences effectuées sur le placement interactif d'ensembles d'équipements et présente deux méthodes, l'une pour le placement simultané de deux équipements et l'autre pour le placement automatique d'un ensemble d'équipements.

5.2 Placement interactif

5.2.1 Logiciel

Le logiciel permet à l'opérateur de choisir un équipement à placer. Ensuite, l'espace admissible pour le placement de l'équipement sur un plan horizontal est calculé et visualisé. On peut alors choisir une configuration admissible pour l'équipement. L'espace admissible donne l'ensemble de toutes les solutions : une configuration choisie est par définition sans collision. De plus, la représentation de l'espace admissible comme un ensemble de régions permet de choisir le placement à distance controlée en vérifiant la distance de la configuration au bord de la région.

Si le placement n'est pas possible, l'opérateur peut décider d'essayer le placement

sur un plan plus élevé ou d'appliquer une rotation à l'équipement. Ensuite, le calcul de l'espace admissible est refait.

Dès que l'équipement est placé définitivement, il fait désormais partie de l'environnement et l'équipement suivant peut être placé.

Nous prenons en compte deux marges. La première modélise l'erreur maximale dans le calcul de l'espace admissible provenant du fait que les polyèdres sont des approximations facettisées de formes à surface courbée. La deuxième est la distance minimale entre composantes des équipements et l'environnement telle qu'il n'y ait pas ou suffisamment peu d'interférence (thermique, électro-magnétique et mécanique) entre ces composantes. La deuxième marge est variable et est réduite lorsque l'aménagement n'est pas possible en la respectant. Soient $\mathcal{P}_1$ et $\mathcal{P}_2$ deux équipements et m la marge à respecter. Si $\mathcal{S}_m$ est une sphère de rayon m centrée à l'origine, la frontière de $(\mathcal{P}_1 \oplus \mathcal{S}_m) \ominus \mathcal{P}_2$ donne toutes les translations de $\mathcal{P}_2$ telles que $\mathcal{P}_1$ et $\mathcal{P}_2$ soient à une distance minimale de m. Nous calculons cela en facettisant $\mathcal{S}_m$. Dans la suite, on dit que l'équipement $\mathcal{P} \oplus \mathcal{S}_m$ est l'équipement $\mathcal{P}$ *grossi de* m.

Bien que les contraintes imposées par l'environnement soient de nature 3D, les placements se font sur un plan. Auparavant, l'opérateur devait valider chaque placement en regardant le modèle de tous les angles : il vérifiait s'il n'y avait pas de collision avec l'environnement ou avec des équipements déjà placés. Ensuite, il mesurait la marge entre l'équipement et l'environnement et les équipements déjà placés. A chaque (dé)placement, ces mesures étaient refaites. Cela impliquait qu'un grand nombre d'itérations était nécessaire avant d'arriver à un placement admissible ; le nombre d'itérations dépendait de la difficulté à placer l'équipement et de l'expérience de l'opérateur.

5.2.2 Réutilisation des sections calculées

Tant que les équipements à placer sont uniquement translatés dans un plan horizontal, on peut réutiliser les sections des différences de Minkowski calculées. Soient A et B deux polyèdres convexes, $\mathcal{Z}$ un plan horizontal et t un vecteur parallèle à $\mathcal{Z}$. Alors

$$\begin{aligned}((A+t) \ominus B) \cap \mathcal{Z} &= \{a-b \mid a \in A+t, b \in B\} \cap \mathcal{Z} \\ &= \{a-b+t \mid a \in A, b \in B\} \cap \mathcal{Z} \\ &= (\{a-b \mid a \in A, b \in B\} + t) \cap \mathcal{Z} \\ &= ((A \ominus B) \cap \mathcal{Z}) + t, \ t \text{ étant parallèle à } \mathcal{Z} \qquad (5.1)\end{aligned}$$

et

$$\begin{aligned}(A \ominus B) \cap \mathcal{Z} &= \{a-b \mid a \in A, b \in B\} \cap \mathcal{Z} \\ &= \{-b+a \mid b \in B, a \in A\} \cap \mathcal{Z}\end{aligned}$$

$$= \ominus\{b - a \mid b \in B, a \in A\} \cap \mathcal{Z}$$
$$= \ominus((B \ominus A) \cap \mathcal{Z}), \text{ si le repère est choisi tq } \mathcal{Z} : Z = 0. \quad (5.2)$$

Considérons les deux équipements $\mathcal{P}_i$ et $\mathcal{P}_j$. Supposons que les opérations suivantes ont été effectuées : à un certain moment, on a placé $\mathcal{P}_j$ tandis que $\mathcal{P}_i$ était déjà placé ; ensuite, $\mathcal{P}_i$ a été déplacé. Maintenant, on veut déplacer $\mathcal{P}_j$.

Pour placer $\mathcal{P}_j$ dans l'environnement où $\mathcal{P}_i$ était présent, on avait calculé les sections des différences de Minkowski $P_i \ominus P_j$ comme dans l'équation 2.1 de la page 14. Considérons une de ces sections $M = P_i \ominus P_j$ et soit t le vecteur qui donne le déplacement de $\mathcal{P}_i$ dans le plan horizontal effectué après le placement de $\mathcal{P}_j$. Il suit de l'observation 5.1 que $((P_i + t) \ominus P_j) \cap \mathcal{Z} = M + t$. Il suffit de translater M suivant t pour obtenir la section nécessaire pour calculer l'espace admissible de $\mathcal{P}_j$.

De même, on déduit de l'observation 5.2 qu'il n'est pas nécessaire de calculer les sections des différences de Minkowski entre $\mathcal{P}_j$ et $\mathcal{P}_i$ pour déplacer $\mathcal{P}_i$ dans l'environnement contenant $\mathcal{P}_j$ si on a placé $\mathcal{P}_j$ auparavant dans un environnement contenant $\mathcal{P}_i$.

La possibilité de réutiliser les sections calculées accélère considérablement les calculs des espaces admissibles. Ceci est important lorsque l'opérateur veut explorer beaucoup de configurations des équipements et s'il veut générer des placements d'ensembles d'équipements de façon automatique (voir la section 5.4).

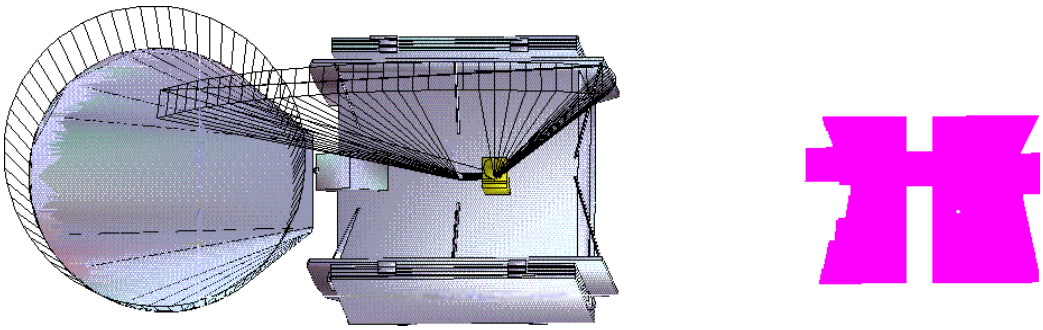
5.2.3 Résultats

Nous avons testé le logiciel sur le modèle de satellite PRISMESAT. Sur ce satellite, il y avait six instruments à placer.

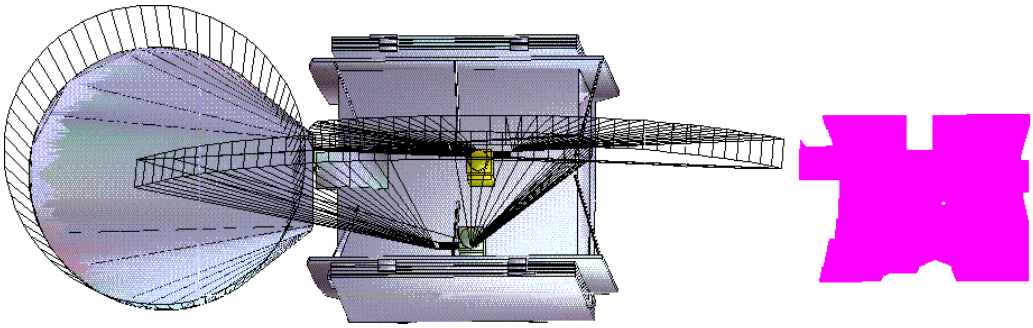
Les figures 5.1 et 5.2 donnent une séquence de placements possible. Les placements choisis prennent en compte les contraintes mécaniques. Par exemple, les deux premiers équipements placés servent à l'orientation du satellite et doivent impérativement être placés au milieu contre une paroi, ce qui est l'endroit le plus stable mécaniquement. Les trois grosses antennes qui sont placées à la fin sont placées dans des coins opposés de la plate-forme telles que le centre d'inertie du satellite soit le plus proche possible de son centre.

A chaque pas, la partie physique de l'équipement en cours de placement est dessinée en jaune. La frontière d'un champ de vision est représentée en fil de fer. L'espace admissible est dessiné en mauve à droite du modèle. Le point blanc dans l'espace admissible correspond à la configuration courante de l'équipement en cours de placement.

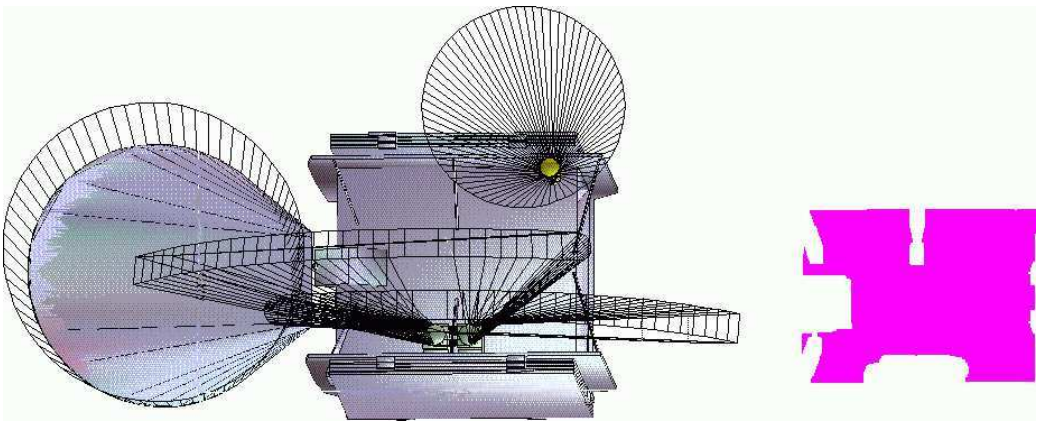
Nous redonnons les temps de calcul pour le placement des six instruments. Vu



(a)

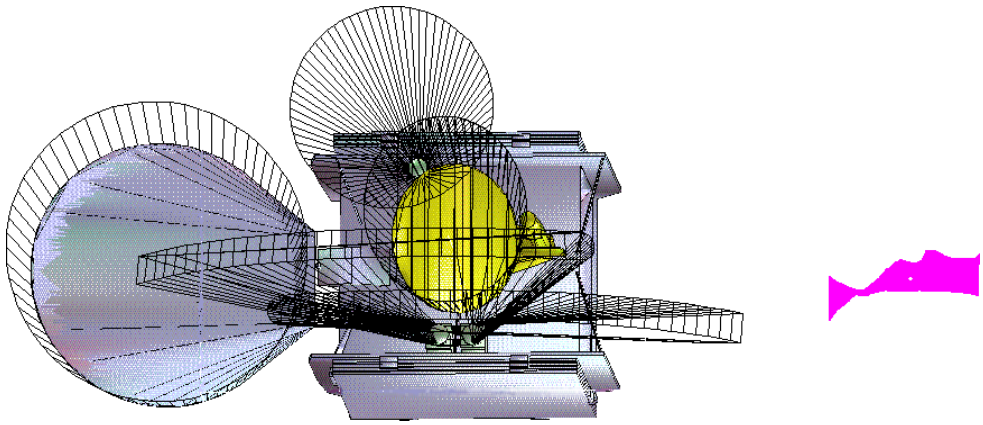


(b)

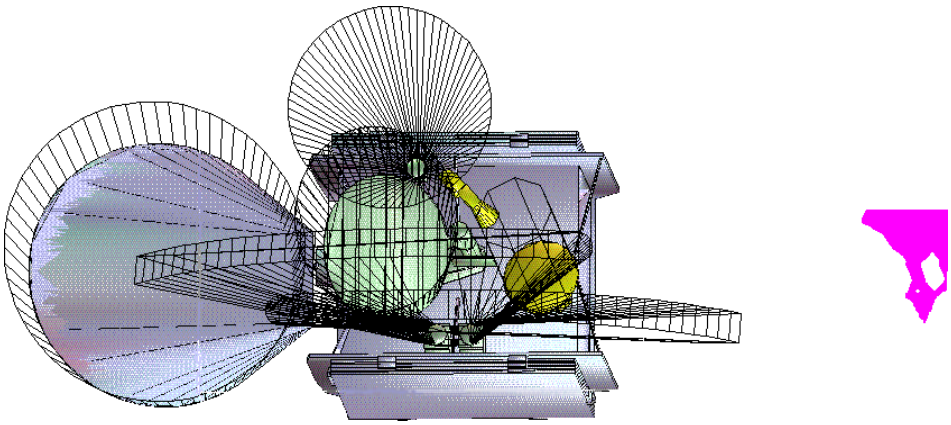


(c)

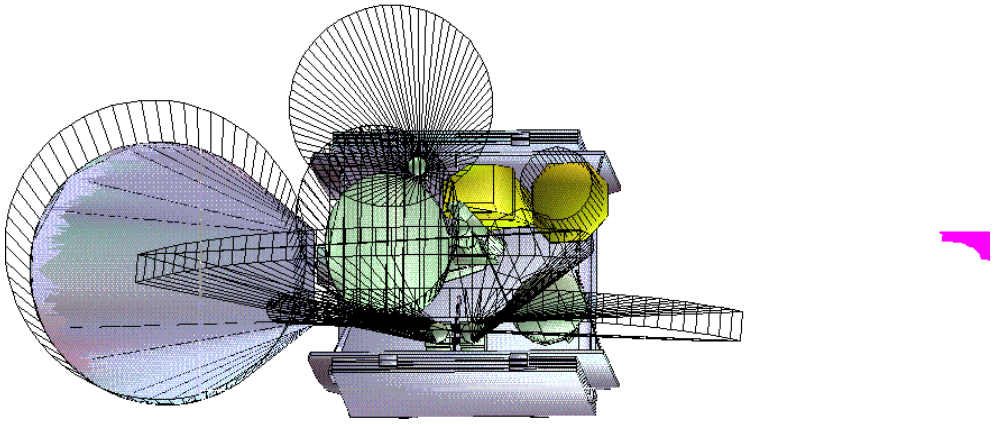
Figure 5.1: La séquence de placements effectuée pour le satellite PrismeSat : l'instrument en cours de placement est dessiné en jaune, l'espace admissible est dessiné à droite du modèle et la position courante de l'instrument apparaît dans l'espace admissible. (a) esri-1, (b) esri-2, (c) horn-7-hst.



(d)



(e)



(f)

Figure 5.2: La séquence de placements effectuée pour PRISMESAT (suite). (d) SVS-ANTENNA, (e) VOGO, (f) TVGAVO-ENS.

Instrument	Calcul sections	Calcul union	Placement
ESRI1	10101	1550	11651
ESRI2	10383	1890	12273
HORN7HST	38676	3800	42476
SVS	136309	11110	147409
VOGO	113931	7990	146060
TVGAVO	45281	4860	58819
Total	354681	64007	418688

Table 5.1: Temps de calcul pour le calcul des espaces admissibles des 6 instruments du modèle PrismeSat.

que les placements ont été refaits sur une autre machine, les temps de calcul ne correspondent pas à ceux des chapitres précédents. Nous séparons les temps en calcul des sections des différences de Minkowski et calcul de l'union des sections. Les tests ont été faits sur un SUN Ultrasparc1 ; les temps sont donnés en millisecondes de temps CPU. Les sections étant réutilisables, un déplacement d'un instrument déjà placé ne coûte donc que le temps de calculer l'union des sections déjà calculées. La table 5.1 donne les temps des premiers placements des 6 équipements.

Bien que le placement d'*un* équipement soit direct, une suite de placements admissibles pour un sous ensemble des équipements ne garantit pas que les autres équipements soient encore plaçables. En pratique, un certain nombre de retours en arrière est nécessaire pour déterminer où placer les premiers équipements pour ne pas arriver à un blocage à la fin. Plus l'aménagement est contraint, plus le nombre d'itérations nécessaires est grand. Cela dépend aussi de l'expérience de l'opérateur (c'est-à-dire sa capacité de raisonner en avance). Sur le modèle de PrismeSat, nous avons du faire une vingtaine de retours en arrière avant d'obtenir un placement des 6 équipements. Aucun changement de hauteur d'un équipement, ni de rotation ont du être appliqués.

5.3 Placement simultané de deux équipements

5.3.1 Méthode

Avnaim et Boissonnat proposent une méthode qui calcule l'espace admissible pour le placement simultané en translation de deux ensembles $\mathcal{P}_1$ et $\mathcal{P}_2$ dans un environnement $\mathcal{E}$. [AB87, Avn89]. La méthode consiste à calculer

1. l'espace admissible $\mathcal{A}(\mathcal{E}, \mathcal{P}_1)$ pour le placement de $\mathcal{P}_1$ dans $\mathcal{E}$,

2. l'espace admissible $\mathcal{A}(\mathcal{E}, \mathcal{P}_2)$ pour le placement de $\mathcal{P}_2$ dans $\mathcal{E}$,

3. l'espace admissible relatif $\mathcal{A}(\mathcal{P}_1, \mathcal{P}_2)$ pour le placement de $\mathcal{P}_2$ par rapport à $\mathcal{P}_1$.

Le dernier espace admissible est défini comme l'ensemble des configurations admissibles pour $\mathcal{P}_2$ dans l'environnement uniquement composé de $\mathcal{P}_1$ fixe.

La somme de Minkowski $\mathcal{A}_2(\mathcal{E}, \mathcal{P}_1, \mathcal{P}_2) = \mathcal{A}(\mathcal{E}, \mathcal{P}_2) \oplus \mathcal{A}(\mathcal{P}_1, \mathcal{P}_2)$ donne l'ensemble des configurations de $\mathcal{P}_1$ qui garantissent un placement admissible de $\mathcal{P}_2$. L'intersection de $\mathcal{A}_2(\mathcal{E}, \mathcal{P}_1, \mathcal{P}_2)$ avec $\mathcal{A}(\mathcal{E}, \mathcal{P}_1)$ donne l'ensemble des configurations *admissibles* pour $\mathcal{P}_1$ qui garantissent un placement admissible de $\mathcal{P}_2$. Voir l'exemple de la figure 5.3.

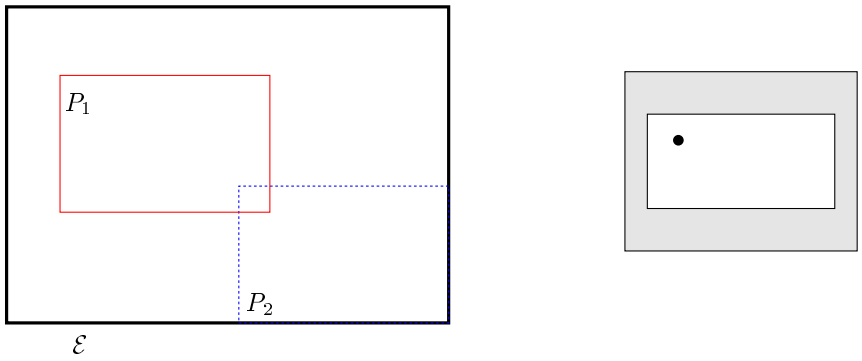


Figure 5.3: Exemple de placement simultané. Les deux rectangles P_1, P_2 doivent être placés dans le rectangle $\mathcal{E}$. A gauche, un placement de P_1. A droite, l'espace admissible pour P_1 réduit des placements non admissibles relatifs de P_2 (l'espace admissible pour P_1 en translation est en gris, le point indique la configuration actuelle de P_1). La configuration choisie pour P_1 empêche le placement de P_2.

Placer deux équipements séparément peut s'avérer très inefficace : l'opérateur doit déplacer le premier équipement jusqu'à ce que le deuxième puisse être placé. Ceci nécessite beaucoup d'itérations lorsque les équipements sont très contraints. Avec le placement simultané, l'opération est *immédiate* pour les deux équipements.

Avnaim montre aussi comment 3 polygones peuvent être placées simultanément dans un *parallélogramme* et, malheureusement, que la formule ne peut pas être généralisée pour plus de 2 polygones dans un environnement polygonal.

5.3.2 Mise en œuvre

Nous pouvons appliquer cette méthode au cas du placement en translation de 2 équipements. Elle nécessite le calcul de la somme de Minkowski des régions polygonales $\mathcal{A}(\mathcal{E}, \mathcal{P}_2)$ et $\mathcal{A}(\mathcal{P}_1, \mathcal{P}_2)$. Nous l'avons implémentée de la manière suivante :

- décomposer les régions polygonales $\mathcal{A}(\mathcal{E}, \mathcal{P}_2)$ et $\mathcal{A}(\mathcal{P}_1, \mathcal{P}_2)$ en trapèzes,
- calculer les sommes de Minkowski $T_1 \oplus T_2$ de toutes les paires de trapèzes $(T_1, T_2) \in \mathcal{A}(\mathcal{E}, \mathcal{P}_2) \times \mathcal{A}(\mathcal{P}_1, \mathcal{P}_2)$,
- calculer l'union de toutes les sommes $T_1 \oplus T_2$.

Ensuite, on calcule l'intersection de cette union avec $\mathcal{A}(\mathcal{E}, \mathcal{P}_1)$. Remarquer qu'une triangulation décomposerait les régions polygonales en moins de polygones : le calcul des sommes de Minkowski des paires de polygones serait donc plus rapide. Une triangulation peut être déterminée efficacement à partir des trapèzes [BY95].

La figure 5.4 montre l'espace admissible pour le placement de l'équipement SVS dans l'environnement composé uniquement de la plate-forme et son espace admissible contraint par l'équipement TVGAVO.

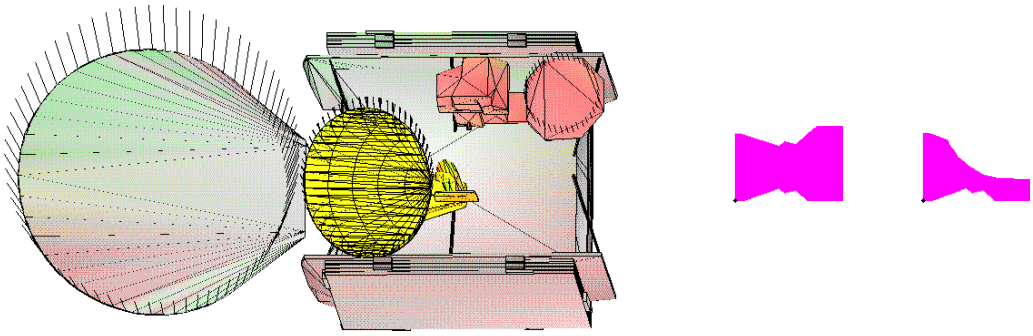


Figure 5.4: Le placement simultané des deux équipements SVS (jaune) et TVGAVO (rouge). A gauche, le placement de SVS. Au milieu, $\mathcal{A}(\mathcal{E}$,SVS) et à droite, $\mathcal{A}(\mathcal{E}$,SVS) contraint par $\mathcal{A}_2(\mathcal{E}$,SVS,TVGAVO).

On pourrait penser que le calcul des sommes de toutes les paires de trapèzes rende la méthode assez lente. En pratique, ça marche dans un temps raisonnable, du fait que les espaces admissibles ne sont pas très fragmentés et sont donc pas de petites tailles. La table 5.2 donne les temps pour contraindre quelques espaces admissibles. Les tests ont été faits sur un SUN Ultrasparc1 ; les temps sont donnés en millisecondes de temps CPU. Les temps pour calculer les sections ne sont pas pris en compte : on suppose que toutes les sections ont été précalculées.

$\mathcal{P}_1$	$\mathcal{P}_2$	**Temps de calcul**				$\mathcal{A}_2(\mathcal{E},\mathcal{P}_1,\mathcal{P}_2)\cap$
		$\mathcal{A}(\mathcal{E},\mathcal{P}_1)$	$\mathcal{A}(\mathcal{E},\mathcal{P}_2)$	$\mathcal{A}(\mathcal{P}_1,\mathcal{P}_2)$	$\mathcal{A}_2(\mathcal{E},\mathcal{P}_1,\mathcal{P}_2)$	$\mathcal{A}(\mathcal{E},\mathcal{P}_1)$
ESRI-1	SVS	1550	9749	32	5230	31
SVS	TVGAVO	9884	3981	60	4870	29
SVS	VOGO	9749	6243	48	17950	32
ESRI-2	HORN7	1563	1988	22	14090	36
HORN7	VOGO	3577	7184	27	15180	32

Table 5.2: Performances pour quelques placements simultanés. $\mathcal{P}_1$ est l'équipement en cours de placement, $\mathcal{P}_2$ l'équipement qui contraint son espace admissible. Les temps sont donnés pour calculer $\mathcal{A}(\mathcal{E},\mathcal{P}_1)$, $\mathcal{A}(\mathcal{E},\mathcal{P}_2)$, $\mathcal{A}(\mathcal{P}_1,\mathcal{P}_2)$, $\mathcal{A}_2(\mathcal{E},\mathcal{P}_1,\mathcal{P}_2)$ et l'intersection du premier avec le dernier. Le calcul des sections des sommes de Minkowski n'est pas pris en compte dans les temps.

5.4 Placement automatique

5.4.1 Méthode

Le placement interactif des équipements peut aussi être fait automatiquement. A chaque étape i, un équipement $\mathcal{P}_i$ est choisi qui n'est pas encore placé. Son espace admissible est calculé et une configuration est choisie. Evidemment, l'espace admissible étant une région polygonale, il y a une infinité de configurations admissibles : un petit ensemble C_i de configurations est déduit de l'espace admissible (par exemple, on utilise les sommets de sa frontière) et une configuration $c_i \in C_i$ est choisie.

Ensuite, les autres équipements $\mathcal{P}_{i+1} \ldots \mathcal{P}_n$ non encore placés sont traités de la même façon itérativement. Si un placement n'a pas pu être trouvé pour l'ensemble de ces équipements (ou si l'on veut générer plusieurs solutions), une autre configuration $c_i' \in C_i$ est choisie pour $\mathcal{P}_i$ et le processus est répété (retour-arrière), jusqu'à ce que toutes les configurations dans C_i aient été explorées ou une solution soit trouvée.

Soit n le nombre d'équipements à placer. La recherche peut être vue comme le parcours d'un arbre $\mathcal{T}$, où :

- Le racine (profondeur 0) représente l'aménagement vide.
- Un nœud de profondeur $i < n$ est un placement pour les équipements $1 \ldots i$. Si, dans cette solution partielle, l'équipement $i+1$ ne peut pas être placé (l'aménagement est impossible), le nœud est une feuille.

- Une feuille de profondeur n est un placement pour les équipements $1 \ldots n$ et donc une solution complète.

On ne parcourt tout l'arbre que si on cherche toutes les solutions.

Pour trouver rapidement une solution, il faut appliquer des heuristiques pour essayer de réduire la taille de $\mathcal{T}$; ces heuristiques portent sur l'ordre dans lequel les équipements sont placés et sur le choix d'une configuration pour chaque équipement. On essaie donc de minimiser le nombre de nœuds de $\mathcal{T}$. On veut egalement savoir le plus rapidement possible si une solution partielle mène à une solution complète : cela impose que les feuilles représentant des aménagements impossibles sont à une profondeur faible dans $\mathcal{T}$.

Il nous semble que la meilleure façon de réduire l'espace de recherche est de placer d'abord les équipements les plus encombrants, ou dont la superficie de l'espace admissible est la plus petite. L'équipement contraint fortement les espaces admissibles des équipements restant à placer.

Ensuite, pour chaque équipement, il faut extraire de l'espace admissible, qui est constitué d'une infinité de configurations, un ensemble de configurations prometteuses, c'est-à-dire qui semblent mener à une solution.

Le plus simple est d'utiliser les sommets de la frontière de l'espace admissible. Les sommets représentent des configurations de l'équipement à distance m aux objets de l'environnement et aux équipements déjà placés, où m est la marge introduite dans la section 5.2.1.

Une recherche qui utilise uniquement des sommets des frontières des espaces admissibles peut ne pas mener à une solution : ceci est le cas lorsqu'il n'existe pas de placement admissible pour lequel un des objets est en double contact avec le conteneur. Un exemple est donné dans la figure 5.5.

L'exemple a des double contacts entre les polygones. Nous allons montrer que, dans ce cas, la combinaison du placement automatique avec le placement simultané de deux objets trouvera la solution. Considérons les deux polygones P_1, P_2 de l'exemple en double contact et supposons qu'aucun polygone n'a été placé :

1. Une arête e'_1 sur la frontière de l'espace admissible $\mathcal{A}(\mathcal{E}, P_1)$ représente tous les contacts de P_1 avec l'arête e_1 du conteneur.

2. Une arête e'_2 sur la frontière de l'espace admissible $\mathcal{A}(\mathcal{E}, P_2)$ représente tous les contacts de P_2 avec l'arête e_2 du conteneur.

3. Dans l'espace admissible relatif $\mathcal{A}(P_1, P_2)$, il existe un sommet p qui représente le double contact entre P_1 et P_2.

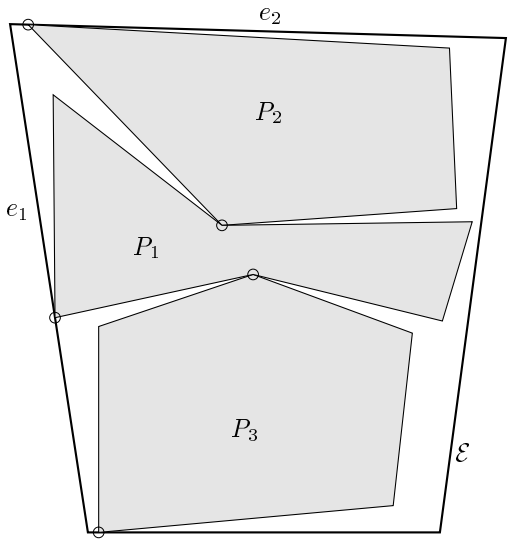


Figure 5.5: Exemple d'un ensemble de trois polygones où le placement du premier, quel qu'il soit, en double contact avec le conteneur ne mène pas à une solution pour le placement des 3 polygones.

4. $\mathcal{A}_2(\mathcal{E}, P_1, P_2) = \mathcal{A}(\mathcal{E}, P_2) \oplus \mathcal{A}(P_1, P_2)$ contient l'arête $e_2'' = e_2' \oplus p$ qui représente toutes les translations de P_1 telles que P_2 puisse être en double contact avec P_1 et en contact avec e_2 en même temps.

5. Dans $\mathcal{A}(\mathcal{E}, P_1) \cap \mathcal{A}_2(\mathcal{E}, P_1, P_2)$, il existe un sommet $e_1' \cap e_2''$ représentant la translation pour P_1 telle qu'il soit en contact avec e_1 et que P_2 puisse être en double contact avec P_1 et en contact avec e_2 en même temps.

En conséquence, la méthode de placement simultané de deux objets permet de résoudre quelques problèmes de placement liés au fait que les sommets des espaces admissibles sont utilisés comme configurations. Ceci ne résoud pas tous les problèmes : l'intersection $e_1' \cap e_2''$ ne paraitrait pas sur la frontière de $\mathcal{A}(\mathcal{E}, P_1) \cap \mathcal{A}_2(\mathcal{E}, P_1, P_2)$ si P_2 était plaçable en dessous de P_1. Devillers donne même un exemple de trois polygones où *aucun* double contact, ni entre conteneur et polygone, ni entre paires de polygones n'existe [Dev93]. Remarquer d'ailleurs que la méthode de placement simultané de 2 équipements contraint le placement d'un équipement et peut donc aussi aider à réduire la taille de $\mathcal{T}$.

5.4.2 Mise en œuvre

Nous avons fait une première implémentation de la stratégie de placement itératif automatique. L'implémentation place une suite d'équipements dans un ordre donné et n'utilise que des sommets des frontières des espaces admissibles. Les polyèdres sont grossis de la marge m pour le calcul des espaces admissibles. Le fait que les polygones calculés sont réutilisés rend l'implémentation assez rapide.

Toutes les sections des différences de Minkowski ont été calculées auparavant. Pour la séquence des 6 équipements de PrismeSat, on a pu générer 8 solutions visiblement différentes en 46 secondes en plaçant les équipements dans l'ordre de taille décroissante (noter que c'est une autre ordre que celle utilisée pour le placement interactif).

Au total, la méthode a exploré 15 feuilles (c'est-à-dire, des solutions complètes et des aménagements partiels où l'équipement suivant ne pouvait pas être placé). La profondeur moyenne des feuilles représentant des solutions impossibles était 1.14 : dans la plupart des cas, il a donc suffi de placer un équipement pour constater que l'aménagement était impossible. 4 des solutions trouvées sont montrées dans la figure 5.6.

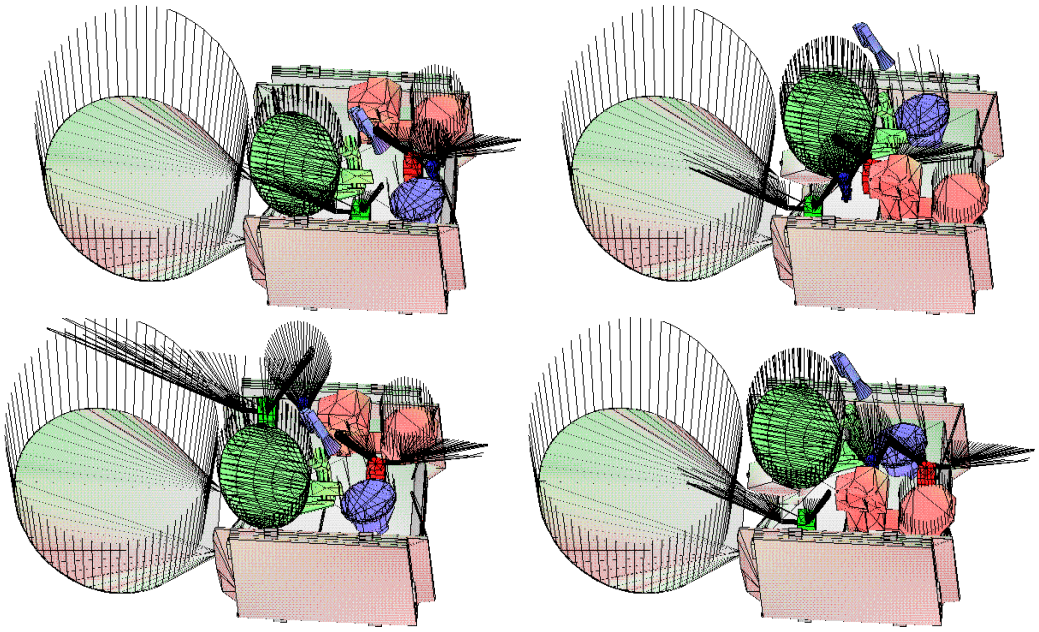


Figure 5.6: Quatre solutions générées par le placement automatique pour PrismeSat.

La plupart des aménagements impossibles a été détectée lors du calcul de l'espace admissible du deuxième équipement. Tous ces aménagements peuvent donc être éli-

minés en plaçant simultanément les deux premiers équipements. Effectivement, en appliquant cette méthode sur les premiers deux équipements, la méthode a pu trouver 8 solutions en n'inspectant que 11 feuilles. La profondeur des feuilles représentant des solutions impossibles était toujours 2, c'est-à-dire, le troisième équipement ne pouvait pas être placé. Dans ce cas particulier, le gain de temps de recherche était à peu près égal au surcoût du calcul nécessaire pour le placement simultané.

A titre de référence, les placements ont été faits dans l'ordre de taille croissante (ce qui est l'ordre dans lequel les équipements ont été placés lors du placement interactif). Après 15 heures et 30 minutes, 42114 feuilles avaient été explorées sans trouver aucune solution. La profondeur moyenne des feuilles était 4.93 : il a fallu presque toujours placer 5 des 6 équipements pour constater que l'aménagement était impossible. Le placement des grands équipements d'abord s'avère donc une bonne heuristique pour réduire le nombre de configurations à explorer.

On ne peut pas gagner beaucoup en efficacité en appliquant le placement simultané ici : les premiers équipements placés sont de petite taille donc ils ne se contraignent pas ou pas beaucoup. Il semble que l'application du placement simultané dans un schéma de placement automatique est plutôt complémentaire que compensatoire.

Indépendemment de l'ordre de placement des équipements, le problème des solutions trouvées est qu'elles ne prennent pas en compte le centrage de la masse et de l'inertie et les contraintes mécaniques, les dernières étant difficilement modélisables géométriquement. Pour bien centrer, plusieurs équipements doivent être placés simultanément. Le placement simultané de deux équipements permettra déjà de placer les équipements deux par deux.

5.5 Perspectives

Nous présentons ici quelques méthodes qui n'ont pas été mises en œuvre. Celles-ci amélioraient l'aide à l'opérateur et optimiseraient le placement automatique.

Cercles inscrits. Dans le calcul de l'espace admissible, on prend en compte la marge minimale m à respecter entre les équipements et l'environnement. L'utilisation des sommets de la frontière de l'espace admissible mène donc à des aménagements qui respectent la marge minimale mais sans rien de plus.

On pourrait *Maximiser* la marge en choisissant des configurations définies par les sommets du diagramme de Voronoï (ou par des points sur les arêtes du diagramme de Voronoï) intérieurs des composantes connexes de l'espace admissible. De tels points correspondent à des configurations qui sont localement à distance maximale (en translation dans le plan) avec les polyèdres de l'environnement et les équipements

déjà placés.

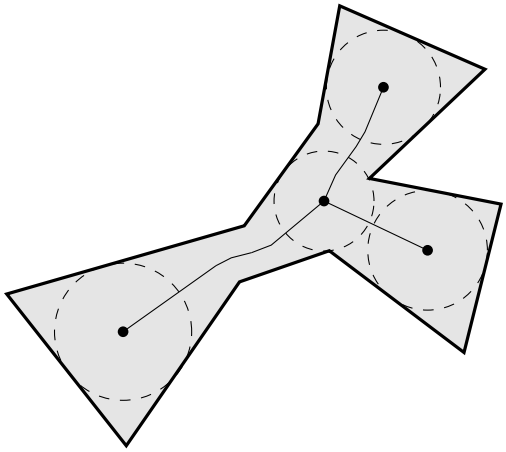


Figure 5.7: Les sommets et arètes de Voronoï intérieurs des régions de l'espace admissible définissent des configurations qui maximisent la distance minimale entre l'équipement et objets de l'environnement.

Les centres des cercles inscrits ne doivent pas être utilisés systématiquement car cela fragmenterait l'espace libre, ce qui peut empêcher le placement d'autres équipements. Pendant un placement interactif, les centres des cercles servent uniquement à titre indicatif pour l'opérateur : c'est lui qui choisit la configuration.

Raisonnement un pas en avant. Le placement simultané de deux équipements permet le raisonnement *un pas en avant* : lorsque l'équipement $\mathcal{P}_i$ est placé, on prendrait l'intersection de $\mathcal{A}(\mathcal{E}, \mathcal{P}_i)$ avec les espaces admissibles $\mathcal{A}_2(\mathcal{E}, \mathcal{P}_i, \mathcal{P}_j)$ de tous les équipements $\mathcal{P}_j, i < j \leq n$ qui ne sont pas encore placés. Cela garantit qu'après placement de $\mathcal{P}_i$, *n'importe quel équipement* $\mathcal{P}_j$ *suivant* peut être placé.

Placement à nombre faible de collisions. L'espace admissible est le complémentaire de l'union des sections des sommes de Minkowski. Considérons l'arrangement des arêtes de toutes les sections et considérons une cellule qui a une arête de la frontière de l'espace admissible sur son bord (supposons position générale). Chaque configuration contenue dans cette cellule mettra l'équipement dans l'environnement tel qu'il y ait une collision entre exactement deux polyèdres, l'un appartenant à l'équipement et l'autre à l'environnement ou à un équipement déjà placé.

Appelons une telle cellule une cellule de niveau 1. Appelons chaque cellule qui

n'est pas de niveau 1 mais qui est incidente à une cellule de niveau 1 une cellule de niveau 2 et ainsi de suite.

On pourrait calculer et visualiser l'ensemble des cellules de niveau $i = 1$ (i=2,...) qui représentent des placements *i-admissibles*. De plus, en étiquetant les cellules, on saurait avec quel équipement il y a collision.

Citons aussi la méthode de compactage de Milenkovic [DML94], qui permet le placement approché d'un ensemble de polygones où une pénétration maximale d'un ϵ spécifiable est permise.

Spécification du domaine. Pour modéliser certaines contraintes physiques, on pourrait imposer des zones interdites pour chaque équipement (zones pas suffisamment solides et caetera) ou au contraire des zones obligatoires. Cela servirait à l'opérateur mais aussi au placement automatique pour éliminer des placements insouhaitables.

Chapitre 6

Conclusion

Dans cette thèse, nous nous sommes intéressés à des problèmes d'aménagement de satellite. Nous avons conçu et mis en œuvre deux algorithmes géométriques efficaces qui ont permis la réalisation d'un logiciel d'assistance à l'aménagement d'équipements sur un satellite. Ce logiciel a servi à beaucoup d'expérimentations afin de tester l'efficacité de notre approche au placement d'équipements.

Résultats théoriques

Un algorithme optimal qui calcule directement une section plane de la somme de Minkowski de deux polyèdres convexes a été conçu et mis en œuvre. L'implémentation de cet algorithme a été testée sur des polyèdres provenant de modèles de satellite et s'est avérée très efficace. Cela est très important lorsque les polyèdres sont des approximations polyédriques très fines de formes à surface courbe.

Un algorithme qui calcule l'union d'un ensemble de polygones en appliquant un schéma de division et fusion a été analysé et implémenté. Bien que non sensible à la sortie dans le cas le pire, l'algorithme est efficace en pratique sur des grands ensembles de polygones, où un algorithme qui calcule l'arrangement des arêtes des polygones et en déduit l'union des polygones le serait beaucoup moins.

Nous avons attentivement analysé les dégénérescences géométriques qui peuvent être rencontrées lors de l'exécution des deux algorithmes et elles sont toutes traitées. La précision numérique nécessaire pour l'exécution correcte des deux algorithmes a été analysée et nous garantissons que les calculs faits dans l'implémentation sont exacts, ce qui rend les implémentations entièrement robustes.

Le probleme UNION-DE-POLYGONES est de degré algébrique 4 ; nous pouvons

déduire l'union d'un ensemble de polygones de l'arrangement des arêtes des polygones. Nous avons donné un algorithme naïf à degré minimal de fusion de régions polygonales : ceci prouve qu'un algorithme de degré 4 existe pour calculer l'union par division et fusion. Néanmoins, sa complexité est plus mauvaise que celle de l'algorithme de début. L'existence d'un algorithme de faible degré, par balayage ou autre, plus efficace que l'algorithme naïf, est un problème ouvert.

Résultats expérimentaux

Le calcul d'une section d'une somme de Minkowski et le calcul de l'union d'un ensemble de polygones forment la base d'un logiciel qui calcule l'espace admissible pour le placement d'un équipement sur un plan dans un environnement, tous les deux composés de polyèdres convexes. Le logiciel permet ainsi l'aménagement d'une plate-forme d'un satellite en plaçant les équipements un par un sur la plate-forme.

Le calcul de l'espace admissible est rapide. De plus, la mémorisation des sections des sommes de Minkowski calculées permet de refaire le calcul de l'espace admissible très rapidement, ce qui est important lorsqu'on souhaite explorer beaucoup de configurations, interactivement aussi bien qu'automatiquement.

La visualisation de l'espace admissible lors de chaque placement d'équipement permet à l'opérateur d'aménager une plate-forme de satellite avec beaucoup moins d'itérations que lorsqu'il effectuerait l'aménagement manuellement : la validité d'une configuration choisie dans l'espace admissible est garantie et la configuration ne doit donc pas être vérifiée.

Une formule en termes d'opérations ensemblistes permet de calculer l'espace admissible pour placer des paires équipements simultanément. Ceci rend immédiat le placement de deux équipements encombrants, difficile à faire manuellement, ce qui a été vérifié expérimentalement.

Une stratégie automatique a été implémentée pour placer un ensemble d'équipements itérativement avec retour-arrière pour résoudre les blocages. La stratégie a trouvé rapidement un jeu de solutions pour le modèle de satellite testé. Par contre, pour trouver des solutions de bonne qualité, il faudra bien prendre en compte certaines contraintes physiques que nous n'avons pas modélisées géométriquement.

Perspectives

Un algorithme pourrait être mis en œuvre pour calculer efficacement des sections de la somme de Minkowski M de deux polyèdres convexes A, B à partir d'une section $M \cap \mathcal{Z}$ déjà calculée. En effet, une section $M \cap \mathcal{Z}'$ proche de $M \cap \mathcal{Z}$, où $\mathcal{Z}'$ est parallèle à $\mathcal{Z}$, pourrait être déterminée efficacement à partir de cette dernière section

en balayant M de $\mathcal{Z}$ à $\mathcal{Z}'$ par un plan. Cela implique que l'espace admissible pour un équipement à une hauteur h pourrait être calculé rapidement si l'on avait déjà essayé de placer l'équipement à la hauteur $h + \epsilon$ ou $h - \epsilon$, où ϵ est petit. Plus précisement, l'efficacité dépendra du nombre de sommets de M contenus entre $\mathcal{Z}$ et $\mathcal{Z}'$.

Le calcul de l'espace admissible n'est que la base d'une gamme d'outils assistant l'opérateur à ses tâches d'aménagement. Nous envisageons surtout des outils interactifs qui aident encore plus l'opérateur à satisfaire les contraintes physiques de l'aménagement.

Le calcul des centres des cercles inscrits à l'intérieur d'un espace admissible permettrait à l'opérateur de repérer les placements à distance maximale du reste de l'aménagement. On pourrait mettre en œuvre le calcul de l'espace des configurations à faible nombre de collisions pour indiquer à quel endroit le placement d'un équipement est presque possible et quel(s) équipement(s) il faudrait alors déplacer pour qu'il le soit vraiment.

On envisage l'application du placement simultané de deux équipements afin d'aider à satisfaire la contrainte de centrage d'inertie. D'autre part, la spécification d'un domaine pour chaque équipement permettrait au placement automatique d'éviter des solutions dont on sait qu'elles sont mécaniquement impossibles.

Annexe A

GÉOTOOLS

Cette annexe contient une description de GEOTOOLS/T3D et une version résumée de son manuel d'utilisation.

A.1 Introduction

A.1.1 Le programme

GEOTOOLS/T3D permet le placement d'équipements sur une paroi d'un satellite en visualisant les contraintes géométriques : les collisions entre équipements et le respect des champs de vision des antennes.

A.1.2 L'interface homme-machine

Le programme utilise l'interface homme-machine T3D. Il a donc le même *"look-and-feel"* que les autres outils de la gamme SYSTEMA.

A.1.3 Les données géométriques du modèle

Le modèle du satellite est stocké en format `SYSEXP`, le format ascii interchangeable de l'outil d'analyse SYSTEMA. Une interface automatique existe pour convertir un modèle EUCLID en format `SYSEXP` et un fichier `SYSEXP` est lisible par EUCLID.

Le programme prend uniquement en considération la description polyédrique des

objets du modèle et non pas leurs formes analytiques. Le programme prend l'enveloppe convexe de tous les éventuels polyèdres non convexes. Cependant, il prend en compte des faces individuelles convexes.

Exemple 1. Une paroi avec des trous peut être modélisée comme un ensemble de faces convexes.

Exemple 2. Un équipement en forme de boite avec un connecteur en forme de boite est modélisé par deux objets convexes.

Exemple 3. Une antenne parabolique est creuse. Mais rien n'est perdu en l'enveloppant : un objet placé dans la cavité occulterait le champ de vision de l'antenne.

Exemple 4. Une coiffe de lanceur est creuse aussi. Mais ici, il faudra décomposer la forme en objets convexes : on veut justement placer le satellite *à l'intérieur* de la coiffe.

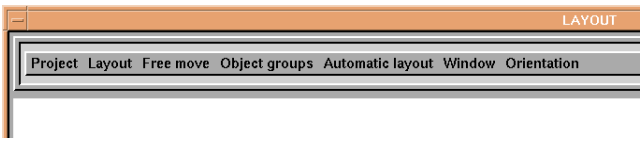
Remarque. SYSTEMA permet la modélisation hiérarchique (système, sous-système, équipement, composante) : chaque nœud de la hiérarchie représente un objet du modèle. Chaque tel objet peut avoir une définition géométrique : une ou plusieurs formes sont alors attachées à cet objet. Une forme est une description analytique plus le polyèdre qui approche la forme géométrique.

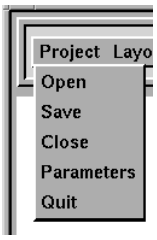
Un objet SYSTEMA avec plusieurs formes est converti en un ensemble d'objets par GEOTOOLS, en préservant la hiérarchie, c'est-à-dire les nouveaux objets seront les enfants de l'objet original. Lors de la conversion du modèle EUCLID, il faut faire attention qu'exactement un polyèdre ou une face est mis dans une forme SYSTEMA : le bon fonctionnement de GEOTOOLS ne peut pas être garanti autrement.

A.2 Manuel de référence

A.2.1 Les menus

Les menus sont accessibles à partir du *Menubar* :





Project

Gestion du projet d'aménagement.

Open Charger un modèle `toto.SYSEXP` (voir la figure A.1a). Un fichier `toto.lp` est attaché la première fois le modèle est chargé (création d'un projet d'aménagement). Il contient les paramètres et les informations du projet. Un fichier `toto.geom` contiendra les données géométriques calculées, afin qu'elles puissent être réutilisées (accélération du calcul). Après l'ouverture des fichiers, un "browser" est affiché qui visualise la structure hiérarchique du modèle (voir la figure A.2).

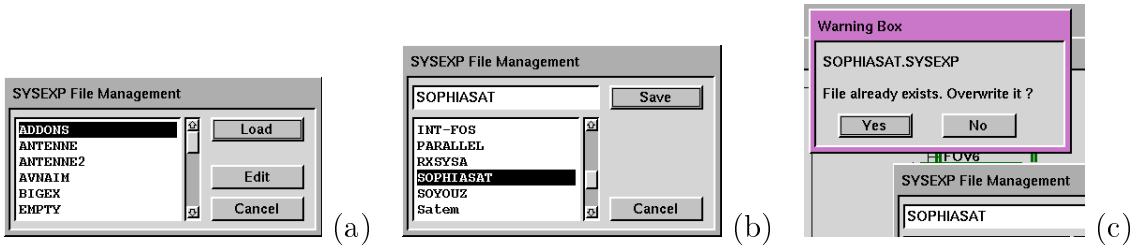


Figure A.1: Les dialogues du menu *Project.* (a) ouvrir un projet, (b) sauvegarder un projet, (c) "le projet existe déjà, voulez-vous l'écraser ?"

Save Sauvegarder le projet : `toto.SYSEXP`, `toto.lp` et `toto.geom`. Un nom est demandé ; par défaut le nom courant est utilisé (voir la figure A.1b). *Il est conseillé de faire des sauvegardes régulièrement.*

Close Fermer le projet. Le programme demande s'il doit être sauvegardé.

Parameters Les paramètres du projet : le domaine et la distance minimale permise entre les équipements (la marge mécanique) peuvent être modifiés (voir le dialogue dans la figure A.3). Les paramètres sont mémorisés dans le fichier du projet.

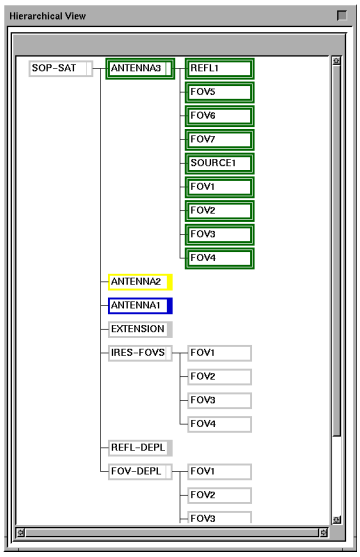


Figure A.2: Le browser des équipements.

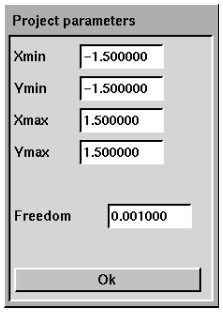


Figure A.3: Le dialogue des paramètres du projet.

Quit Quitter GEOTOOLS. Si un projet est ouvert, le programme demande s'il doit être sauvegardé.

Layout

Les fonctions de placement interactif et de vérification.

Constrained move Entrer en mode de déplacement des objets sélectionnés par rapport à tous les autres. Ceci est le déplacement contraint : la couleur de l'ensemble d'objets sélectionnés change en indiquant si la position actuelle est permise ou non. Les régions représentant l'ensemble des placements permis sont visualisées (voir la figure A.4).

Uniquement le déplacement dans le plan XY est possible : pour déplacer dans un autre plan, il faudra tourner le modèle entier avant avec la fonction *Freely move* (page 102). Les données géométriques calculées sont stockées afin de pouvoir être réutilisées. Rien ne peut être (dé)sélectionné dans le browser tant que la fonction *Constrained move* est active.

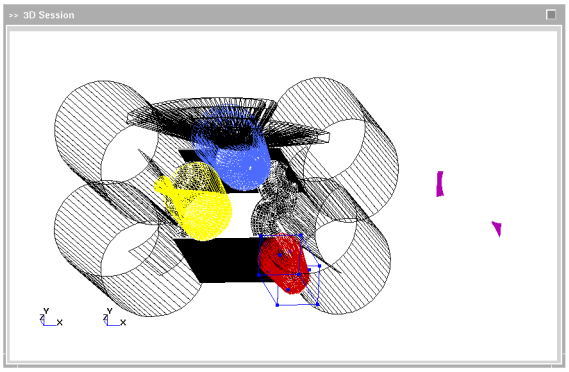
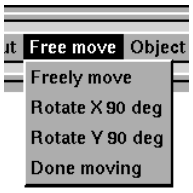


Figure A.4: La visualisation de l'espace admissible des équipements à placer.

Done moving Terminer la fonction *Constrained move*.

Verify layout Vérifier si l'aménagement courant est bon, i.e. s'il n'y a pas de collision entre équipements/champs de vision. Rapporter toutes les collisions détectées. La vérification calcule toutes les données géométriques ; la fonction *Constrained move* effectuée après est alors plus rapide.



Free move

Les fonctions de déplacement/rotation d'objets sans vérification de validité (transformation libre).

Ces fonctions permettent d'effectuer les manipulations suivantes :

- Tourner une antenne par rapport à son axe (dans le repère du père qui doit donc être correctement défini !).
- Eloigner/rapprocher un équipement de la plate-forme.
- Sélectionner un plan XY de placement en tournant le modèle entier en des pas de 90^o.

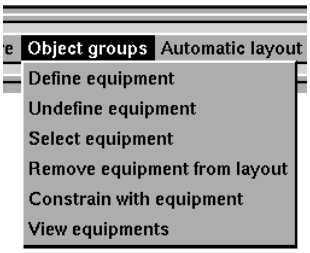
Remarque. D'éventuelles données géométriques calculées concernant un objet sont invalidées et donc supprimées si cet objet est modifié avec les fonctions de ce menu.

Freely move Entrer en mode de déplacement libre des objets sélectionnés dans le browser par rapport aux autres. Le mode de déplacement est contrôlé avec le dialogue *Object Movement Constraints* (voir la section A.2.2 sur la page 107). Rien ne peut être (dé)sélectionné dans le browser tant que la fonction *Freely move* est active.

Rotate X 90^o Rotation de 90^o autour l'axe X de l'ensemble d'objets en cours de déplacement libre. Le bouton est active uniquement après affectation de la fonction *Freely move*.

Rotate Y 90^o, **Rotate** Z 90^o Idem, autour les axes Y et Z.

Done moving Terminer la fonction *Freely move*.



Object groups

Manipulation d'équipements composés (= groupes d'objets).

Define equipment Définir un équipement composé (=groupe d'objets). La fonction prend l'ensemble des objets sélectionnés dans le browser et demande un nom pour l'équipement (voir le dialogue dans la figure A.5a). Un objet ne peut pas être dans plusieurs équipements ; l'utilisateur est averti dans le *Output box* s'il essaie d'inclure un objet dans un nouvel équipement.

Undefine equipment Eclater un équipement défini. Un dialogue permet la sélection d'un ou plusieurs équipements à éclater (voir la figure A.5b).

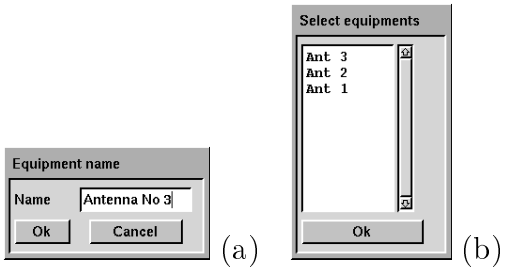


Figure A.5: Les dialogues du menu *Object groups*. (a) création d'un équipement, un nom est demandé ; (b) sélection d'un ou plusieurs équipements définis.

Select equipment Sélectionner les composantes d'un équipement défini (voir le dialogue dans la figure A.5b) dans le browser et désélectionner tout autre objet.

Remove equipment from layout Enlever les composantes d'un équipement défini (A.5b) de l'aménagement. Les composantes seront ni visualisées ni prises en compte dans le calcul des espaces admissibles.

Constrain with equipment Cette fonction est active uniquement lorsqu'on est en train de placer un équipement E avec la fonction *Constrained move*. Dans ce cas, le dialogue A.5b permet de sélectionner un ou plusieurs équipements ; l'espace admissible de l'équipement E est ensuite contraint avec les équipements sélectionnés.

View equipments Montrer la liste des équipements définis et de quels objets ils sont composés. Le rapportage est produit dans le *Output box* (voir la figure A.6).

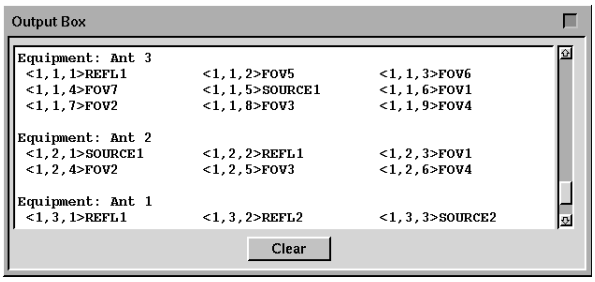
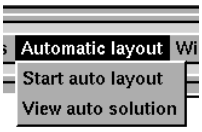


Figure A.6: La liste des équipements composés produite par la fonction *View equipments*.

Lorsqu'aucun équipement défini n'est choisi (voir le dialogue dans la figure A.5b) lors de n'importe quelle fonction de ce menu, l'opération actuelle est annulée. Si exactement *un* équipement devait être choisi mais plusieurs le sont, celui qui est le plus bas dans la liste est pris.



Automatic layout

Placement automatique d'un ensemble d'équipements composés.

L'objectif du placement automatique d'ensembles d'équipements est de générer un petit ensemble de solutions (approximatives) pour étude et éventuel affinement.

Start layout Démarrer l'aménagement automatique. Le programme génère un nombre de placements de tous les équipements définis dans le menu *Object groups*

(par rapport aux objets pas dans un équipement défini). Le nombre maximal de solutions à générer ainsi que la distance minimale entre deux solutions telle qu'elles soient considérées uniques peuvent être spécifiés dans le dialogue qui précède le placement automatique (voir la figure A.7a).

View solution Visualiser une des solutions trouvées ou rétablir l'ancienne configuration. La solution est choisie dans un dialogue (voir la figure A.7b).

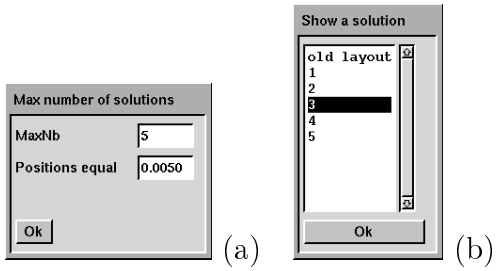
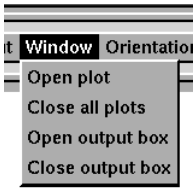


Figure A.7: Les dialogues du menu *Automatic layout*. (a) Démarrer le placement automatique, (b) Choisir une solution générée.



Window

Gestion des fenêtres de visualisation et des messages.

Open plot Ouvrir une fenêtre de visualisation du modèle (voir la figure A.8). Plusieurs fenêtres de visualisation peuvent être ouvertes au même temps. L'orientation de la caméra qui regarde le modèle peut être changée avec les contrôles standards de T3D. On peut obtenir une orientation orthogonale avec les fonctions du menu *Orientation*. La fonction est inactive lors des fonctions *Constrained move* et *Freely move* ; il faut appeler *Done moving* afin de pouvoir ajouter une fenêtre de visualisation.

Close all plots Fermer toutes les fenêtres de visualisation.

Open output box Ouvrir la boite des messages (voir figure A.9).

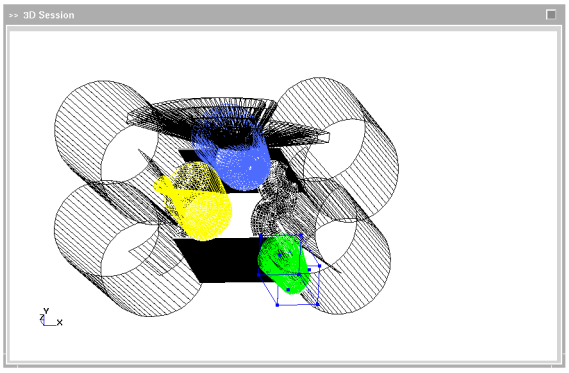


Figure A.8: La fenêtre GEOTOOLS/T3D de visualisation 3D du modèle.

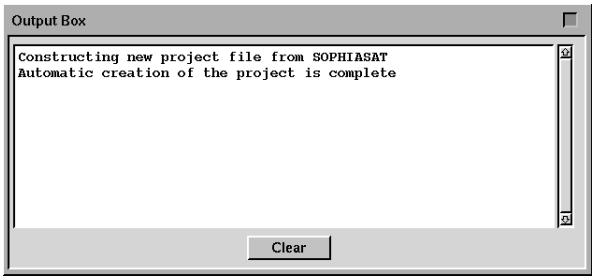
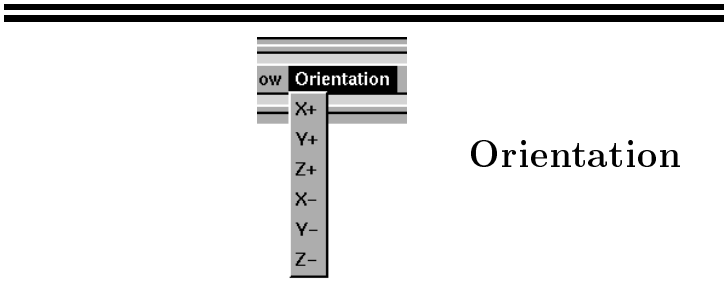


Figure A.9: La fenêtre GEOTOOLS/T3D des messages.

Close output box Fermer la boite des messages.



Changer la vue de la fenêtre de visualisation courante.

Remarque Ceci change la vue, mais pas le repère du modèle. Il faut utiliser la fonction *Freely move* sur le modèle complet pour aligner le plan souhaité avec le plan XY de placement contraint.

A.2.2 Manipulation d'objets

Etat d'un objet En double-cliquant sur un objet dans le browser, un dialogue apparaît qui permet la modification de l'état de l'objet du modèle (voir figure A.10). Les propriétés suivantes peuvent être modifiées :

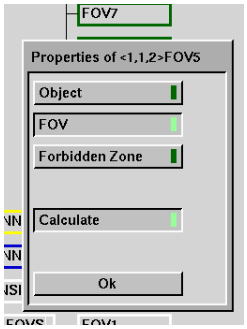


Figure A.10: Modifications des propriétés d'un objet.

- l'objet, est-il une composante matériel, un champ de vision ou une zone interdite ?

- l'objet, est-il pris en compte dans le calcul ?

Les objets matériels sont visualisés comme des solides dans les fenêtres de visualisation ; les champs de vision et les zones interdites en fil de fer. Les objets pas pris en compte ne sont pas affichés.

Contraintes de déplacement En double-cliquant sur une fenêtre de visualisation, le *3D Tool Box* apparaît (voir la figure A.11a). En cliquant sur le bouton *Selection*, on appelle le dialogue *Selection Box* (A.11b). Cliquer sur le bouton *Move* dans le *Selection Box* pour appeler le dialogue *Object Movement Constraints* (A.11c). Ce dernier permet de gérer les contraintes de déplacement de l'objet/le modèle entier :

- Activer la fonction *Move* permet le déplacement de l'objet. Si inactif, un déplacement s'applique à la caméra qui regarde le modèle.

  Remarque. Le déplacement d'objets est uniquement possible lors des fonctions *Constrained move* et *Freely move*. Lors d'un *Constrained move*, uniquement les déplacements dans le plan XY feront déplacer les objets.

- Activer les boutons de déplacement X, Y et Z permet le déplacement dans cette direction. Par exemple, lors de la fonction *Constrained move*, X et Y sont actifs (mais peuvent être désactivés par l'utilisateur). Il y a aussi trois boutons pour activer les trois axes de rotation (relevant lors d'un *Free move*), mais qu'*un* axe de rotation ne peut être actif à la fois.

Les autres fonctions dans ces dialogues ne sont pas relevantes pour GEOTOOLS.

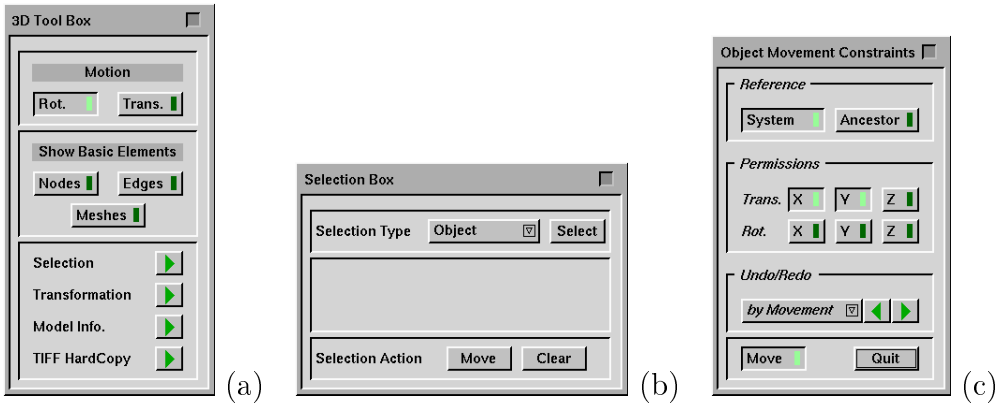


Figure A.11: Les dialogues de la fenêtre de visualisation. (a) La boite à outils 3D, (b) La boite de sélection, (c) Les contraintes de déplacement de l'objet.

Annexe B

Evaluation par le bureau d'études

GEOTOOLS a été évalué par le bureau d'études de Matra Marconi Space. L'évaluation porte sur deux aspects du logiciel : son interface homme-machine (désormais appelée IHM) et le module de calcul géométrique.

Au début, GEOTOOLS utilisait l'outil Geomview pour la visualisation et la bibliothèque Tcl/Tk pour les menus. Vu que cette version de GEOTOOLS manquait l'interface nécessaire pour communiquer avec les autres outils du bureau d'études, le logiciel a été intégré avec l'IHM T3D, développée par Matra Marconi Space.

Vu que T3D s'est montrée insuffisante pour des opérations d'aménagement et que, de toutes façons, la thèse s'intéresse principalement au côté géométrique, ces deux aspects sont présentés séparément dans cette annexe.

B.1 Problèmes de l'interface homme-machine

L'IHM T3D assure la visualisation 3D du modèle, la visualisation de l'espace admissible calculé, l'interaction avec l'utilisateur et la lecture, manipulation et écriture du modèle. T3D se sert de modèles qui sont stockés en format SYSEXP, le format interchangeable de SYSTEMA, un outil d'analyses physiques de satellite développé par Matra Marconi Space.

L'interface présente deux problèmes :

- Les modèles de satellite sont habituellement conçus avec l'outil de CAO EUCLID. Un modèle EUCLID doit donc être converti en SYSEXP afin de pouvoir être lu. Cette conversion est faite quasi-manuellement et prend beaucoup de temps.

- La manipulation graphique est lente, ce qui la rend inutilisable pour des modèles de taille réaliste : en pratique, l'opérateur souhaite pouvoir observer le modèle sous plusieurs angles pour déterminer si un placement satisfait aussi les contraintes qui n'ont pas été modélisées géométriquement. De plus, la saisie d'une configuration sur le clavier n'est pour l'instant pas disponible ni une fonction de mesure de distance entre équipements.

Ces difficultés rencontrées lors des tests du logiciel font que l'évaluation même du module de calcul géométrique ne mène qu'à une estimation du potentiel d'un logiciel qui l'intègre.

B.2 Evaluation du module de calcul géométrique

L'évaluation a été faite sur l'aménagement de deux antennes pour l'appel d'offre en cours du satellite de télécommunication ASTRA 2B. Le modèle concret sur lequel s'est déroulé l'évaluation concerne la paroi supérieure du satellite comprenant la structure et tous les équipements "auxiliaires", tels que les instruments d'orientation du satellite, les tuyères de propulsion et les fixations des générateurs solaires. Cet aménagement contraint fortement le placement des deux antennes, dont une est mobile.

Les priorités de placement sont :

- les parties portantes des antennes doivent se situer le plus près possible du milieu de la plate-forme. Cela est surtout important pour l'antenne mobile, qui est motorisée et donc lourde : on souhaite la centrer pour des raisons de centrage de masse,
- la distance minimale des champs de vue par rapport aux pièces d'équipement doit être maximisée.

La procédure suivie était de :

1. Charger le modèle SYSEXP.
2. Placer les deux antennes. Un placement n'a pas pu être trouvé pour l'antenne mobile.
3. Chercher un placement en hauteur pour l'antenne mobile. Ceci consiste a déterminer, pour une suite de hauteurs pour l'antenne mobile, si les deux antennes peuvent être placées. Nous rappelons que l'espace admissible doit être recalculé pour chaque hauteur. Le placement simultané de deux objets, qui rendrait le placement des deux antennes immédiat pour chaque hauteur, n'a malheureusement pas été testé.

Le calcul et la visualisation de l'espace admissible pour un équipement qu'on essaie de placer rend la localisation d'un bon placement beaucoup plus facile. Au total, le calcul des espaces admissibles aux différents hauteurs a pris environ 18 minutes. Sans compter la conversion du modèle en SYSEXP, l'aménagement fait avec GEOTOOLS a pris environ 5 heures et 30 minutes à cause des problèmes d'IHM.

Une partie du même aménagement a aussi été faite avec la version de GEOTOOLS sous Geomview : nous estimons à partir de cela que l'aménagement entier aurait pris environ 3 heures et 15 minutes. Cette version de GEOTOOLS se sert aussi de fichiers SYSEXP.

La conversion du modèle en SYSEXP a pris 3 heures et 30 minutes. Nous estimons que, avec une bonne IHM et la lecture directe de modèles EUCLID, l'opération totale aurait pu être faite en 1 heure et 40 minutes, où le même placement fait manuellement prend facilement 8 à 12 heures.

B.3 Suggestions

MEGAVISION est un produit développé par Matra Datavision qui permet la vérification d'un modèle en déterminant s'il y a des collisions entre les objets du modèle. MEGAVISION *vérifie* uniquement un aménagement tandis que notre approche *propose des solutions* pour le placement des équipements. MEGAVISION lit directement des modèles EUCLID.

Pour que GEOTOOLS soit utilisable pour le bureau d'études, la meilleure solution serait d'integrer le module de calcul d'espace admissible dans MEGAVISION comme une fonction supplémentaire. Le module intégré profiterait de la lecture directe de modèles EUCLID, une IHM plus efficace et toutes les fonctions de calcul de proximité et autres.

Concernant la présentation de l'espace admissible, on aimerait avoir la possibilité de calculer et de sélectionner les centres des cercles inscrits dans l'espace admissible : de telles configurations garantissent que l'équipement est localement à distance maximale (en translation dans le plan) des autres équipements. L'ensemble de ces points est égal à l'ensemble des sommets intérieurs du diagramme de Voronoï de la région polygonale représentant l'espace admissible. Le calcul du diagramme de Voronoï ne présente aucune difficulté.

Une fonctionnalité intéressante serait aussi la possibilité de tester des séquences de placements :

- Trouver automatiquement la hauteur minimale où le placement d'un équipe-

ment dans un aménagement est possible. Ceci peut être fait en augmentant la hauteur par petits pas ; à chaque hauteur, l'espace admissible est calculé jusqu'à ce qu'il ne soit plus vide, ce qui veut dire qu'il existe un placement.

- En testant le placement d'une séquence de variantes d'un équipement, trouver la première dans la séquence qui est plaçable dans un aménagement. La séquence est par exemple un équipement en taille décroissante et ceci permet alors de déterminer quelle est la plus grande taille de l'équipement telle qu'il soit plaçable.

Bibliographie

[AB87] F. Avnaim and J.-D. Boissonnat. Simultaneous containment of several polygons. In *Proc. 3rd Annu. ACM Sympos. Comput. Geom.*, pages 242–250, 1987.

[AB89] F. Avnaim and J.-D. Boissonnat. Polygon placement under translation and rotation. *RAIRO Inform. Theor.*, 23:5–28, 1989.

[ABD+97] F. Avnaim, J.-D. Boissonnat, O. Devillers, F. Preparata, and M. Yvinec. Evaluating signs of determinants using single-precision arithmetic. *Algorithmica*, 17:111–132, 1997.

[Avn89] F. Avnaim. *Placement et déplacement de formes rigides ou articulées.* Thèse de doctorat en sciences, Université de Franche-Comté, Besançon, France, 1989.

[BDH93] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The Quickhull algorithm for convex hull. Technical Report GCG53, Geometry Center, Univ. of Minnesota, July 1993.

[BdLT97] J-D. Boissonnat, E. de Lange, and M. Teillaud. Minkowski operations for satellite antenna layout. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, 1997.

[BdLT98] J-D. Boissonnat, E. de Lange, and M. Teillaud. Slicing Minkowski sums for satellite antenna layout. *Computer-Aided Design*, 1998. à paraître.

[BDS+92] J.-D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec. Applications of random sampling to on-line algorithms in computational geometry. *Discrete Comput. Geom.*, 8:51–71, 1992.

[BEPP97] H. Brönnimann, I. Emiris, V. Pan, and S. Pion. Computing exact geometric predicates using modular arithmetic with single precision. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 174–182, 1997.

[BMS94] C. Burnikel, K. Mehlhorn, and S. Schirra. On degeneracy in geometric computations. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 16–23, 1994.

[BO79] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28:643–647, 1979.

[BP97] Jean-Daniel Boissonnat and Franco P. Preparata. Robust plane sweep for intersecting segments. Technical Report 3270, INRIA, 1997.

[BY95] J.-D. Boissonnat and M. Yvinec. *Géométrie algorithmique*. Ediscience international, Paris, 1995.

[CE92] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39:1–54, 1992.

[Cha83] B. Chazelle. The polygon containment problem. In F. P. Preparata, editor, *Computational Geometry*, volume 1 of *Adv. Comput. Res.*, pages 1–33. JAI Press, London, England, 1983.

[Cha95] Philippe Charman. *Gestion des contraintes géométriques pour l'aide à l'aménagement spatial*. Thèse de doctorat en sciences, Ecole Nationale des Ponts et Chaussées, 1995.

[Cla92] K. L. Clarkson. Safe and effective determinant evaluation. In *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 387–395, 1992.

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[Dan95] K. Daniels. *Containment algorithms for nonconvex polygons with applications to layout*. Ph.D. thesis, Harvard University, Cambridge, MA, 1995.

[dB92] M. de Berg. *Efficient algorithms for ray shooting and hidden surface removal*. Ph.D. dissertation, Dept. Comput. Sci., Utrecht Univ., Utrecht, Netherlands, 1992.

[DD92] Kathryn A. Dowsland and William B. Dowsland. Packing problems. *European J. Oper. Res.*, 56:2–14, 1992.

[Dev93] O. Devillers. Simultaneous containment of several polygons: analysis of the contact configurations. *Internat. J. Comput. Geom. Appl.*, 3(4):429–442, 1993.

[DM95] Karen Daniels and Victor Milenkovic. Multiple translational containment: Approximate and exact algorithms. In *Proc. 6th ACM-SIAM Sympos. Discrete Algorithms*, pages 205–214, 1995.

[DML94] K. Daniels, V. Milenkovic, and Z. Li. Multiple containment methods. Technical Report 12-94, Center for Research in Computing Technology, Division of Applied Sciences, Harvard University, Cambridge, MA, 1994.

[DP97] O. Devillers and F. Preparata. A probabilistic analysis of the power of arithmetic filters. *Discrete amd Computational Geometry*, 1997. to appear.

[Dyc90] H. Dyckhoff. A typology of cutting and packing problems. *European J. Oper. Res.*, 44(2):145–160, 1990.

[For85] S. J. Fortune. A fast algorithm for polygon containment by translation. In *Proc. 12th Internat. Colloq. Automata Lang. Program.*, volume 194 of *Lecture Notes Comput. Sci.*, pages 189–198. Springer-Verlag, 1985.

[FV93] S. Fortune and C. J. Van Wyk. Efficient exact arithmetic for computational geometry. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 163–172, 1993.

[GGHT97] Michael T. Goodrich, Leonidas J. Guibas, John Hershberger, and Paul J. Tanenbaum. Snap rounding line segments efficiently in two and three dimensions. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages – , 1997.

[GM95] Leonidas Guibas and David Marimont. Rounding arrangements dynamically. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 190–199, 1995.

[GO95] A. Gajentaan and M. H. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Comput. Geom. Theory Appl.*, 5:165–185, 1995.

[GRS83] L. J. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 100–111, 1983.

[GS87] L. J. Guibas and R. Seidel. Computing convolutions by reciprocal search. *Discrete Comput. Geom.*, 2:175–193, 1987.

[GSS89] L. J. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry: building robust algorithms from imprecise computations. In *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, pages 208–217, 1989.

[GY86] D. H. Greene and F. F. Yao. Finite-resolution computational geometry. In *Proc. 27th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 143–152, 1986.

[KLPS86] K. Kedem, R. Livne, J. Pach, and M. Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.*, 1:59–71, 1986.

[KOS92] M. J. Katz, M. H. Overmars, and M. Sharir. Efficient hidden surface removal for objects with small union size. *Comput. Geom. Theory Appl.*, 2:223–234, 1992.

[LP83] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. Comput.*, C-32:108–120, 1983.

[LPT96] Giuseppe Liotta, Franco P. Preparata, and Roberto Tamassia. Robust proximity queries in implicit Voronoi diagrams. Technical Report CS-96-16, Center for Geometric Computing, Comput. Sci. Dept., Brown Univ., Providence, RI, 1996.

[Mil90] V. Milenkovic. Rounding face lattices in d dimensions. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 40–45, 1990.

[Mil95] Victor J. Milenkovic. Practical methods for set operations on polygons using exact arithmetic. In *Proc. 7th Canad. Conf. Comput. Geom.*, pages 55–60, 1995.

[MN94] K. Mehlhorn and S. Näher. The implementation of geometric algorithms. In *Proc. 13th World Computer Congress IFIP94*, volume 1, pages 223–231, 1994.

[MPS+94] J. Matoušek, J. Pach, M. Sharir, S. Sifrony, and E. Welzl. Fat triangles determine linearly many holes. *SIAM J. Comput.*, 23:154–169, 1994.

[OWW85a] T. Ottmann, P. Widmayer, and D. Wood. A fast algorithm for Boolean mask operations. *Comput. Vision Graph. Image Process.*, 30:249–268, 1985.

[OWW85b] T. Ottmann, P. Widmayer, and D. Wood. A worst-case efficient algorithm for hidden line elimination. *Internat. J. Comput. Math.*, 18:93–119, 1985.

[PT93] Simone Pimont and Michel Tollenaere. Modèles et techniques pour l'aménagement spatial. *Sciences et techniques de la conception*, 2(1):97–123, 1993.

[She97] Jonathan Richard Shewchuk. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete Comput. Geom.*, 18:305–363, 1997.

[Tei93] M. Teillaud. *Towards dynamic randomized algorithms in computational geometry*, volume 758 of *Lecture Notes Comput. Sci.* Springer-Verlag, 1993.

[vK93] Marc van Kreveld. On fat partitioning, fat covering, and the union size of polygons. In *Proc. 3rd Workshop Algorithms Data Struct.*, volume 709 of *Lecture Notes Comput. Sci.*, pages 452–463. Springer-Verlag, 1993.

Aide géométrique à l'aménagement de satellites

Nous nous intéressons dans cette thèse à des problèmes d'aménagement de satellites. Le but est de développer des algorithmes efficaces et robustes menant à des outils simples pour aider l'ingénieur du bureau d'études dans ses tâches répétitives d'aménagement.

Nous proposons un algorithme optimal qui calcule une section plane de la somme de Minkowski de deux polyèdres convexes et un algorithme efficace qui calcule l'union d'un ensemble de polygones par division et fusion. Nous avons soigneusement analysé la précision numérique nécessitée pour le fonctionnement correct de ces deux algorithmes et le traitement des dégénérescences géométriques qui peuvent apparaître.

Nous avons conçu le logiciel GEOTOOLS pour le placement d'une suite d'équipements et en particulier des antennes qui ont un champ de vision. GEOTOOLS permet le placement interactif d'un objet dans un aménagement partiel en visualisant les contraintes imposées par cet aménagement (l'espace admissible). La deuxième partie de cette thèse consiste en une expérimentation de GEOTOOLS sur des modèles réalistes de satellites en plaçant des séquences d'antennes interactivement et automatiquement.

Mots-clés : Géométrie algorithmique, placement de formes, espace libre, aménagement de satellite, robustesse, précision numérique

Geometric tools for satellite layout

This thesis addresses satellite layout problems. The goal is to develop efficient algorithms that lead to simple tools that help the design office engineer in his frequent layout problems.

We propose an optimal algorithm that calculates a planar section of the Minkowski sum of two convex polyhedra and an efficient algorithm that calculates the union of a set of polygons. We closely analyzed the numerical precision that is necessary for the two algorithms to work correctly and the handling of geometric degeneracies that can be encountered.

We developed the program GEOTOOLS for the placement of a sequence of equipment and in particular antennas that have a field of vision. GEOTOOLS allows interactive placement of a piece of equipment in a partially completed layout while visualizing the constraints imposed by this layout (the admissible space). The second part of this thesis discusses our experiences using GEOTOOLS on realistic models of satellites when placing sequences of antennas interactively and automatically.

Keywords: Computational geometry, shape placement, free space, satellite layout, robustness, numerical precision