Institut
National
Polytechnique
de Lorraine

Ecole Nationale Supérieure de Géologie de Nancy
(E.N.S.G)
Laboratoire Informatique et Analyse des Donnécs
(L.I.A.D)

THESE

présentée devant l'Institut National Polytechnique de Lorraine
en vue de l'obtention du titre de

Docteur de l'I.N.P.L

(Informatique)

par

Yun-Gao HUANG

Modélisation et Manipulation des Surfaces Triangulées

Soutenue publiquement le 19 Octobre 1990, devant le Jury composée de :

Jean-Laurent MALLET	Président
René SCHOTT	Rapporteur
Bernard LACOLLE	Rapporteur
Jean-Paul HATON	Examinateur
Pierre LESCANNE	Examinateur
[illegible] MOHR	Examinateur

Institut
National
Polytechnique
de Lorraine

Ecole Nationale Supérieure de Géologie de Nancy
(E.N.S.G)
Laboratoire Informatique et Analyse des Données
(L.I.A.D)

THESE

présentée devant l'Institut National Polytechnique de Lorraine
en vue de l'obtention du titre de

Docteur de l'I.N.P.L

(Informatique)

par

Yun-Gao HUANG

Modélisation et Manipulation des Surfaces Triangulées

Soutenue publiquement le 19 Octobre 1990, devant le Jury composée de :

Jean-Laurent MALLET	**Président**
René SCHOTT	**Rapporteur**
Bernard LACOLLE	**Rapporteur**
Jean-Paul HATON	**Examinateur**
Pierre LESCANNE	**Examinateur**
Roger MOHR	**Examinateur**

REMERCIEMENTS

En tout premier lieu, ma gratitude va au Professeur Jean-Laurent MALLET, Directeur du Département informatique de l'Ecole de Géologie, qui m'a donné la possibilité de travailler au Laboratoire, dans ce domaine passionnant qu'est la Modélisation Géométrique des surfaces naturelles. Sans son soutien et ses encouragements, ce travail n'aurait pas abouti. Il est aussi le Directeur de cette thèse. Depuis plusieurs années, il m'a orienté et suivi de très près, dans ce travail, avec sa compétence, sa patience, sa gentillesse et sa rigueur intellectuelle.

Je remercie vivement Pierre JACQUEMIN, Ingénieur de recherche au CNRS, qui m'a aidé chaleureusement et avec patience tant pour le présent mémoire et pour les probèmes rencontrés en informatique.

Ma reconnaissance s'adresse tout particulièrement à Monsieur René SCHOTT, Professeur à l'Université de Nancy I. Je sais que je lui dois plusieurs précieuses suggestions lors du parachèvement et de la rédaction de ce mémoire.

J'aimerais remercier Monsieur Bernard LACOLLE, Directeur de recherche à l'IMAG, qui a gentiment accepté d'être le Rapporteur extérieur.

Mes remerciements vont également à Monsieur Pierre LESCANNE, Directeur de recherche à l'INRIA et à Monsieur Jean-Paul HATON, Professeur au CRIN, qui m'ont fait l'honneur de juger ce travail et à qui je dois quelques remarques utiles.

J'adresse également mes remerciements à tous les membres du LIAD (Laboratoire Informatique et Analyse des Données de l'Ecole Nationale Supérieure de Géologie de Nancy) et tous ceux qui m'ont soutenu dans ce travail. Leur amitié et leur aide chaleureuse m'ont permis d'avoir vécu dans une ambiance de travail aussi intéressante qu'agréable que je n'oublierai jamais.

Résumé

En *CAO* classique, on suppose que les surfaces admettent une représentation paramétrique $B(u,v)$ et sont interpolées par des interpolations du type *Bezier*, *B_spline*, *β_spline* ou *NURBS*. Ces méthodes ne permettent pas de traiter valablement des surfaces géologiques complexes. C'est la raison pour laquelle le projet *GOCAD* a été développé au Laboratoire Informatique et Analyse des données de l'Ecole Nationale Supérieure de Géologie de Nancy. Dans ce projet, les surfaces sont modélisées par des facettes triangulaires (T-surfaces); car cette sorte de caractérisation est un bon compromis entre la précision d'approximation et la simplicité de manipulation de la surface. Les travaux décrits font partie du projet *GOCAD* et ont pour but de modéliser et manipuler les T-surfaces.

Nous développons d'abord une méthode qui convertit une surface isovaleur définie par une grille régulière 3D en T-surface. Les données d'entrée de cette méthode sont une liste de valeurs correspondant à la valeur en chaque noeud de la grille. Dans le cas où il existe des discontinuités, il est possible de modéliser la T-surface de façon interactive.

La mise en oeuvre des opérations booléennes sur des solides définis par leur bord a toujours été un problème délicat. Dans cette thèse, nous proposons une méthode générale permettant de résoudre le problème posé dans le cas où les objets sont des T-solides. L'algorithme exploite pleinement la base de données du projet *GOCAD*, il tient compte de tous les cas particuliers qui peuvent se présenter en pratique. Ensuite, ces résultats sont étendus aux solides quelconques.

En s'appuyant sur les études précédentes, nous développons une méthode de découpage de T-surfaces complexes. En utilisant cette méthode, nous effectuons des opérations "chirurgicales" correspondant aux applications géologiques sur les T-surfaces.

Les résultats obtenus à l'issue de ce travail constitue une contribution originale dans le domaine de l'informatique appliquée aux sciences de la terre.

Abstract

In geometric modeling by classical CAD methods, the surface are supposed to be a parametric representation $B(u,v)$ et are interpolated by numerical methods such as *Bezier*, *B_Spline* β*_Spline* or $NURBS$. However, these methods cannot treat correctly complex geological surfaces. This is the main reason of the $GOCAD$ project ($ENSG$). In this project, the complex surface are modelled by triangular facets. Since this sort of caracterisation is a good compromise between the precision of approximating and the simplicity of manipulating of the surface. This thesis is a part of the $GOCAD$ project, its aim is to model and manipulate the T-surfaces.

We develop first one method which converts an isovalue surface defined by a regular 3D grid into T-surfaces. The input data of this method is a list of values corresponding to the values at each node of the grid. In the case where there is some discontinuity, we can model these T-surfaces interactively.

The implementation of the boolean set operations of solids defined by their border has always been a difficult problem. In this thesis, we propose one general method which allows to resolve the problem in the case where the objects are T-solids. The algorithm makes use of the data-base of $GOCAD$ project; it can treat all special cases and these results are extended to any manifolds.

Based on the previous studies, we develop a method to cut the complex T-surfaces. With the help of this method, we accomplish some "surgical" operations on T-surfaces corresponding to geological applications.

The resultats obtained in this work constitute an original contribution to computer applications in earth sciences.

Tables des matières

Partie I

Outils de base

Introduction

0.1 Modélisation géométrique de surfaces par l'approche classique

En *CAO* classique, les surfaces modélisées sont supposées admettre une représentation paramétrique $B(u,v)$ et sont interpolées par des interpolations du type ***Bezier***, ***B_spline***, ***β_spline*** ou ***NURBS*** (voir [32]). Comme l'indique la Figure 0.1, une surface de ***Bezier*** $B(u,v)$ est définie par une série de points de contrôle $\{p_{ij}\}$ formant un réseau régulier rectangulaire $\{p_{11}, p_{12}, \ldots, p_{mn}\}$:

$$B(u,v) = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} \cdot B_{i,m}(u) \cdot B_{j,n}(v) \qquad u,\ v\ \in [0,1]$$

avec

$$\left| \begin{array}{l} B_{i,m}(u) = \frac{m!}{i!(m-1)!} u^i (1-u)^{m-1} \\ \\ B_{j,n}(v) = \frac{n!}{j!(n-1)!} v^i (1-v)^{n-1} \end{array} \right.$$

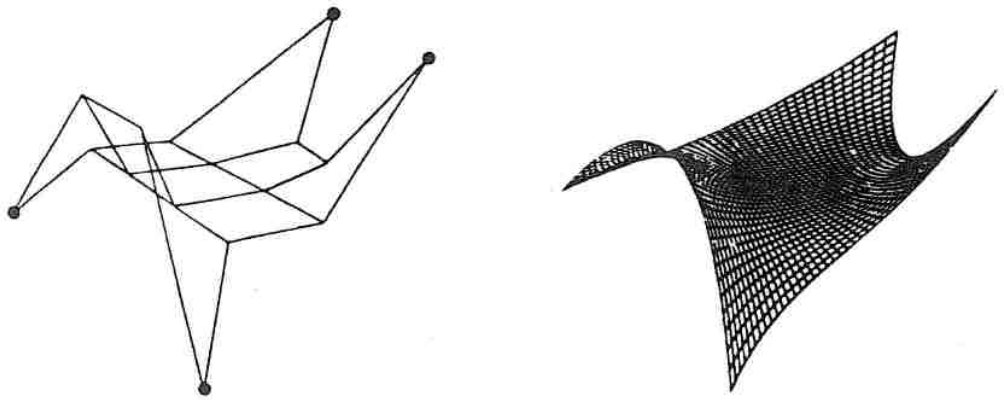
Dans cette formule, les points de contrôles $\{p_{ij}\ (0 \leq i \leq m$ et $0 \leq j \leq n\}$ sont approximés et non pas interpolés par $B(u,v)$ (sauf les points situés aux 4 coins).

Les bases mathématiques des ***B_splines***, des ***β_splines*** ou des ***NURBS*** sont proches de celle de la méthode de ***Bezier*** si bien que les interpolateurs correspondant ont des propriétés similaires. Les surfaces modélisées par ces méthodes sont généralement régulières et lisses et possèdent des propriétés très séduisantes pour les applications mécaniques (par exemple, ces surfaces sont souvent plusieurs fois dérivables et ceci est important en *CFAO*). Grâce à leur simplicité de mise en oeuvre, ces méthodes sont largement utilisées dans tous les domaines concernant la modélisation géométrique des surfaces manufacturées.

Inconvénients de ces méthodes

Les difficultés concernant l'application de ces méthodes classiques à la modélisation des surfaces géologiques ont trois origines :

- En général, la surface à modéliser est connue aux points échantillonnés $\{q_1, q_2, \ldots, q_m\}$ qui sont répartis irrégulièrement sur la couche. Ce fait implique que les points de contrôle doivent être considérés comme une fonction de ces points connus $\{p(i,j) = D(q_1, \ldots, q_m)\}$.



(a) (b)

Figure 0.1 *(a) Les points de contrôle $\{p_{ij}\}$; (b) la surface de Bezier associée.*

Dans ce cas, évidemment, la solution du problème n'est en général pas unique et une infinité d'ensembles de points de contrôle peuvent être construits à partir de ces m points informés générant des surfaces de *Bezier*.

- Ces méthodes ne permettent pas de traiter valablement des objets très complexes. Une surface représentant un objet géologique est souvent trouée pour rendre compte d'accidents naturels (faille, intersection par un dôme de sel, etc..), il est dans ce cas extrêmement difficile de définir les points de contrôles. De plus, ces méthodes ne fonctionnent que sur une base de points de contrôle complète homéomorphe à une grille rectangulaire, ce qui limite par conséquent la variété de surfaces que l'on peut modéliser avec ces méthodes.

- Ces méthodes s'adaptent difficilement aux modifications interactives de la topologie des surfaces modélisées. Par exemple, pour mettre à jour une faille (ajouter ou supprimer une faille, etc..), il est indispensable de reconstruire complètement le réseau rectangulaire correspondant aux "points de contrôle" p_{ij}, et ceci est incompatible avec une utilisation interactive.

- Ces méthodes ne permettent pas de prendre en compte des données hétérogènes, plus ou moins précises et irrégulièrement distribuées comme le sont, par exemple, les données géologiques.

0.2 Modélisation géométrique des surfaces complexes par des éléments triangulaires dans le cadre du projet *GOCAD*

Comme le montre la Figure 0.2, depuis une dizaine d'années, dans des domaines variés comme la biologie, la médecine, la géologie, on s'oriente souvent vers des techniques de triangulation

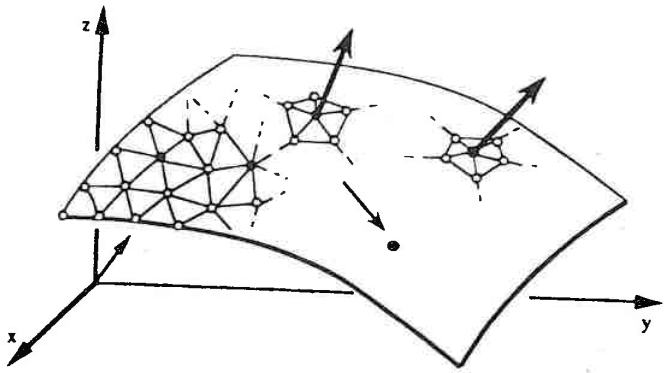


Figure 0.2 *Modélisation des surfaces par triangles.*

afin d'obtenir une représentation à la fois simple et relative précise d'une surface complexe à modéliser dont on ne connait, à priori, qu'un ensemble de points. Cette sorte de caractérisation est un bon compromis entre la précision d'approximation et la simplicité de modélisation et de manipulation de la surface.

De plus, les facettes triangulaires s'adaptent à la compléxité des surfaces et permettent ainsi une précision plus fine dans la définition de leurs formes. Cette aptitude à s'adapter à toutes les topologies rend l'utilisation des facettes triangulaires très séduisantes pour la représentation de surfaces complexes.

De nombreuses méthodes ont été développées dans la littérature pour modéliser les surfaces par les éléments triangulaires comme par exemple, la méthode de la triangulation de *Delaunay* (en 2D ou 3D) ou encore la méthode de reconstruction d'une surface connue par des coupes sériées (cf Chapitre 3.1). Grâce à la solidité de leur base mathématique et à la simplicité de leur mise en oeuvre, ces méthodes se sont montrées extrêmement efficaces pour modéliser les surfaces complexes correspondant aux objets "naturels" rencontrés par exemple en géologie et en médecine.

Ce type de modélisation des surfaces est celui qui a été retenu dans le projet *GOCAD* développé à l'Ecole de Géologie de Nancy (*INPL*) par l'equipe infographie du *CRIN* sous la direction de J.L Mallet. Le présent mémoire s'appuie largement sur les acquis du projet *GOCAD*, notamment en ce qui concerne :

- La base de données géométrique
- L'interpolation des surfaces triangulées basée sur l'algorithme *DSI* (voir [16])
- L'interface graphique

0.3 Objectif et Organisation de cette thèse

La géométrie algorithmique est un domaine en plein essort qui a pour objectif d'effectuer, de façon efficace, des calculs géométriques sur ordinateur. Cette thèse constitue en fait une étude de Géométrie Algorithmique dans la perspective d'applications géologiques et minières. Les travaux décrits font partie du project *GOCAD* et ont pour but de résoudre les deux problèmes suivants :

- tout d'abord modéliser géométriquement les objets géologiques par des surfaces triangulées appelées T-surfaces;
- ensuite, construire des outils pour la manipulation de ces T-surfaces tels que :
  - Calcul des opérations booléennes sur les solides définis par des T-surfaces fermées
  - Découpage des T-surfaces complexes par des couteaux ou ciseaux informatiques (cf Chapitre 2.2)
  - Calcul de volume entouré par des T-surfaces fermées quelconques
  - Détermination de la position d'un point par rapport à une T-surface fermée
  - etc ...

Première partie

La première partie de ce mémoire est dédiée à la construction d'un ensemble d'outils de base permettant la modélisation et la manipulation des T-surfaces. Elle est composée de 3 chapitres :

1. le premier chapitre rappelle brièvement quelques structures de données utilisées pour la modélisation des T-surfaces, des T-solides dans le cadre du projet *GOCAD*.
2. le deuxième chapitre est consacré à la construction et à l'utilisation du T-octree associé à une T-surfaces. Plusieurs problèmes au niveau de la construction du T-octree sont abordés; des résultats expérimentaux montrent l'efficacité de la construction du T-octree et la rentabilité de son utilisation.
3. Le troisième chapitre exposera certains problèmes géométriques de bas niveau qui peuvent être rencontrés dans les modélisations et les manipulations des T-surfaces. Les solutions de ces problèmes peuvent servir dans tous les domaines concernés par la Géométrie Algorithmique.

Deuxième partie

Dans la deuxième partie de ce mémoire, nous développons des algorithmes de manipulation des T-surfaces.

1. Le premier chapitre de cette partie est consacré aux T-solides (identifiées à leur peau constituée par des T-surfaces fermées). Dans un premier temps nous n'étudions que les opérations booléennes sur les T-solides. L'algorithme proposé exploite pleinement les structures de données du projet *GOCAD*. Il tient compte de toutes les situations délicates qui

peuvent être rencontrées en pratique. Ensuite ces résultats sont étendus aux solides quelconques définis par leurs bords. C'est une nouvelle approche qui simplifie considérablement la méthodologie de la mise en oeuvre des opérations booléennes des solides définis par leur bords. Cet algorithme est basé sur les T-solides et, en utilisant le T-octree associé sa complexité est en moyenne $O(\sqrt{N})$ (N est le nombre de triangles), alors que les autres méthodes proposées dans la littérature sont de $O(N)$ et nécessitent de nombreux calculs (voir [21]) préparatoires et $O(N^2)$ (voir [4]).

Dans un deuxième temps, le calcul du volume entouré par une T-surface fermée et la détermination de la position d'un point par rapport à une T-surface fermée sont abordés dans ce chapitre, ces deux problèmes sont très importants pour les applications géologiques.

2. Le deuxième chapitre de cette deuxième partie est intitulé "opérations sur les T-surfaces". On présente tout d'abord une méthode de découpage des T-surfaces et ensuite on effectue des opérations chirurgicales sur des T-surfaces complexes, ces opérations peuvent correspondre à :

   - un découpage d'une couche par une faille
   - une rupture de gisement à cause d'une faille
   - la mise en place d'un trou dans une couche percée par un dôme de sel
   - la suppression des morceaux isolés (ou incorrects)
   - etc ...

Troisième partie

La troisième partie expose trois méthodes concernant la modélisation géométrique des surfaces complexes par des éléments triangulaires. Dans la plupart des cas, les méthodes existantes pour la fabrication des T-surfaces (comme la méthode de la triangulation de Delaunay) peuvent donner des résultats satisfaisants, cependant, ces méthodes ne permettent pas de traiter de manière adéquate des objets complexes qui sont connus à partir des valeurs appartenant aux noeuds d'une grille régulière 2D ou 3D. Par exemple, en exploitation des mines, un gisement est souvent découpé en petits blocs qui constituent un réseau parallélépipédique formant une grille régulière 3D. La teneur $t = \varphi(x, y, z)$, à chaque noeud de cette grille, est estimée par une méthode géostatistique et le corps du gisement (noté G) est défini par l'ensemble des points ayant une teneur supérieure à une teneur de coupure (t_0) donnée $\Rightarrow G = \{\varphi(x, y, z) > t_0\}$. G est souvent composé de morceaux en forme de lentilles. La peau de G est très complexe, il est souvent difficile d'obtenir un modèle géométrique à la fois précis et simple à manipuler, et ceci constitue un inconvénient important pour la planification de l'exploitation des mines.

Dans cette partie nous avons développé une méthode qui convertit une surface isovaleur (cf page 130) définie par une grille régulière $3D$ en T-surface. Les données d'entrée de cette méthode sont une liste de valeurs correspondant à la valeur en chaque noeud de la grille. Dans le cas de la présence d'une discontinuité (faille), il est possible de modéliser ces T-surfaces interactivement avec les outils développés dans la deuxième partie de ce mémoire. Les exemples proposés montrent la précision et l'efficacité de cette méthode en modélisation géométrique des objets définis sur les grilles.

Chapitre 1

Structure de données

1.1 Définitions prélimilaires

1.1.1 T-surface simple

Définition

Nous appellerons T-surface, toute surface décomposée sous forme d'un ensemble de facettes triangulaires. Les côtés de chaque triangle sont les "arêtes" de la T-surface tandis que les sommets de chaque triangle sont appelés les "noeuds" ou encore "vertex" de la T-surface. Dans cette thèse, nous ne considérons que le sous-ensemble des T-surfaces "simples" satisfaisant les deux contraintes suivantes :

1. Chaque arête de la T-surface ne peut au plus appartenir qu'à deux facettes triangulaires différentes; cette condition exclu en particulier le cas indiqué dans la figure 1.1(a). Une arête qui n'appartient qu'à un seul triangle (respectivement à deux triangles) est dite "libre" (respectivement liée).

2. Si T_1 et T_2 sont deux facettes triangulaires quelconques appartenant à une même T-surface, alors les positions relatives autorisées entre T_1 et T_2 sont les suivantes :

   - $T_1 \cap T_2 = \phi$; elles n'ont pas de point commun.
   - T_1 *et* T_2 ne possèdent qu'un seul sommet commun.
   - T_1 *et* T_2 ont un côté commun .

La frontière d'une T-surface simple est constituée par l'ensemble des arêtes libres de celle-ci. Plus précisement les arêtes libres forment des cycles élémentaires simples disjoints. Un tel cycle est appelé ***bord*** de la T-surface, et la frontière de celle-ci est alors définie comme l'ensemble de ses bords. Enfin, une T-surface est dite ouverte, si l'ensemble de ses bords n'est pas vide.

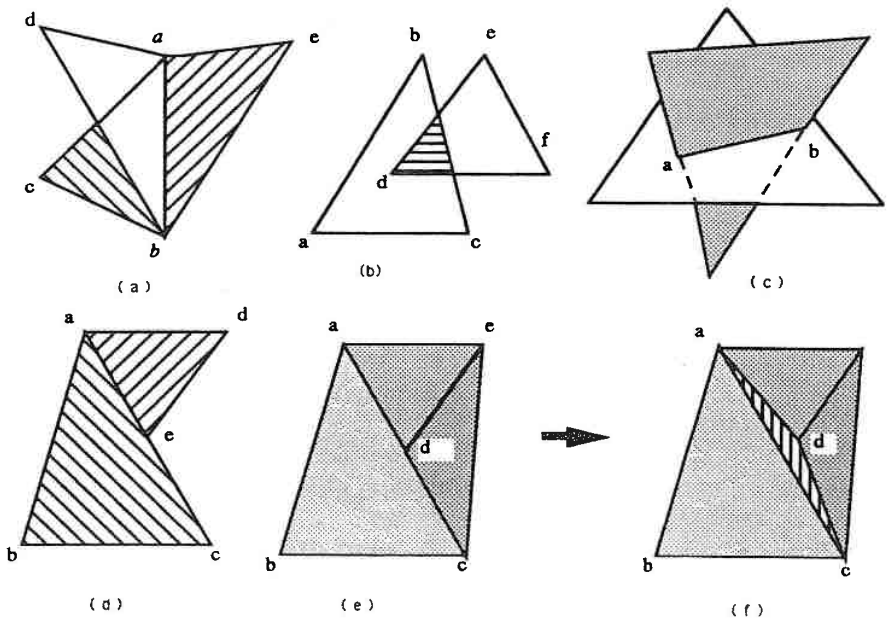


Figure 1.1 *Différents cas de figure entre deux triangles quelconques.*

Notation

Si un triangle T_1 appartenant à une T-surface simple a un côté commun avec un triangle T_2, alors T_2 est appelé triangle adjacent de T_1 et cette relation est notée $adj(T_1, T_2)$. Evidemment, cette relation est symétrique et l'on a donc :

$$adj(T_1, T_2) = adj(T_2, T_1)$$

Il convient de remarquer que deux triangles qui se touchent suivant l'un de leurs côtés ne sont pas nécessairement adjacents comme le montre l'exemple représenté sur la Figure 1.1.d où le triangle $T(a, e, d)$ n'est pas le triangle adjacent du triangle $T(a, b, c)$.

Remarque

La deuxième contrainte intervenant dans la définition de la notion de T-surface simple implique les conséquences suivantes :

- Deux triangles T_1 et T_2 appartenant à une même T-surface ne peuvent avoir une intersection d'aire non nulle (cf figure 1.1(b)).
- L'intersection de deux triangles T_1 et T_2 appartenant à une même T-surface ne peut être contenue, en partie ou en totalité, à l'intérieur de l'une de ces deux facettes (cf figure 1.1(c)).

- Une T-surface peut être ouverte ou fermée.
- Une T-surface peut être constituée de plusieurs morceaux disjoints ou joints par leurs sommets .
- Un triangle quelconque peut avoir au plus 3 triangles adjacents.
- Pour toute T-surface fermée, chaque facette triangulaire doit avoir trois facettes adjacentes et seulement trois. Ceci exclu en particulier le cas présenté sur la figure 1.1(e), mais si l'on ajoute le triangle $T(a, c, d)$, alors la surface représentée sur la Figure 1.1(f) est une T-surface topologiquement valide même si $T(a, c, d)$ est d'aire nulle).

T-surface simple fermée

Soit S une T-surface qui est topologiquement fermée et soit V le volume (supposé non nul) entouré par S dans l'espace 3D. S est définie comme une T-surface simple fermée si la condition suivante est vérifiée :

> Pour tout couple de points p et q appartenant à V $(p \neq q)$, il existe toujours un chemin $C(p, q)$ joignant p et q tel que :
>
> - $C(p, q)$ appartient à V.
> - pour tout point $z \in C(p, q)$ $(z \neq p \text{ et } q)$, il existe une constante $r > 0$ telle que tout point w verifiant $dist(w, z) \leq r$ appartient à V.

P-surface

Nous appellons P-surface toute surface constituée par un ensemble de facettes polygonales simples (définies plus loin). Toute P-surface est supposée respecter des contraintes semblables à celles introduites pour les T-surfaces.

T-solide

Un solide est un objet dont le bord (peau) sépare l'espace en deux régions: l'intérieur et l'extérieur (Autrement dit, la peau d'un solide est une surface fermée qui sépare l'espace en 2 régions). En principe, le volume d'un solide (l'espace entouré par la peau) ne peut être nul. Le solide est indéformable, c'est à dire, la distance entre deux de ses points est constante. Un solide peut être composé de plusieurs morceaux, contenant éventuellement des trous à l'intérieur de chacun de ces morceaux; il peut être convexe, concave ou quelconque, mais les excroissances filiformes ou planes sont interdites (cf Figure 1.2.b)

Dans ce qui suit, nous désignerons par $S(V)$ la peau de tout solide V et nous dirons que V est un T-solide si $S(V)$ est l'union d'une suite $S_s(V) = \{S_s^1, S_s^2, \ldots, S_s^n\}$ de T-surfaces simples fermées :

$$S(V) = \bigcup_{S_s^i \in S_s(V)} S_s^i$$

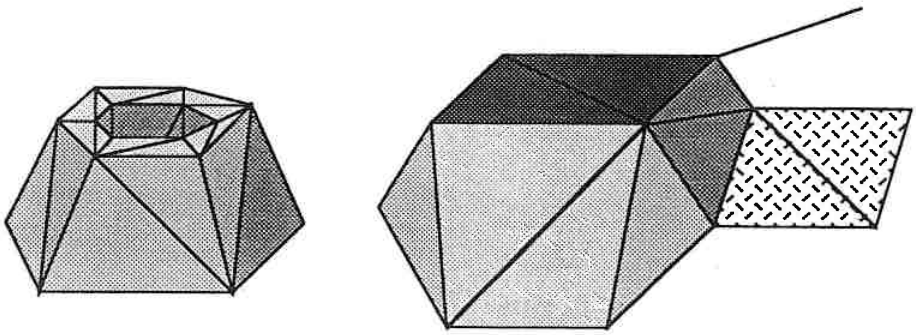


Figure 1.2 *(a) un T-solide correct; (b) un T-solide interdit.*

Remarques

Pour tout couple (S_s^i, S_s^j) de T-surfaces simples fermées appartenant à $S_s(V)$ on peut avoir les positions relatives suivantes :

- Les deux T-surfaces S_s^i et S_s^j sont disjointes.
- Les deux T-surfaces S_s^i et S_s^j partagent des points communs. Compte tenu que $S(V)$ est une T-surface simple, cela ne peut se faire que de l'une des deux façons suivantes :
  1. Comme l'indique la Figure 1.3.b, S_s^i et S_s^j peuvent se toucher suivant une arête géométriquement commune. Bien que géométriquement commune, une telle arête doit tout de même rester topologiquement distincte afin que l'on ne se trouve pas dans la situation où une arête est partagée par plus de 2 triangles, ce qui est incompatible avec le fait que $S(V)$ soit une T-surface simple.
  2. Comme l'indique la Figure 1.3.a, S_s^i et S_s^j peuvent se couper suivant une ligne $L = S_s^i \cap S_s^j$. Pour que $S(V)$ soit une T-surface simple, il est nécessaire que les arêtes (resp. les sommets) de L soient à la fois les arêtes (resp. les sommets) de S_s^i et de S_s^j; dans le cas contraire, $S(V)$ a des facettes qui se recoupent dans leur intérieur et n'est donc pas une T-surface simple.

L'exemple illustré dans la Figure 1.3 montre que l'ensemble $S_s(V)$ n'est pas unique pour un T-solide V donné mais l'union des éléments de $S_s(V)$ constitue toujours la peau $S(V)$ de V qui est unique géométriquement.

Pour clarifier la définition d'un T-solide V à l'aide de sa peau $S(V)$, nous donnons le lemme suivant :

Lemme 1

Deux T-surfaces simples fermées S_1 et S_2 qui se croisent peuvent toujours être transformées en un ensemble de T-surfaces simples fermées $S^* = \{S_1^*, S_2^*, \ldots, S_n^*\}$ tel que :

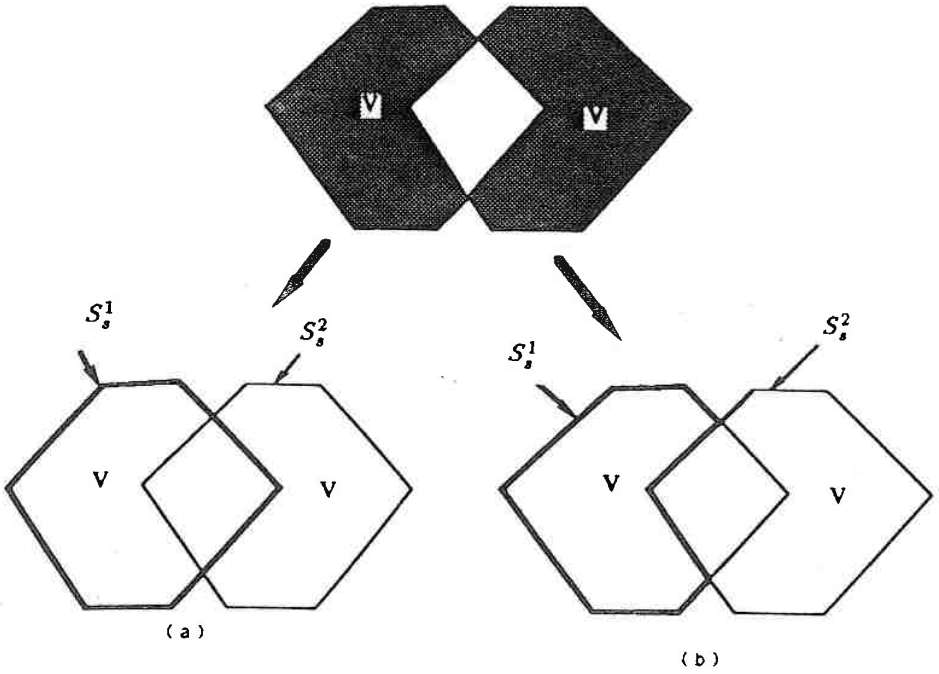


Figure 1.3 *(a) S_s^1 coupe S_s^2; (b) S_s^1 ne coupe pas S_s^2 mais joint par les arêtes communes*

- Deux T-surfaces quelconques S_i^* et S_j^* de S^* ne se croisent pas mais peuvent éventuellement être connectées par des arêtes géométriquement communes mais topologiquement distinctes.

- L'union de toutes les T-surfaces de S^* est géométriquement identique à $S_1 \cup S_2$.

Démonstration

Soient S_1 et S_2 deux T-surfaces simples fermées qui se croisent, soit $L = S_1 \cap S_2 = \{l_1, l_2, ..., l_m\}$ la ligne de l'intersection de S_1 avec S_2 qui peut être composée d'un ensemble de contours polygonaux disjoints et soient V_1 et V_2 les T-solides définis par S_1 et S_2. La ligne L découpe S_i $(i = 1, 2)$ en $m + 1$ T-surfaces simples $\{S_i(k) \ : \ k = 1, m + 1\}$ et l'on pose :

$$
\left|
\begin{array}{l}
S_1^{in} = \{S_1(k) \ : \ S_1(k) \text{ est à l'intérieur de } V_2\} \\
S_1^{out} = \{S_1(k) \ : \ S_1(k) \text{ est à l'extérieur de } V_2\} \\
S_2^{in} = \{S_2(k) \ : \ S_2(k) \text{ est à l'intérieur de } V_1\} \\
S_2^{out} = \{S_2(k) \ : \ S_2(k) \text{ est à l'extérieur de } V_1\}
\end{array}
\right.
$$

Il est évident que S_1^{in} et S_2^{out} (resp. S_1^{out} et S_2^{in}) constituent la peau de $(V_1 - V_2)$ (resp. $(V_2 - V_1)$). Les T-surface de S_1^{in} (resp de S_1^{out}, de S_2^{in} et de S_2^{out}) ne peuvent pas se croiser mais peuvent éventuellement partager les arêtes appartenant à L, donc si l'on colle chaque T-surface de S_1^{in} avec les T-surfaces de S_2^{out} sur le long de leur frontière commune, on obtient une T-surface correspondant à $S_1^{in} \cup S_2^{out}$ qui ne peut être qu'une T-surface fermée simple ou qu'un ensemble de T-surfaces simples fermées qui sont jointes par des arêtes de L. Symétriquement pour S_1^{out} et S_2^{in}. On peut constater que la T-surface $S_1^{in} \cup S_2^{out}$ (resp. $S_2^{in} \cup S_1^{out}$) est géométriquement identique à $S(V_1 - V_2)$ (resp. $S(V_2 - V_1)$) car on a la relation suivante :

$$\left| \begin{array}{lcl} S(V_1 - V_2) \cup S(V_2 - V_1) & = & (S_1^{in} \cup S_2^{out}) \cup (S_1^{out} \cup S_1 2^{in}) \\ (S_1^{in} \cup S_2^{out}) \cup (S_1^{out} \cup S_2^{in}) & = & (S_1^{in} \cup S_1^{out}) \cup (S_1 2^{in} \cup S_2^{out}) = S_1 \cup S_2 \end{array} \right.$$

donc le collage des T-surfaces de S_1^{in} avec celles de S_1^{out} (resp. le collage des T-surfaces de S_2^{in} avce celles de S_2^{out}) donne l'ensemble des T-surfaces simples fermées cherché S^*.

Corollaire 1

pour un T-solide quelconque V, il existe toujours un ensemble de T-surfaces simples fermées $S = \{S'_1, S'_2, \ldots, S'_n\}$ tel que :

- Deux T-surfaces quelconques S'_i et S'_j de S ne se croisent pas mais peuvent éventuellement être géométriquement connectées par des arêtes communes.
- L'union de toutes les T-surfaces de S est géométriquement identique à $S(V)$.

Démonstration

Ce corollaire peut être démontré à l'aide de la procédure suivante décrite en pseudo-C et utilisant le lemme précédent. Soit $S(V) = \{S_1, S_2, ..., S_n\}$ l'ensemble de T-surfaces simples fermées décrivant la peau d'un T-solide V.

```
début
  S_T = S_1

  pour-tout (T-surface S_i sauf S_1)
  {
    pour-tout (T-surface S'_j de S_T)
    if (S'_j coupe S_i)
    {
      /*transformer S_i et S' en T-surfaces
        simples fermées non croisées (Lemme 1)*/
      S" = S'_j + S_i;
      ajouter toutes les T-surfaces de S" à S_T;
    }
  }
```

return(S_T);
fin

A l'issue de cette procédure, S_T est un ensemble de T-surfaces simples fermées non croisées (mais se touchant).

En géométrie, deux T-solides différents ne peuvent avoir la même peau, sinon, ils sont le même T-solide .

Corollaire 2

Un ensemble quelconque de T-surfaces simples fermées définit dans l'espace un seul T-solide dont la peau est constituée de ces T-surfaces simples fermées.

Démonstration

Soient S_1, S_2 deux T-surfaces simples fermées (disjointes ou croisées) et V_1, V_2 deux T-solides définis respectivement par S_1 et S_2. De la démonstration du lemme ci-dessus, on déduit facilement qu'il existe un seul T-solide V tel que $S(V)$ soit géométriquement identique à $S_1 \cup S_2$ et tel que $V = (V_1 - V_2) \cup (V_2 - V_1)$. On peut donc démontrer le corollaire 2 par la procédure suivante décrite en pseudo-C :

Soit $S = \{S_1, S_2, \ldots, S_n\}$ un ensemble de T-surfaces simples fermées et soit $V = \{V_1, V_2, \ldots, V_n\}$ l'ensemble des T-solides tels que $S_i = S(V_i)$.

début
$V_T = V_1$;
pour-tout (T-surface V_i de V **sauf** V_1)
{
- calcul de $\bigcup_{V_j \in V_T} V_j$;
- U = $\bigcup_{V_j \in V_T} V_j$;
- calcul de $V_i - U$ et $U - V_i$;
- ajout de $V_i - U$ et de $U - V_i$ à V_T;

}
return(V_T);
fin

A l'issue de cette procédure, V_T est le T-solide cherché: $S(V_T) = \bigcup_{S_i \in S} S_i$.

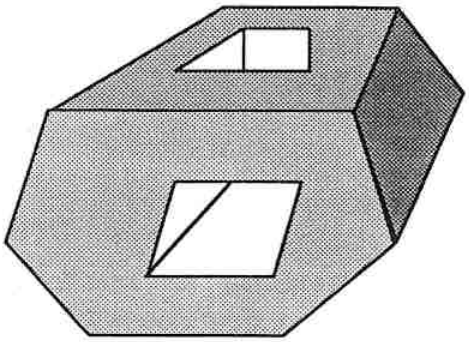


Figure 1.4 *un P-solide avec des trous.*

Remarque

D'après le corollaire 2, Une série quelconque de T-surfaces simples fermées peut être considérée comme la peau d'un T-solide, plus précisément, un ensemble quelconque de T-surfaces simples fermées définit dans l'espace 3D un et seulement un T-solide. Compte tenu du corollaire 1, dans ce mémoire, pour faciliter le calcul tout T-solide manipulé sera défini par un ensemble de T-surfaces simples fermées tel que deux T-surfaces quelconques de cet ensemble ne se croisent pas.

P-solide

un P-solide est un solide dont la peau est une surface fermée, formée de facettes planes polygonales quelconques. Ces facettes peuvent contenir à leurs intérieurs des trous polygonaux (cf figure 1.4). Comme nous avons fait pour la définition d'un T-solide, nous supposons que tout P-solide est aussi indéformable et peut être composé de plusieurs morceaux, mais les excroissances filiformes ou planes sont interdites. La peau d'un P-solide est toujours une P-surface.

Notion de polygone simple

Définition

Soit ∂P un contour polygonal fermé situé dans un plan de l'espace 3D et entourant une région polygonale P de surface finie non nulle. Nous dirons que P est un polygone simple si, pour tout couple de points p_1 et p_2 appartenant à l'intérieur $\overset{\circ}{P}$ de P, il existe au moins un chemin $C(p_1,p_2)$ allant de p_1 vers p_2 et tel que :

$$p \in C(p_1,p_2) \Longleftrightarrow \begin{cases} p \in \overset{\circ}{P} \\ \quad \text{ou bien} \\ p \equiv \text{ sommet de } \partial P \end{cases}$$

On trouvera sur la Figure 1.5 quelques exemples et contre exemples de polygones simples.

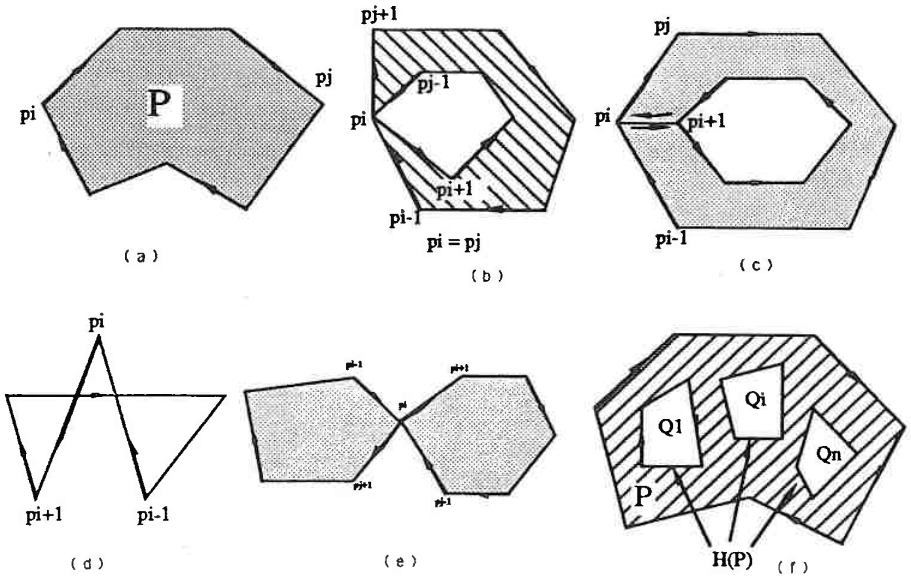


Figure 1.5 *Exemples de polygones: (a), (b), (c) et (e) sont les polygones simples ; (d) ne l'est pas.*

En général, le contour polygonal est défini par un ensemble de points ordonnés correspondant aux sommets du contour; lorsque l'on se déplace sur le contour, la région polygonale ainsi définie doit toujours être du même côté du contour orienté.

Polygone simple avec trous

En pratique, l'intérieur d'un polygone simple peut éventuellement contenir des trous polygonaux $\{Q_1, Q_2, ...Q_m\}$. Nous conviendrons de noter $H(P)$, l'ensemble des trous polygonaux contenus dans un polygone simple P :

$$H(P) = \{Q_1, Q_2, ...Q_m\}$$

Lorsque $H(P)$ est non vide, P est appelé "polygone simple avec trous".

1.2 Structure de données des entités géométriques

Remarque préliminaire

Les travaux décrits dans cette thèse font partie du project *GOCAD* et toutes nos structures de données sont celles de la base de données géométriques réalisée dans ce projet. Cette base de données est décrite en *Langage C*. Nous proposons dans ce qui suit un bref rappel de celles-ci.

Vertex

La notion de vertex est la structure de données la plus simple du système de *GOCAD*. Un vertex est un point de l'espace 3D auquel on associe des informations.

L'implémentation d'un tel ***vertex*** en ***Langage C*** est la suivante:

```
#define COOR_t  float


typedef struct VERTEX_t
         {
              COOR_t  x, y, z  ;
              short   movable  ;
              long    stuff    ;
          } VERTEX_t ;
```

Les significations des champs de cette structure sont les suivantes :

- (x, y, z) sont les trois coordonnées de ce sommet
- ***movable*** est un drapeau spécifant les modes de déplacement autorisées pour le ***vertex*** lorsque l'on applique l'algorithme *DSI* (voir [16]).

Noeud de vertex

Un VERTEX_NODE_t est défini comme une structure dans laquelle il y a un pointeur vers un ***vertex*** :

```
typedef struct VERTEX_NODE_t
        {
           struct VERTEX_NODE_t  * next    ;
                  VERTEX_t       * p_vrtx  ;
        } VERTEX_NODE_t ;
```

Les raisons de cette définition sont les suivantes :

- un ***vertex*** peut être partagé par plusieurs objets différents. Par exemple, sur la Figure 1.6, les vertex $(a_3, a_4, a6)$ sont partagés par les surfaces S_a et S_b. Ces deux surfaces "pointent" vers ces ***vertex*** au travers de la structure VRTX_NODE_t.
- Deux objets différents peuvent partager des ***vertex*** commums. Comme le montre la Figure 1.6. Si S_b est translatée d'une distance 'd', alors les trois sommets commums sont aussi translatés de la même distance d. Il est évident qu'après cette translation S_a reste liée à S_b puisqu'elles ont les 3 ***vertex*** $(a_3, a_4, a6)$ en commun; cette structure sera utilisée pour le découpage de triangles dans le chapitre 2.1.

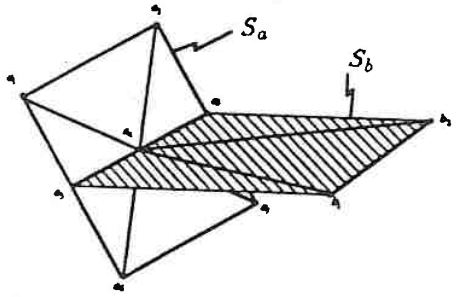


Figure 1.6 *Sa est attachée avec Sb.*

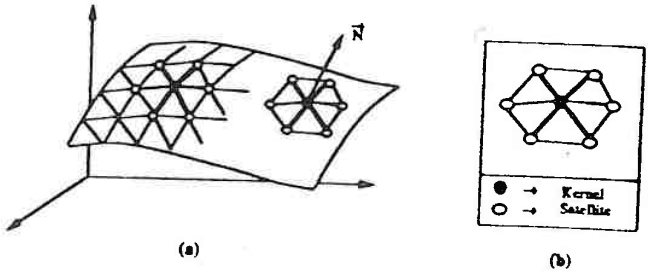


Figure 1.7 *Structure atomique.*

Atome

Afin d'accélérer la rapidité de l'interpolation d'une T-surface par la méthode *DSI* (voir [16]), il est nécessaire d'avoir un accés rapide à tous les vertex connectés avec un vertex donné. Pour cette raison, on considère les liens entre un vertex donné et ceux qui se trouvent autour de lui comme une structure atomique (cf Fig 1.7)

- chaque vertex "k" est considéré comme le noyau de l'atome k ;
- l'orbite $\Lambda(k)$ de l'atome "k" est constitué par les vertex ayant un lien direct avec "k". Tous ces vertex sont appelés les satellites de noyau (vertex) "k".

L'implémentation de cette structure en *Langage C* est la suivante :

```
typedef struct ATOM_t
        {
           struct VERTEX_t   * p_vrtx     ;
           struct ATOM_t     ** p_sat     ;
           int                  nb_sat    ;
           COOR_t               Nx,Ny,Nz;
           CONTRAINT_t        * p_constaint ;
           long                 stuff     ;
        } ATOM_t ;
```

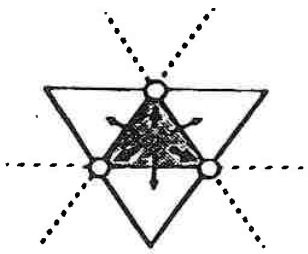


Figure 1.8 *Structure de TRGL_t.*

Les significations de certain champs de cette structure sont les suivantes :

- ***p_vrtx*** est un pointeur vers le vertex noyau de l'atome;
- (N_x, N_y, N_z) désignent les 3 composants du vecteur orthogonal à la surface au point pointé par ***p_vrtx***; ce tableau correspond en fait au codage de l'orbite $\Lambda(k)$.
- ***p_sat[i]*** est supposé contenir l'adresse de l'atome correspondant au i-éme satellite du noyau pointé par ***p_vrtx***.

Triangle

Dans ce qui va suivre, un triangle est décrit par deux groupes de pointeurs :

- 3 pointeurs ***p_atom[i]*** vers les 3 atomes dont les noyaux correspondant aux 3 sommets du triangle.
- 3 pointeurs ***p_trgl[i]*** vers les 3 triangles adjacents. Certains de ces pointeurs peuvent éventuellement être $NULL$.

L'implémentation en *Langage C* est la suivante:

```
typedef struct TRGL_t
        {
           struct TRGL_t   * next        ;
           struct TRGL_t  ** p_trgl[3]   ;
           struct ATOM_t  ** p_atom[3]   ;
           long              stuff       ;
        } TRGL_t ;
```

Pour la consistance de cette structure, les pointeur p_trgl[ ] et p_atom[ ] sont supposés obéir aux règles suivantes:

- les deux groupes de pointeurs sont respectivement numérotés de 0 à 2.
- le côté opposé de l'atome p_atom[i] est commun avec le triangle voisin pointé par p_trgl[i].
- le triangle est supposé être orienté par l'ordre de rangement de ses 3 sommets dans *p_atom*[ ].

Dans certaines circonstances, ces règles permettent d'accéder plus rapidement aux triangles voisins. c'est par exemple, le cas lorsqu'étant donné un sommet a_1 du triangle T, on veut chercher le triangle adjacent qui ne contient pas le sommet a_1.

Commentaires

1. Dans la structure TRGL_t, les 3 sommets du triangle sont modélisés par 3 pointeurs vers 3 atomes dans lesquels il y a des pointeurs vers les vertex correspondants. La question qui se pose est: pourquoi accéder indirectement plutôt qu'avoir des pointeur vers les 3 vertex correspondant aux 3 sommets; la raison est que, lorsqu'on veut visualiser une T-surface, non seulement on a besoin des coordonnées des sommet des triangles, mais aussi les composants des vecteurs orthogonaux à la T-surface en chacun des sommets des triangles et ces normales sont rangées non pas dans les vertex mais dans les atomes.
2. On peut imaginer qu'il ne vaille pas la peine de construire la structure TRGL_t parce que la structure ATOM_t contient toutes les informations nécessaires pour reconstruire les triangles correspondants. En réalité, ce n'est pas le cas, parce que l'on n'a pas de moyen de savoir si le triangle formé par un atome et deux de ses satellites constitue un vrai triangle appartenant à la T-surface ou un trou triangulaire.
3. Dans la structure TRGL_t, les 3 pointeurs vers les trois triangles adjacents constituent un grand avantage pour les manipulations des T-surfaces. A l'aide de cette structure :
   - il est facile de se déplacer d'un triangle à l'autre.
   - l'orientation des triangles liés avec un triangle donné peut s'effectuer rapidement en utilisant une procédure récursive (cf page 101)

Polygone

Ici, les polygones traités sont des polygones simples. Un polygone simple est défini comme une liste de VERTEX_NODE_t correspondant aux sommets ordonnés du polygone. Le premier élément et le dernier élément de cette liste constituent implicitement un côté du polygone.

L'implémentation en Langage C est la suivante :

```
typedef struct POLYGON_t
        {
           struct POLYGON_t   * next      ;
           VERTEX_NODE_t      * p_line    ;
           long                 stuff     ;
        } POLYGON_t ;
```

T-surface (T-solide)

Une T-surface dans le système de *GOCAD* est définie comme un ensemble de triangles non ordonnés dont les sommets eux-mêmes constituent une liste (V-set). On peut constater qu'une telle T-surface peut être composée de plusieurs morceaux, chaque morceau peut être ouvert ou fermé.
La notion de T-surface est implémentée en Langage C de la façon suivante :

```
typedef struct TSURF_t
        {
           struct TSURF_t   * next       ;
           int                open       ;
           int                nb_trgl    ;
           int                nb_atom    ;
           TRGL_t           * p_trgl     ;
           VSET_t           * p_vset     ;
           ATOM_t           * p_atom     ;
           Toctree_t        * p_toctree  ;
           long               stuff       ;
        } TSURF_t ;
```

Les significations des principaux champs de cette structure est la suivante :

- *open* est drapeau qui spécifie si la T-surface est ouverte ou fermée:

  $=0 \Longrightarrow$ la T-surface est considérée comme fermée. Si la T-surface est composée de plusieurs morceaux, alors chacun d'eux doit être fermé. Une T-surface fermée peut être considérée comme un T-solide.

  $<0 \Longrightarrow$ la T-surface doit être considérée comme ouverte. Si la T-surface est composée de plusieurs morceaux, alors elle est considérée comme ouverte lorsqu'au moins un morceau est ouvert.

  $>0 \Longrightarrow$ on ne sait pas si la T-surface est ouverte ou fermée.

- *p_vset* est un pointeur vers la liste de VERTEX_NODE_t correspondant aux sommets de la T-surface.

- *nb_trgl* (resp. *nb_atom*) indique le nombre de triangles (resp. atomes) de la T-surface.

- *p_tocitree* est le pointeur vers le T-octree des triangles (cf chapitre suivant).

Chapitre

2

Construction et utilisation d'un T-octree

2.1 Introduction

Dans la manipulation et la modélisation des T-surfaces (cf Partie II), il existe de nombreux problèmes géométriques pour lesquels il est nécessaire d'accéder de façon efficace aux triangles d'une T-surface donnée; par exemple, on peut citer les problèmes suivants :

- intersection d'un triangle avec une T-surface.
- intersection de deux T-surfaces .
- opérations booléennes sur des solides dont les surfaces frontières sont constituées par des T-surfaces.
- intersection d'une T-surface par une droite.
- position d'un point par rapport à une T-surface.
- etc...

Pour les problèmes de ce genre, il existe toujours un algorithme naïf consistant à parcourir tous les triangles afin d'identifier les triangles concernés. La notion de T-octree, présentée dans ce chapitre, permet d'améliorer les performances en n'accédant qu'aux triangles qui ont une certaine chance d'être concernés.

2.2 Notion de T-octree

Subdivision de boîte et Octree associé

Comme l'indique la Fiqure 2.1, considérons une boîte B parallélepipédique dont les arêtes sont supposées être paralleles aux axes (ox, oy, oz) de l'espace 3D. Cette boîte peut être subdivisée

en 8 sous-boîtes homogènes $B(i_1)$ numérotées de $i_1 = 0$ à $i_1 = 7$ (cf Fig 2.1). En fait, ce processus de subdivision peut être effectué récursivement en considérant qu'une boîte quelconque $B(i_1, i_2, ..., i_n)$ génère 8 sous-boîtes numérotées de la façon suivante:

$$B(i_1, i_2, ..., i_n) \Longrightarrow \begin{cases} B(i_1, i_2, ..., i_n, 0) \\ B(i_1, i_2, ..., i_n, 1) \\ B(i_1, i_2, ..., i_n, 2) \\ B(i_1, i_2, ..., i_n, 3) \\ B(i_1, i_2, ..., i_n, 4) \\ B(i_1, i_2, ..., i_n, 5) \\ B(i_1, i_2, ..., i_n, 6) \\ B(i_1, i_2, ..., i_n, 7) \end{cases}$$

Comme l'illustre la figure 2.1, le processus de subdivision peut être décrit par un arbre octal respectant les deux règles suivantes:

1. la boîte initiale est représentée par la racine de l'arbre.
2. chaque boîte $B(i_1, i_2, ..., i_n)$ est représentée par un noeud du n^{eme} niveau de l'arbre.

La deuxième règle ci-dessus implique que :

- chaque noeud de l'arbre possède 7 noeuds frères, sauf évidemment si le noeud est la racine de l'arbre.
- chaque noeud engendre huit noeuds fils, sauf évidemment si le noeud est une feuille.
- le niveau n de l'arbre est composé de 8^n noeuds.

Par définition, cet arbre est appellé *octree* (voir [1] [33]). On peut remarquer que tout octree ainsi défini possède une infinité de niveaux et n'a aucune feuille; en pratique, celà est inacceptable et nous devons définir des règles permettant d'arrêter le processus de subdivision et générant des feuilles afin d'obtenir un arbre fini.

T-octree : Présentation sur un exemple

Comme le montre la figure 2.2(a), considérons une T-surface composée de 15 triangles[1] et, comme indiqué sur la figure 2.2(a), soit B la plus petite boîte parallélépipédique contenant la surface et ayant ses côtés parallèles aux axes (ox, oy, oz) de l'espace 3D. On peut appliquer à cette boîte le processus de subdivision décrit au paragraphe précédent en mettant dans chaque sous-boîte le numéro d'identification des triangles qui l'intersectent. Afin d'obtenir un octree fini, nous utiliserons les deux règles suivantes :

1. Si le nombre de triangles contenu dans une boîte est inférieur à un nombre donné fixé à priori, alors cette boîte est une feuille de l'octree et tous ses successeurs doivent être supprimés de l'octree.

[1] On ne manquera pas de noter que cette surface possède un trou triangulaire situé en son centre.

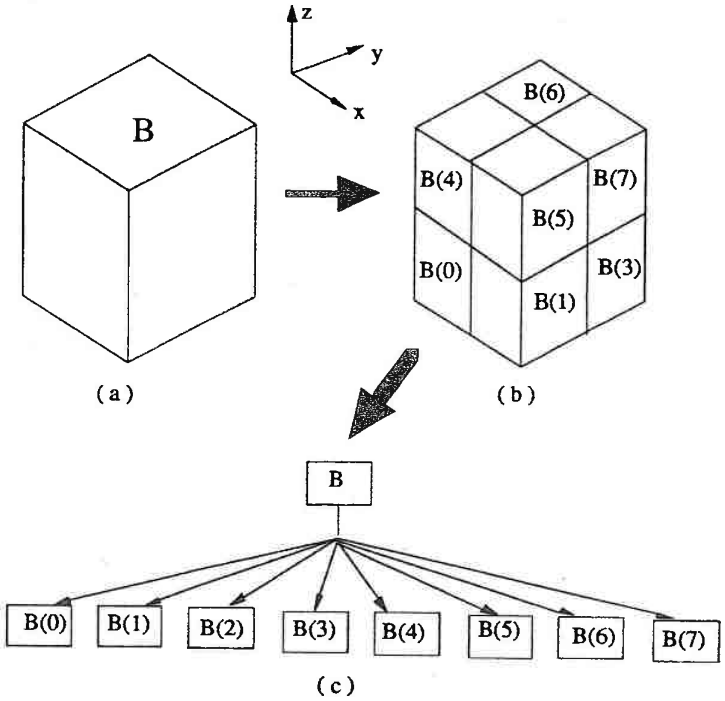


Figure 2.1 *Subdivision octale.*

2. Si une boîte a une taille inférieure à un seuil donné fixé à l'avance, alors cette boîte doit être considérée comme une feuille de l'octree et tous ses successeurs doivent être supprimés de l'octree.

Si nous appliquons ces deux règles à la surface présentée sur la Figure 2.2(1.a), alors elles génèrent les boîtes représentées sur la Figure 2.2(1.c) et l'octree représenté sur la Figure 2.2(2).

Un tel octree associé à une T-surface sera appelé un "**T-octree**".

2.3 Construction d'un T-octree

2.3.1 Structure de données TOCTREE_t

Dans le projet de *GOCAD*, chaque noeud de T-octree est implémenté comme une structure de données dont le type TOCTREE_t est défini comme suit :

```
typedef struct TOCTREE_t
        {
           typedef struct TOCTREE_t   * parent  ;
           typedef struct TOCTREE_t  ** child   ;
           int                         nb_trgl ;
        } TOCTREE_t ;
```

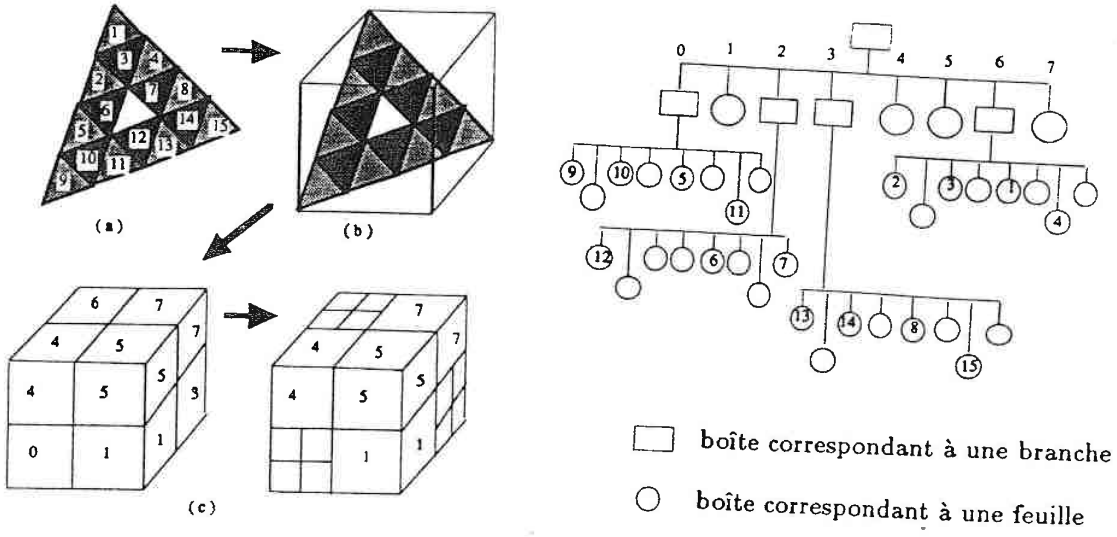


Figure 2.2 *(1) Subdivision octale de la boîte initiale; (2) T-octree associé.*

Les différents champs de cette structure ont les significations suivantes :

- ***nb_trgl*** est un entier dont la valeur absolue est égale au nombre de triangles qui intersectent la boîte correspondant au noeud.
- ***parent*** est un pointeur vers le noeud parent.
- ***child*** est un pointeur qui est considéré :
  - comme un pointeur vers un tableau de 8 pointeurs ***child[i]*** vers les enfants de ce noeud si ce noeud correspond à une branche de l'arbre.
  - comme un pointeur vers une sous-liste de triangles intersectant la boîte associée au noeud si le noeud correspond à une feuille de l'arbre.

De plus, afin de faciliter l'idendification du type de noeud, on impose les deux règles suivantes:

- si le champ ***nb_trgl*** de la structure TOCTREE_t contient une valeur négative, alors ce noeud est considéré comme une branche.
- si le champ ***nb_trgl*** de TOCTREE_t contient une valeur positive (ou 0), alors ce noeud est de type feuille.

Ce champs ***nb_trgl*** est rempli au moment de la construction du T-octree.

Structure de données TRGL_NODE_t

Jusqu'à présent, nous n'avons pas explicité la liste des triangles stockés dans les noeuds terminaux du T-octree. Une telle liste sera codée comme une liste chaînée de "noeuds-triangles" contenant les triangles et dont le type est défini ci-dessous :

```
typedef struct TRGL_NODE_t
        {
           struct TRGL_NODE_t * next   ;
           TRGL_t             * p_trgl ;
         } TRGL_NODE_t;
```

Les deux champs de cette structure ont la signification suivante :

- ***next*** est un pointeur vers le "noeud-triangle" suivant correspondant au prochain triangle de la liste.
- ***p_trgl*** est un pointeur vers le triangle de la liste attaché au "noeud-triangle".

On peut se poser la question de savoir si l'on n'aurait pas pu utiliser directement la structure TRGL_t pour coder les listes de triangles contenues dans les feuilles du T-octree. En effet, la structure TRGL_t contient, elle aussi, un champ ***next*** que l'on pourrait songer à utiliser pour construire de telles listes chaînées. En fait, celà n'est pas possible car :

- ce champ est déjà utilisé pour le codage des T-surfaces et son contenu ne peut donc pas être changé,
- un triangle peut intersecter plusieurs boîtes correspondant aux noeuds terminaux du T-octree et il doit alors se trouver dans plusieurs listes or, dans la structure TRGL_t, on ne dispose que d'un seul champ ***next***.

Pour ces deux raisons nous sommes obligés d'introduire la structure TRGL_NODE_t pour construire les listes chaînées de triangles contenus dans les noeuds d'un T-octree.

Algorithme de construction d'un T-octree[2]

L'algorithme de construction du T-octree associé à une T-surface est réalisé par la fonction ***Build_Toctree***() et peut être décomposé en deux fonctions :

- la fonction ***Build_Toctree***() elle-même qui crée la boîte racine du T-octree et lance la procédure recursive de construction du T-octree.
- la fonction ***Split_Box_of_Toctree_Recursively***() qui est appelée par ***Build_Toctree***() pour diviser la boîte racine en 8 sous-boîtes. Cette fonction est récursive et s'appele elle-même pour diviser les 8 sous-boîtes qu'elle génère.

nous proposons dans ce qui suit une version "pseudo-C" de ces deux fonctions :

[2]cet algorithme nécessite de tester l'intersection d'un triangle avec un cube (cf Chapitre 1.3)

```
TOCTREE_t * Build_Toctree( p_tsurf )
{
   /* Quelques contrôles */
   if( p_tsurf-> p_toctree
      && le T-octree est déjà à jour)  return (p_tsurf-> p_toctree) ;

   /* Création d'une liste de TRGL_NODE_t */

   Soit T la liste de triangles de la T-surface;
   Soit Θ la liste de TRGL_NODE_t correspondant à T;

   pour-tout( triangle τ ∈ T)
   {
      Création d'un TRGL_NODE_t θ ;
      θ.p_trgl = adresse de τ ;
      Insersion de θ à Θ ;
   }

   /* Création de la boîte racine */

   allouer la mémoire pour la boîte racine pointée par B ;
   p_tsurf-> p_toctree = B ;
   B-> nb_trgl = p_tsurf->nb_trgl ;
   B-> parent = NULL ;
   B-> child = Θ ;

   Détermination de la géométrie de la boîte racine B:

        {x_min, y_min, z_min, d_x, d_y, d_z}

   /* Division de la boîte racine B en sous-boîtes */

   Split_Box_of_Toctree_recursively(0, B, x_min, y_min, z_min, d_x, d_y, d_z) ;
   return(p_tsurf->p_toctree) ;
}


void Split_Box_of_Toctree_recursively(level, B, x_min, y_min, z_min, d_x, d_y, d_z)
{
   /* arrêter la processus récursif pour produire une feuille */

   if( B->nb_trgl < MAX_TRGL || level > MAX_LEVEL)  return ;

   /* diviser la boîte B en 8 sous-boîtes: {B[0], B[1], ..., B[7]} */
```

$d_x = d_x/2$;
$d_y = d_y/2$;
$d_z = d_z/2$;

allouer la mémoire pour $\{B[0], B[1], \ldots, B[7]\}$;

pour-chaque(sous-boîte $B[i]$**)**
{
- Construire la sous-liste Θ_{sous} de triangles intersecté $B[i]$;
- $B[i] -> child = \Theta_{sous}$;
- Déterminer la géométrie de $B[i]$: $\{x[i], y[i], z[i], d_x, d_y, d_z\}$;

}

/* construire le T-octree récursivement */

pour-chaque(sous-boîte $B[i]$ **)**
{
Split_Box_of_Toctree_Recursiveley$(level + 1, B[i], x[i], y[i], z[i], d_x, d_y, d_z)$;
}
return ;
}

On ne manquera pas de remarquer que cet algorithme nécessite de tester si un triangle donné intersecte ou non un parallélépipède rectangle ayant ses côtés parallèles aux axes de coordonnées; ce problème sera résolu ultérieurement au chapitre 1.3.

2.4 Evaluation de l'algorithme

2.4.1 Complexité de l'algorithme

Supposons que la distribution des triangles de la T-surface soit assez homogène dans l'espace 3D et que les triangles aient à peu près la même taille comme c'est souvent le cas dans la plupart des applications pratiques en géologie ou en médecine.

Soient B un noeud de type branche et $L(B)$ (resp. $N(B)$) la liste (resp. le nombre) de triangles attachée à B. Si l'on divise B en 8 sous-boîtes $\{B_0, B_1, B_2, ..., B_7\}$, alors les triangles contenus dans B sont répartis dans les 8 sous-listes $\{L_0, L_1, L_2, ..., L_7\}$ correspondantes. Pour chacune de ces sous-boîtes B_i ($i = 0, 7$), le calcul principal consiste à tester l'intersection de chacun des $N(B)$ triangles contenus dans B avec le parallélépipède correspondant à B_i et le nombre total de ces tests pour les 8 sous-boîtes est donc égal à $8N$.

Soient $N(B_i)$ le nombre de triangles contenus dans B_i; compte tenu que les triangles sont supposés distribués de façon homogène dans B, on peut supposer que ce nombre est proportionnel à $N(B)$:

$$N(B_i) \approx h \times N(B)$$

On en déduit que l'on peut écrire :

$$\sum_{i=0}^{7} N(B_i) \quad \approx \quad 8h \times N(B) \quad = \quad k \times N(B) \qquad \text{avec : } k > 1$$

Soit N le nombre total de triangles d'une T-surface. Afin de construire le premier niveau du T-octree associé, on effectue $8N$ tests et la somme des nombres de triangles dans les 8 sous-boîtes est alors égale à kN. Si l'on répète cette procédure récursivement, au $n^{\text{ième}}$ niveau, le nombre total des triangles contenus dans tous les noeuds de ce niveau est alors égal à $k^n N$ et le nombre total de tests effectués est égal à :

$$8 \cdot N + 8k \cdot N + 8k^2 \cdot N \ldots + 8k^{n-1} \cdot N = \frac{8N(k^n - 1)}{(k-1)}$$

Dans la pratique, on fixe la profondeur maximale (n_{max}) du T-octree et l'algorithme fonctionne, en moyenne, alors de façon linéaire (voir [1]).

2.4.2 Analyse de performance

Afin d'évaluer l'efficacité réelle de l'algorithme de construction d'un T-octree, nous avons testé celui-ci sur trois types de surfaces générées de façon automatique pour lesquelles la profondeur de l'octree a été fixée à 7 et le nombre maximal de triangles par feuille à 5 :

- La T-surface S_1 représentée sur la Figure 2.3.a est de forme rectangulaire et a tous ses triangles situés dans un même plan. Les triangles ont la même taille et possèdent tous un côté parallèle au plan (xoy).

- La T-surface S_2 représentée sur la Figure 2.3.b est l'union de la surface S_1 définie ci-dessus avec la surface S_1' obtenue par rotation de S_1 d'un angle de $(180 - 2 \cdot \theta)$ degrés autour de l'axe AB.

- La surface S_3 est la peau d'une sphère dont les facettes triangulaires ont approximativement la même taille.

Avec ces trois types de T-surfaces, nous avons obtenu les résultats présentés dans le tableau suivant sur une machine fonctionnant à la vitesse de 7 Mips :

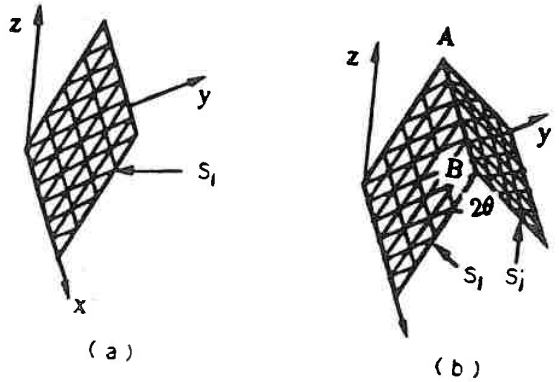


Figure 2.3 *(a) S_1 est une T-surface plate rectangulaire; (b) S_2 est constitué de S_1 et S_1'.*

Performance de la construction de T-octree				
	nb_trgl	**temps de calcul**	**nb_trgl moyen dans les feuilles**	**mémoires occupées**
S_1	500	$2s$	7	$13k$
	1000	$3.5s$	8	$44k$
	1500	$4.5s$	10	$49k$
	2000	$6s$	12	$65k$
	3000	$9s$	13	$85k$
	4000	$11s$	17	$103k$
S_2	1000	$3s$	7	$48k$
	2000	$5s$	9	$70k$
	3000	$8s$	11	$89k$
	4000	$11s$	15	$119k$
	6000	$15s$	16	$154k$
	8000	$20s$	19	$193k$
S_3	500	$2.5s$	5	$13k$
	1000	$4.5s$	6	$60k$
	2000	$10s$	7	$99k$
	3000	$13s$	15	$140k$
	4000	$16s$	15	$181k$

Remarque

- Comme le montre le tableau ci-joint, le temps de calcul varie quasi-linéairement en fonction du nombre de triangles; ceci vérifie l'analyse de complexité ci-dessus.
- Il est possible de limiter l'algorithme de construction d'un T-octree à une partie d'une T-surface en précisant la boîte de départ. Ceci peut être très intéressant dans les cas où l'espace de travail est limité.

2.5 Utilisation du T-octree

2.5.1 Recherche des intersections d'un segment avec une T-surface

Supposons que l'on ait à déterminer l'intersection d'un segment E avec une T-surface S. En supposant que E n'est pas tangent à la surface S, on est assuré que cette intersection est constituée par l'ensemble fini des points d'intersections de E avec les facettes de S. La recherche de ces points peut alors être effectuée à l'aide de la procédure récursive suivante appliquée récursivement à chaque noeud $B(i_1, i_2, ..i_n)$ du T-octree associé à la T-surface S :

- tout d'abord on initialise à 0 tous les champs ***stuff*** des triangles composant la T-surface (voir page 20 la structure TRGL_).
- si E n'intersecte pas la boîte correspondante, alors E n'intersecte pas non plus les sous-boîtes dont la racine est $B(i_1, i_2, ..., i_n)$.
- si E intersecte la boîte correspondante, alors il est nécessaire d'examiner le type de $B(i_1, i_2, ..., i_n)$. Compte tenu de ce qui a été dit précédemment, le type de ce noeud est indiqué par le champ ***nb_trgl*** de la structure contenant $B(i_1, i_2, ..., i_n)$:
  - si $nb\_trgl = 0$, alors $B(i_1, i_2, ..., i_n)$ est une feuille vide.
  - si $nb\_trgl > 0$, alors $B(i_1, i_2, ..., i_n)$ est une feuille contenant une liste de triangles. Soit T_j un triangle de cette liste; le fait que T_j peut être inclu dans plusieurs noeuds du T-octree nous conduit à considérer les deux cas suivants :
    1. l'intersection de E avec T_j a été déjà calculée; en pratique, cette information est stockée dans le champ ***stuff*** de la structure de type TRGL_t dans laquelle est rangé le triangle T_j ($stuff = 1$). Dans ce cas, on passe au triangle suivant sans rien faire.
    2. sinon, on calcule $E \cap T_j$ et l'on marque T_j comme ayant déjà été considéré pour le calcul de l'intersection ($stuff = 1$).
  - si $nb\_trgl < 0$, alors ce noeud est type branche (non terminal). On exécute alors récursivement la même procédure pour chacune des huit sous-boîtes $B(i_1, i_2, ..., 0)$,..., $B(i_1, i_2, ..., 7)$ de ce noeud.

Cet algorithme doit être appliqué à la racine B du T-octree. Il est implémenté par la fonction *Segment_Inter_Tsurf*() suivante décrite en pseudo-C :

```
void  Segment_Inter_Tsurf(segment,B,intersection)
{
   /* initialisations */
     • initialiser à 0 tous les champs stuff des triangles ;
     • poser :
                        intersection = ensemble vide ;
```

```
if( B-> nb_trgl == 0) /* noeud vide*/
   return ;

/* Si B est un noeud terminal, il faut calculer
    l'intersection du segment avec chaque triangle attaché à B */

Soit Θ la liste des triangles rangés dans B ;

if( B-> nb_trgl ≥ 0)
   pour-tout(triangle θ ∈ Θ)
   {
      p = intersection du segment avec θ;
      if(p ≠ φ)
       Insersion de p dans intersection;
   }

/* B est un noeud non terminal */
else if(segment intersecte la boîte B)
{
   /* Calcul de l'intersection du segment avec les triangles */

   Recherche de huit noeuds fils B[0] B[1] B[2] B[3] B[4] B[5] B[6] B[7];
   for(i=0;i< 8;i++)
    Segment_Inter_Tsurf(segment,B[i],intersection) ;
}

   return ;
}
```

Remarques

- La détermination de l'intersection d'un triangle ou d'un cube avec une T-surface, peut être obtenue à l'aide d'un algorithme analogue à celui présenté ci-dessus.
- l'implémentation de l'algorithme d'intersection d'un segment avec une T-surface requière la recherche de l'intersection de ce segment avec un triangle. La solution de ce problème sera présentée au chapitre suivant.

Chapitre

Position relative d'objets élémentaires 3D

3.1 Position du problème

Il existe de nombreux problèmes où l'on a besoin de déterminer la position relative de certains objets de l'espace 3D. Par exemple, certaines opérations sur les T-surfaces, qui seront présentées dans les chapitres ultérieurs, nécessitent la détermination de :

- la position d'un point par rapport à un polygone plan,
- la position d'une droite par rapport à un triangle,
- la position d'un triangle par rapport à un parallélépipède
- ...

Afin de ne pas alourdir inutilement l'exposé des méthodes qui seront pésentées dans les chapitres ultérieurs, nous avons rassemblé dans ce chapitre l'exposé des quelques méthodes de positionnement dont nous aurons besoin ultérieurement.

3.2 Position relative d'un point

3.2.1 Appartenance d'un point à un triangle 2D

Soit T un triangle du plan (ox, oy) dont les trois sommets sont notés $p_0(x_0, y_0)$, $p_1(x_1, y_1)$ et $p_2(x_2, y_2)$ et soit $p(x, y)$ un point du plan (ox, oy). Pour déterminer la position de $p(x, y)$ par rapport à T, on peut utiliser l'un des deux algorithmes suivants :

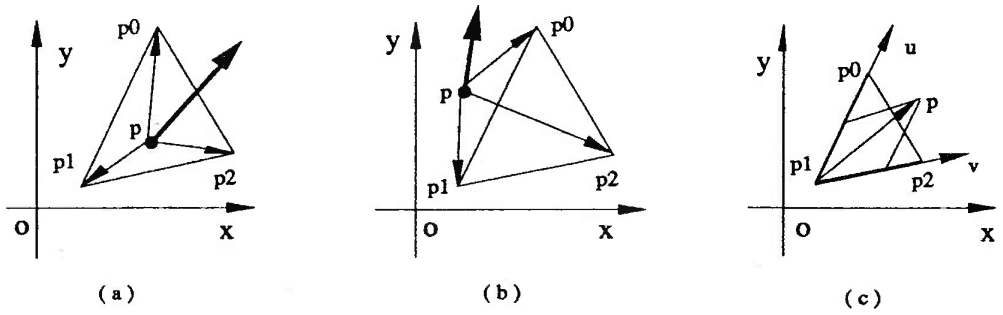


Figure 3.1 *La position relative d'un point p par rapport à un triangle T.*

algorithme 1

On calcule tout d'abord les 3 produits vectoriels suivantes:

$$\left|\begin{array}{lcl} \vec{V_1} & = & p\vec{p}_0 \wedge p\vec{p}_1 \\ \vec{V_2} & = & p\vec{p}_1 \wedge p\vec{p}_2 \\ \vec{V_3} & = & p\vec{p}_2 \wedge p\vec{p}_3 \end{array}\right.$$

On utilise ensuite les règles de décision suivante :

- $p \in T$, si $\vec{V_1}$, $\vec{V_2}$ et $\vec{V_3}$ ont la même direction (cf Fig 3.1.(a)).
- $p \in$ un côté de T, si l'un des vecteurs $\vec{V_1}, \vec{V_2}$ ou $\vec{V_3}$ est nul.
- p est l'un des points p_0, p_1, p_2 si deux des vecteurs $\vec{V_1}$, $\vec{V_2}$, $\vec{V_3}$ sont nuls.
- $p \notin T$, dans tous les autres cas (cf Fig 3.1.(b)).

algorithme 2

Supposons que l'aire de T ne soit pas nulle et, comme le suggère la figure 3.1(c), considérons le repère d'origine p_0 et de vecteurs de base $\vec{U}$ et $\vec{V}$ tels que :

$$\left|\begin{array}{lcl} \vec{U} & = & p_0\vec{p}_1 \\ \vec{V} & = & p_0\vec{p}_2 \end{array}\right.$$

Les coordonnées (α, β) du point p dans ce repère son alors telles que :

$$p_0\vec{p} = \alpha \cdot \vec{U} + \beta \cdot \vec{V} \quad (1)$$

α et β peuvent être obtenus de la façons suivante :
Compte tenu de (1), on peut ecrire :

$$\begin{array}{l} (x_1 - x_0)\alpha + (x_2 - x_0)\beta = x - x_0 \\ (y_1 - y_0)\alpha + (y_2 - y_0)\beta = y - y_0 \end{array}$$

$$\text{on pose : } D = \begin{vmatrix} (x_1 - x_0) & (x_2 - x_0) \\ (y_1 - y_0) & (y_2 - y_0) \end{vmatrix}$$

Puisque l'aire de T est supposée non nulle et que D est égal à 2 fois l'aire de T, on est assuré que $D \neq 0$ et l'on a :

$$\alpha = \frac{\begin{vmatrix} (x - x_0) & (x_2 - x_0) \\ (y - y_0) & (y_2 - y_0) \end{vmatrix}}{D}$$

$$\beta = \frac{\begin{vmatrix} (x_1 - x_0) & (x - x_0) \\ (y_1 - y_0) & (y - y_0) \end{vmatrix}}{D}$$

Il est évident que :

- $p \in T$ si :

$$\left\| \begin{array}{ll} \alpha > 0 & et \\ \beta < 0 & et \\ \alpha + \beta < 1 & \end{array} \right.$$

- $p \in$ un côté de T.
  - si $\alpha = 0$ et $0 < \beta < 1$, alors $p \in p_0p_2$;
  - si $\beta = 0$ et $0 < \alpha < 1$, alors $p \in p_0p_1$;
  - si $\alpha, \beta > 0$ et $\alpha + \beta = 1$, alors $p \in p_1p_2$;
- p est l'un des points p_0, p_1 *ou* p_2.
  - si $\alpha = 0$ et $\beta = 0$, alors p $= p_0$;
  - si $\alpha = 0$ et $\beta = 1$, alors p $= p_2$;
  - si $\alpha = 1$ et $\beta = 0$, alors p $= p_1$;
- p $\notin T$, dans tous les autres cas.

Comparaison des deux algorithmes

On peut remarquer que le coût du calcul de 3 vecteurs $\vec{V_1}$, $\vec{V_2}$, $\vec{V_3}$ est identique à celui du calcul de α et β. Une fois α calculé, si $\alpha > 0$, on peut conclure que $p \notin T$, l'algorithme 2 nécessite donc en moyenne une fois moins de calcul que l'algorithme 1.

3.2.2 Appartenance d'un point à un polygone simple

La méthode décrite ci-dessous pour tester l'appartenance d'un point à un polygone a été présentée par ***Preparata*** et ***Shamos*** (voir [22]). Comme l'indique la figure 3.2(a), considérons une droite horizontale l passant par le point z, dont on veut tester l'appartenance au polygone P du plan (ox, oy).

Supposons tout d'abord que l ne passe pas strictement par aucun sommet de P; deux cas peuvent se présenter :

1. Si l n'intersecte pas P, alors z est à l'extérieur de P
2. Si l ne passe par aucun sommet de P et si L est le nombre d'intersections de l avec P, alors :
   - si L est impair, alors z est à l'extérieur de P
   - si L est pair, alors z est à l'intérieur de P
   - on peut remarquer que si une intersection de l avec P est confondue avec z, alors z est sur la frontière de P

Considérons maintenant le cas particulier ou l passe par un sommet de P. Il est évident qu'une petite rotation de l autour de z ne modifiera pas la classification de z (extérieur ou intérieur de P) et cette remarque va nous permettre d'enlever l'ambiguité. Pour ce faire, considérons les deux côtés $p_{i-1}p_i$ et p_ip_{i+1} consécutifs de P adjacents au sommet p_i par lequel passe la droite l; comme l'indique la Figure 3.2, deux cas peuvent se présenter :

- p_{i+1} et p_{i-1} ont des ordonnées supérieures (respectivement inférieures) à celle de p_i. Dans ce cas, une petite rotation de l autour de z évite que l passe par p_i : ou bien l n'intersecte aucun des segments $p_{i+1}p_i$ et p_ip_{i-1}, ou bien l intersecte les deux. Donc, le nombre d'intersections est égal soit à 0 soit à 2 .
- p_{i-1} et p_{i+1} se trouvent de part et d'autre de l et une petite rotation de l autour de z permet d'éviter que l passe par p_i mais l'intersection de l avec $p_{i-1}p_i$ ou p_ip_{i+1} continue d'exister. Ceci prouve que dans ce cas, le nombre d'intersections doit être pris égal à 1.

En fait, ce sont les ordonnées de p_{i-1} et de p_{i+1} qui jouent le rôle essentiel: le côté p_ip_{i+1} dont le sommet p_i appartient à l doit être pris en compte si p_{i+1} a une ordonnée plus grande que celle de p_{i-1} ,sinon, il sera ignoré (le côté ayant deux extrémités appartenant à l est également ignoré).

3.2.3 Position relative d'un point par rapport à un parallélépipéde

Soient $p(x, y, z)$ un point de l'espace 3D dont on veut déterminer la position par rapport à un parallélépipéde rectangle B ayant ses arêtes parallèles aux axes de coordonnées et défini par :

- les coordonnées { ***x_min***, ***y_min***, ***z_min*** } de son sommet le plus proche de l'origine des axes de coordonnées

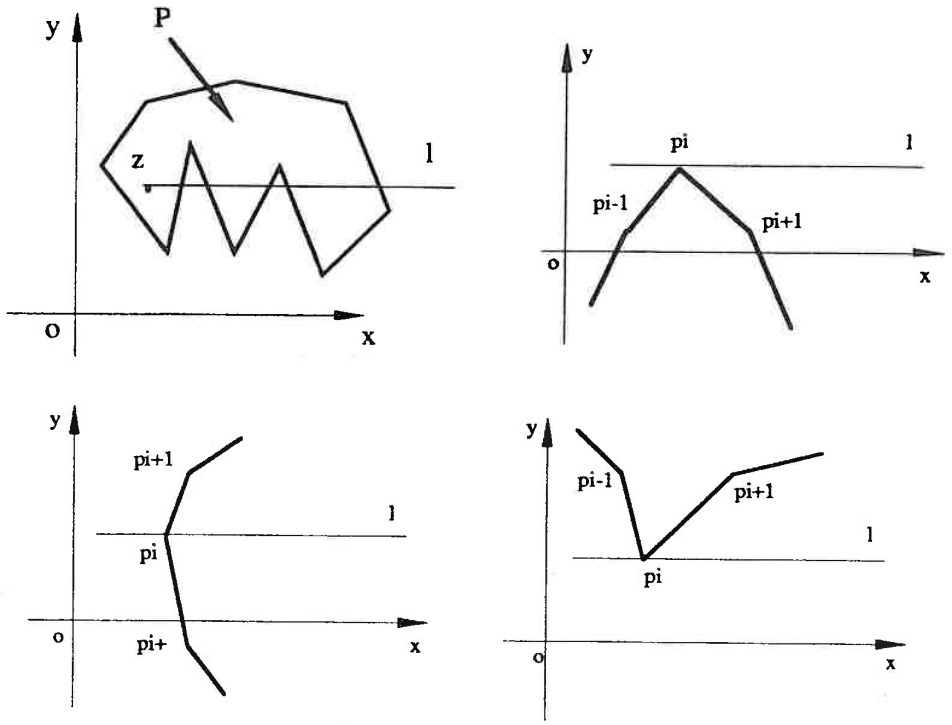


Figure 3.2 *L'appartenance d'un point z à un polygone simple P.*

- les dimensions { ***scale_x***, ***scale_y***, ***scale_z*** } de ses arêtes suivant les 3 directions de l'espace.

La méthode de positionnement d'un point par rapport à un parallélépipède est due à ***Cohen*** et ***Sutherland*** (voir [8]). Le principe de la méthode est basé sur une technique de codage binaire. Chaque bit de ce code caractérise de façon unique l'une des six positions possibles de p par rapport à B: en haut, en bas, à gauche, à droite, devant ou derrière. La signification de ces bits dans le code en fonction de la position de p par rapport à B est la suivante :

bit 1 = 1 $\Rightarrow$ en haut
bit 2 = 1 $\Rightarrow$ en bas
bit 3 = 1 $\Rightarrow$ à gauche
bit 4 = 1 $\Rightarrow$ à droite
bit 5 = 1 $\Rightarrow$ devant
bit 6 = 1 $\Rightarrow$ derrière

Si tous les bits sont 0, alors p est à l'intérieur de B.

3.3 Position relative d'un segment

Intersection de deux segments du plan (xoy)

Soient $E_1(p_1,p_2)$ et $E_2(p_3,p_4)$ deux segments du plan (xoy) dont les extrémités sont respectivement constituées par les points $\{p_1(x_1,y_1), p_2(x_2,y_2)\}$ et $\{p_3(x_3,y_3), p_4(x_4,y_4)\}$. Supposons que les équations des deux droites supportant ces deux segments soient respectivement :

$$\left| \begin{array}{rcl} A_1x + B_1y + C_1 = 0 \\ A_2x + B_2y + C_2 = 0 \end{array} \right.$$

Pour tout point $q(x,y)$ du plan (xoy), on appelle "puissances de q par rapport à $E_1(p_1,p_2)$ et $E_2(p_3,p_4)$" et l'on note respectivement $pow_1(q)$ et $pow_2(q)$ les nombres réels définis par :

$$\left| \begin{array}{rcl} pow_1(q) = A_1x + B_1y + C_1 = 0 \\ pow_2(q) = A_2x + B_2y + C_2 = 0 \end{array} \right.$$

Ces deux nombres nous permettent de déterminer l'intersection de $E_1(p_1,p_2)$ et $E_2(p_3,p_4)$ de la façon suivante (voir [8]) :

- si $pow_1(p_3) \cdot pow_1(p_4) > 0$ ou $pow_2(p_1) \cdot pow_2(p_2) > 0$, alors E_1 n'intersecte pas E_2.
- si $pow_1(p_3) = 0$, alors $E_1 \cap E_2 = p3$ (idem pour p_1, p_2, p_4).
- dans tous les autres cas, E_1 intersecte E_2 et les coordonnées $(x^\star,y^\star)$ du point d'intersection peuvent être déterminées de la façon suivante :

$$\left| \begin{array}{l} \left\{ \begin{array}{rcl} x^\star = \frac{x_1+\lambda x_2}{1+\lambda} \\ y^\star = \frac{y_1+\lambda y_2}{1+\lambda} \end{array} \right. \\ \text{avec : } \lambda = \frac{pow_2(p_1)}{pow_2(p_2)} \end{array} \right.$$

Test d'intersection d'un segment avec un rectangle du plan (xoy)

Considérons un segment $E(a,b)$ et un rectangle R situés dans le plan (xoy) et dont on se propose de tester l'intersection. On suppose que ce rectangle a ses côtés parallèles aux axes de coordonnées et que :

- (x_{min},y_{min}) sont les coordonnées des sommets de ce rectangle les plus petites.
- $(scale\_x, scale\_y)$ sont les longueurs des côtés de ce rectangle dans les directions x et y.

Comme le suggère la Figure 3.3, plusieurs cas peuvent se présenter et l'existence d'une intersection peut être déterminée de la façon suivante où (x_a,y_a) et (x_b,y_b) sont supposés représenter les coordonnées de a et b :

- si, comme l'indique la Figure 3.3.a on est dans le cas où :

$$\begin{array}{llll} x_a, x_b & < & x_{min} & \text{ou} \\ y_a, y_b & < & y_{min} & \text{ou} \\ x_a, x_b & > & x_{min} + scale\_x & \text{ou} \\ y_a, y_b & > & y_{min} + scale\_y & \end{array}$$

  alors le segment E n'intersecte pas le rectangle R.

- si, comme l'indique la Figure 3.3.b on est dans le cas où :

$$\begin{array}{lllll} x_{min} < x_a < x_{min} + scale\_x & \text{et} \\ y_{min} < y_a < y_{min} + scale\_y & \\ \text{ou} & \\ x_{min} < x_b < x_{min} + scale\_x & \text{et} \\ y_{min} < y_b < y_{min} + scale\_y & \end{array}$$

  alors le segment E intersecte le rectangle R.

- si l'on ne se trouve pas dans l'un des deux cas triviaux précédents, alors il est nécessaire de calculer l'équation du segment :

$$Ax + By + C = 0$$

$$\text{avec : } \begin{cases} A = y_a - y_b \\ B = x_b - x_a \\ C = x_b \cdot y_a - x_a \cdot y_b \end{cases}$$

  si la puissance des 4 sommets du rectangle R par rapport au segment E est toujours de même signe, alors, comme le suggère la Figure 3.3.c, le segment n'intersecte pas le rectangle.

- dans tous les autres cas le segment E intersecte le rectangle R (voir [22]).

Test d'intériorité d'une diagonale de polygone

Soient P un polygone simple[1] ayant n sommets notés $\{p_0, p_1, \ldots, p_n\}$. Etant donnée une diagonale $E(p_\alpha, p_\beta)$ de ce polygone, nous nous proposons de déterminer si $E(p_\alpha, p_\beta)$ est entièrement située à l'intérieur de P ou non.

Afin de simplifier l'exposé, considérons tout d'abord le cas particulier où $\alpha = i-1$ et $\beta = i+1$ et posons :

$$\begin{array}{lcl} E(p_{i-1}, p_{i+1}) & = & \text{diagonale à tester} \\ \theta(p_{i-1}, p_i, p_{i+1}) & = & \text{angle interne associé à } E(p_{i-1}, p_{i+1}) \end{array}$$

Si $\theta(p_{i-1}, p_i, p_{i+1}) > 180°$ alors $E(p_{i-1}, p_{i+1})$ est évidemment à l'extérieur de P mais, si ce n'est pas le cas, il est nécessaire de tester la position relative de chaque coté $E(p_j, p_{j+1})$ de P par rapport à la diagonale $E(p_{i-1}, p_{i+1})$; en pratique, trois cas peuvent se présenter :

[1] Voir page 17 la définition d'un polygone "simple".

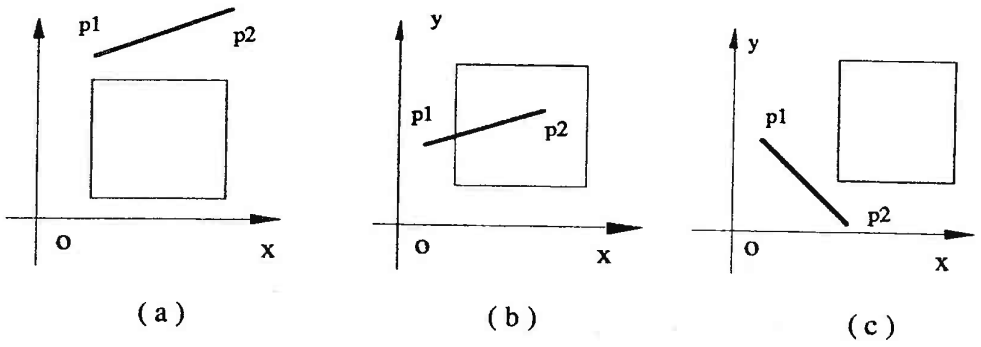


Figure 3.3 *Le test d'intersection d'un segment et d'un rectangle.*

1. $E(p_{i-1},p_{i+1})$ et $E(p_j,p_{j+1})$ se coupent.
2. une extrémité, p_j ou p_{j+1} du côté $E(p_j,p_{j+1})$ est sur la diagonale $E(p_{i-1},p_{i+1})$; (le cas où l'un des deux points p_j ou p_{j+1} coincide avec l'une des extrémités de $E(p_{i-1},p_{i+1})$ est supposé inclus).
3. les deux extrémités du côté $E(p_j,p_{j+1})$ sont sur la diagonale $E(p_{i-1},p_{i+1})$.

La position relative de la diagonale $E(p_{i-1},p_{i+1})$ par rapport au polygone P peut être déterminée en considérant les nombres N et M définis de la façon suivante :

$$\left|\begin{array}{lcll} N & = & & \{\text{nombre de cas de type 1}\} \\ M & = & & \{\text{nombre de cas de type 2}\} \\ & + & 2\times & \{\text{nombre de cas de type 3}\} \end{array}\right.$$

On utilise alors la règle de décision suivante :

- si $N > 0$, alors $E(p_{i-1},p_{i+1}) \notin P$ (cf Fig 3.4(b)).
- si $N = 0$, alors on doit considérer deux cas :
  - si $M = 4$, alors il est nécessaire de tester si le sommet p_{i+2} appartient au triangle $T(p_{i-1},p_i,p_{i+1})$:

$$p_{i+2} \in T(p_{i-1},p_i,p_{i+1}) \Longrightarrow E(p_{i-1},p_i+1) \notin P$$
$$p_{i+2} \notin T(p_{i-1},p_i,p_{i+1}) \Longrightarrow E(p_{i-1},p_i+1) \in P$$

  - si $M > 4$, alors on doit tester les positions relatives de tous les sommets de P (exceptés p_{i-1}, p_i et p_{i+1}) par rapport au triangle $T(p_{i-1},p_i,p_{i+1})$. Si, comme le suggère la Figure 3.4(c), l'un de ces sommets est situé à l'intérieur de $T(p_{i-1},p_i,p_{i+1})$, alors $E(p_{i-1},p_i+1) \notin P$, sinon $E(p_{i-1},p_i+1) \in P$.

Dans le cas général où $|\alpha-\beta| > 1$, on peut s'inspirer de ce qui précède pour déterminer de façon analogue la position relative d'une diagonale $E(\alpha,\beta)$ par rapport au polygone P.

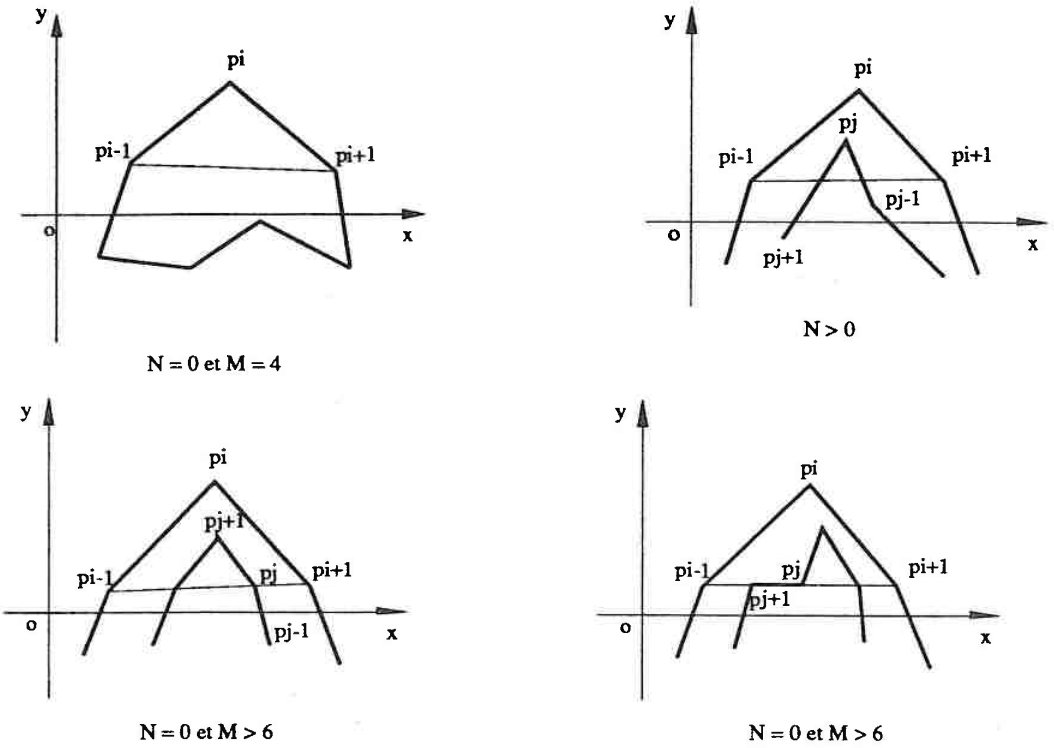


Figure 3.4 *Test d'intériorité d'une diagonale du polygone P.*

Intersection d'un segment avec une facette triangulaire

Nous nous proposons de déterminer l'intersection entre un segment $E(a,b)$ et un triangle $T(p_0,p_1,p_2)$ definis respectivement par :

- les extrémités $a(x_a,y_a,z_a)$ et $b(x_b,y_b,z_b)$
- les trois sommets $p_0(x_0,y_0,z_0)$, $p_1(x_1,y_1,z_1)$ et $p_2(x_2,y_2,z_2)$

Remarquons tout d'abord que l'équation du plan P contenant $T(p_0,p_1,p_2)$ peut s'exprimer sous la forme suivante :

$$Ax + By + Cz + D = 0$$

avec :

$$A = \begin{vmatrix} y_0 & z_0 & 1 \\ y_1 & z_1 & 1 \\ y_2 & z_2 & 1 \end{vmatrix} \quad B = -\begin{vmatrix} x_0 & z_0 & 1 \\ x_1 & z_1 & 1 \\ x_2 & z_2 & 1 \end{vmatrix}$$

$$C = \begin{vmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} \quad D = - \begin{vmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix}$$

La puissance $pow(p)$ d'un point p quelconque de coordonnées (x, y, z) par rapport au plan P est alors définie par :

$$pow(p) = Ax + By + Cz + D$$

On peut alors tester l'intersection du segment $E(a,b)$ avec le triangle $T(p_0,p_1,p_2)$ en utilisant la règle de décision suivante :

- si les puissances des deux points a et b par rapport au plan P sont de même signe, alors le segment n'intersecte pas la facette.
- si la puissance de a (resp. b) par rapport à P est nulle, alors l'intersection du segment avec la facette est a (resp. b) si a (resp. b) est à l'intérieur de la facette.
- si l'on n'est pas dans l'un des deux cas précédents, il est nécessaire de calculer l'intersection c du segment $E(a,b)$ avec le plan P. Pour ce faire, comme l'indique la Figure 3.5, considérons les distances d_a et d_b de a et b au plan P :

$$d_a = \frac{|A \cdot x_a + B \cdot y_a + C \cdot z_a + D|}{\sqrt{(A^2+B^2+C^2)}}$$

$$d_b = \frac{|A \cdot x_b + B \cdot y_b + C \cdot z_b + D|}{\sqrt{(A^2+B^2+C^2)}}$$

Les coordonnées (x_c, y_c, z_c) du point c sont alors définies par :

$$\left| \begin{array}{l} \left\{ \begin{array}{lcl} x_c & = & \frac{x_a + \lambda \cdot x_b}{1+\lambda} \\ y_c & = & \frac{y_a + \lambda \cdot y_b}{1+\lambda} \\ z_c & = & \frac{z_a + \lambda \cdot z_b}{1+\lambda} \end{array} \right. \\ \text{avec : } \lambda = \frac{d_a}{d_b} = \frac{pow(a)}{pow(b)} \end{array} \right.$$

Si c est à l'intérieur[2] de $T(p_0,p_1,p_2)$, alors l'intersection du segment $E(a,b)$ avec la facette est c, sinon il n'y a pas d'intersection.

Test d'intersection d'un segment avec un parallélépipède

Dans ce qui suit, nous nous proposons de déterminer la position relative d'un segment $E(a,b)$ par rapport à un parallélépipède rectangle ayant ses côtés parallèles aux axes de coordonnées. Nous supposerons que :

- $E(a,b)$ est défini par ses extrémités a et b de coordonnées (x_a, y_a, z_a) et (x_b, y_b, z_b).

[2] Nous avons déjà proposé au paragraphe 3.2.1 une méthode permettant de tester si un point est à l'intérieur d'une facette triangulaire.

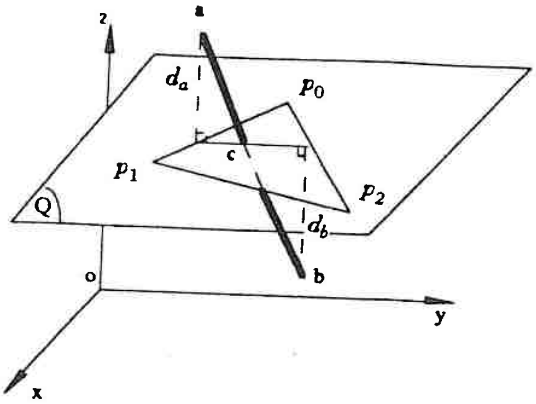


Figure 3.5 *Détermination de l'intersection d'un segment ab avec un triangle T.*

- le parallélépipède est défini par les coordonnées minimales $(x_{min}, y_{min}, z_{min})$ de ses sommets et par les tailles $(scale\_x, scale\_y, scale\_z)$ de ses cotés dans les trois directions de l'espace.

Dans ce qui suit, nous nous proposons de déterminer la position relative de ce segment et de ce parallélépipède; pour ce faire, nous proposons de procéder de la manière suivante :

- si l'une des conditions suivantes est vérifiée :

$$\begin{array}{lcll} x_a, x_b & < & x_{min} & \text{ou} \\ y_a, y_b & < & y_{min} & \text{ou} \\ z_a, z_b & < & z_{min} & \text{ou} \\ x_a, x_b & > & x_{min} + scale\_x & \text{ou} \\ y_a, y_b & > & y_{min} + scale\_y & \text{ou} \\ z_a, z_b & > & z_{min} + scale\_z & \end{array}$$

  alors le segment $E(a,b)$ n'intersecte pas le parallélépipède.

- de même, si l'une des conditions suivantes est vérifiée :

$$\begin{array}{lcl} x_{min} < x_a < x_{min} + scale\_x & \text{et} \\ y_{min} < y_a < y_{min} + scale\_y & \text{et} \\ z_{min} < z_a < z_{min} + scale\_z & \\ \text{ou} & \\ x_{min} < x_b < x_{min} + scale\_x & \text{et} \\ y_{min} < y_b < y_{min} + scale\_y & \text{et} \\ z_{min} < z_b < z_{min} + scale\_z & \end{array}$$

  alors le segment $E(a,b)$ intersecte le parallélépipéde .

- si l'on ne se trouve pas dans l'un des deux cas triviaux précédents, alors il est nécessaire de projetter le segment et le parallélépipède sur **chacun** des trois plans (xoy), (yoz) et (zox) :
  - si parmi ces trois plans il n'y en a aucun dans lequel la projection du parallélépipède[3] intersecte[4] la projection du segment[5] alors le segment intersecte le parallélépipède.
  - sinon, ils ne s'intersectent pas.

3.4 Position relative d'un triangle

Test d'intersection d'un triangle avec un rectangle de l'espace 2D

Dans ce qui suit, nous nous proposons de déterminer la position relative d'un triangle $T(p_0,p_1,p_2)$ par rapport à un rectangle ayant ses côtés parallèles aux axes de coordonnées. Nous supposerons que le triangle et le rectangle sont situés dans le paln (xoy) et que :

- $T(p_0,p_1,p_2)$ est défini par ses sommets (p_0,p_1,p_2) de coordonnées (x_0,y_0), (x_1,y_1) et (x_2,y_2).
- le rectangle est défini par les coordonnées minimales (x_{min},y_{min}) de ses sommets et par les tailles $(scale\_x, scale\_y)$ de ses côtés dans les deux directions de l'espace.

Pour déterminer si le triangle et le rectangle s'intersectent ou non, nous proposons de procéder de la facon suivante :

- si l'une des conditions suivantes est vraie:

$$\begin{array}{lcll} x_0,x_1,x_2 & < & x_{min} & \text{ou} \\ y_0,y_1,y_2 & < & y_{min} & \text{ou} \\ x_0,x_1,x_2 & > & x_{min}+scale\_x & \text{ou} \\ y_0,y_1,y_2 & > & y_{min}+scale\_y & \end{array}$$

  alors le triangle n'intersecte pas le rectangle.

- de même si:

$$\begin{array}{lcllcll} x_{min} & < & x_0 & < & x_{min}+scale\_x & & \text{et} \\ y_{min} & < & y_0 & < & y_{min}+scale\_y & & \\ & & & \text{ou} & & & \\ x_{min} & < & x_1 & < & x_{min}+scale\_x & & \text{et} \\ y_{min} & < & y_1 & < & y_{min}+scale\_y & & \\ & & & \text{ou} & & & \\ x_{min} & < & x_2 & < & x_{min}+scale\_x & & \text{et} \\ y_{min} & < & y_2 & < & y_{min}+scale\_y & & \end{array}$$

  alors le triangle intersecte le rectangle.

[3] Cette projection est alors un rectangle ayant ses côtés parallèles aux axes de coordonnées.
[4] Le test d'intersection d'un segment avec un rectangle a été présentée précédement au paragraphe 3.3.
[5] Cette projection peut être réduite à un point.

- si l'on ne se trouve pas dans l'un des deux cas triviaux précédents, alors il est nécessaire de déterminer si au moins un côté du triangle intersecte le rectangle[6]; si oui, alors le triangle et le rectangle s'intersectent.

  Si aucun des trois côtés du triangle n'intersecte le rectangle, alors on doit tester si le rectangle est contenu dans le triangle. Pour ce faire, il suffit de tester si un sommet du rectangle est contenu à l'intérieur du triangle[7]. Si le sommet n'est pas contenu dans le triangle, alors le rectangle n'intersecte pas le triangle, sinon ils s'intersectent.

- dans tous les autres cas, le triangle intersecte le rectangle.

Recherche de l'intersection de deux triangles de l'espace 3D

Soient T_1, T_2 deux triangles définis par leurs sommets et soient P_1 et P_2 les deux plans contenant respectivement T_1 et T_2. Posons que $E_0 = T_1 \cap T_2$, $E_1 = T_1 \cap P_2$ et $E_2 = T_2 \cap P_1$; il est évident que :

- si $E_1 \cap E_2 = \phi$, alors $E_0 = \phi$ (T_1 n'intersecte pas T_2).

- dans tous les autres cas, compte tenu que E_1 et E_2 sont colinéaires, on est assuré que E_0 est la partie commune de E_1 et de E_2. La détermination de cette partie commune peut être effectuée de la manière suivante :

  1. on teste tout d'abord la position relative de chacune des deux extrémités du segment E_1 par rapport au segment E_2 :
     - si les deux extrémités sont toutes à l'intérieur de E_2, alors $E_0 = E_1$.
     - si l'une des deux extrémités de E_1 est à l'intérieur de E_2, alors cette extrémité constitue une extrémité de E_0 (cette extrémité est dite "interne").
  2. de la même façon, on teste ensuite la position des deux extrémités de E_2 par rapport à E_1.

  Si dans les deux étapes ci-dessus on a extrait deux extrémités "internes", alors $E_0 = T_1 \cap T_2$ est constituée par le segment dont les deux extrémités sont ces deux points. Notons toutefois que si ces deux extrémités sont confondues, alors E_0 peut se réduire à un point.

Test d'intersection d'un triangle avec un parallélépipède

Considérons dans l'espace 3D un triangle $T(p_0, p_1, p_2)$ et un parallélépipède rectangle ayant ses côtés parallèles aux axes de coordonnées tels que :

- $T(p_0, p_1, p_2)$ est défini par ses trois sommets (p_0, p_1, p_2) de coordonnées (x_0, y_0, z_0), (x_1, y_1, z_1) et (x_2, y_2, z_2).

[6] Le test d'intersection d'un segment avec un rectangle a été présenté au paragraphe 3.3

[7] Le test de l'appartenance d'un point à un triangle au paragraphe 3.2.1

- le parallélépipède est défini par les coordonnées minimales $(x_{min}, y_{min}, z_{min})$ de ses sommets et par les tailles $(scale\_x, scale\_y, scale\_z)$ de ses côtés dans les trois directions de l'espace.

Dans ce qui suit , nous nous proposons de déterminer la position relative de ce triangle et de ce parallélépipède; pour ce faire, nous proposons de procéder de la manière suivante :

- il est évident que si l'une des conditions suivantes est vérifiée :

$$\begin{array}{lll} x_0, x_1, x_2 < x_{min} & \text{ou} \\ y_0, y_1, y_2 < y_{min} & \text{ou} \\ z_0, z_1, z_2 < z_{min} & \text{ou} \\ x_0, x_1, x_2 > x_{min} + scale\_x & \text{ou} \\ y_0, y_1, y_2 > y_{min} + scale\_y & \text{ou} \\ z_0, z_1, z_2 > z_{min} + scale\_z \end{array}$$

  alors le triangle n'intersecte pas le parallélépipède.

- de même si:

$$\begin{array}{lllll} x_{min} < x1 < x_{min} + scale\_x & \text{et} \\ y_{min} < y1 < y_{min} + scale\_y & \text{et} \\ z_{min} < z1 < z_{min} + scale\_z \\ \text{ou} \\ x_{min} < x2 < x_{min} + scale\_x & \text{et} \\ y_{min} < y2 < y_{min} + scale\_y & \text{et} \\ z_{min} < z2 < z_{min} + scale\_z \\ \text{ou} \\ x_{min} < x3 < x_{min} + scale\_x & \text{et} \\ y_{min} < y3 < y_{min} + scale\_y & \text{et} \\ z_{min} < z3 < z_{min} + scale\_z \end{array}$$

  alors le triangle intersecte le parallélépipède.

- si l'on ne se trouve pas dans l'un des deux cas triviaux précédents, alors il est nécessaire de calculer l' équation du plan contenant le triangle et la puissance de chaque sommet du parallélépipède par rapport à ce plan :

  - si la puissance des 8 sommets du parallélépipède est toujours de même signe, alors le parallélépipède n'intersecte pas le plan du triangle et par conséquent n'intersecte pas le triangle.
  - sinon on projette le triangle et le parallélépipède sur les trois plans (xoy), (yoz) et (zox). Si parmi ces trois projections il y en a au moins une telle que la projection du triangle[8] n'intersecte pas la projection du parallélépipède[9] alors le triangle n'intersecte pas le parallélépipède.

- dans tous les autres cas, le triangle intersecte le parallélépipède.

[8] Cette projection est aussi un triangle éventuellement dégénéré en un segment.
[9] Cette projection est un rectangle.

3.5 Position relative d'un polygone simple

Recherche de l'intersection de deux polygones simples en 3D

Considérons deux polygones simples sans trous R et Q situés dans des plans différents et définis par leurs sommets respectifs r_i et q_j :

$$\begin{array}{ll} R & \text{défini par :} \quad \{r_1, r_2, ..., r_m\} \\ Q & \text{défini par :} \quad \{q_1, q_2, ..., q_n\} \end{array}$$

Soit P_R et P_Q les plans contenant respectivement R et Q; Nous nous proposons de construire la suite des segments appartenant à $P_R \cap P_Q$ et constituant l'ensemble $P \cap Q$. Pour ce faire, nous proposons de procéder de la façon suivante :

- si les puissances des sommets de Q (resp. R) par rapport au plan P_R (resp. P_Q) sont de même signe, alors $R \cap Q = \phi$.
- si R et Q n'appartiennent pas au cas trivial ci-dessus, il est nécessaire de calculer effectivement l'intersection de $R \cap P_Q$ et $Q \cap P_R$ à l'aide de l'algorithme présenté au prochain paragraphe; posons :

$$\begin{array}{lcll} L(Q, P_R) & = & Q \cap P_R & = \{l_Q^1, l_Q^2, ..., l_Q^k\} \\ L(R, P_Q) & = & R \cap P_Q & = \{l_R^1, l_R^2, ..., l_R^h\} \\ L(P_R, P_Q) & = & P_R \cap P_Q & \end{array}$$

  Il est évident que $L(P_R, P_Q)$ est une droite tandis que $L(Q, P_R)$ et $L(R, P_Q)$ sont constituées respectivement de segments colineaires l_Q^i et l_R^j contenus dans $L(P_R, P_Q)$.

  On supposera que la droite $L(P_R, P_Q)$ est orientée et que les segments l_Q^i et l_R^j sont numérotés de façon croissante suivant la direction positive de $L(P_R, P_Q)$.

  En remarquant que

$$P \cap Q = L(P_R, Q) \cap L(P_Q, R)$$

  on en conclut que la résolution du problème consiste à calculer la partie commune de $L(P_R, Q)$ et $L(P_Q, R)$ (voir [34] [28]).

Détermination de l'intersection d'un polygone simple avec un plan

Ainsi que nous venons de le voir, l'algorithme précédent nécessite le calcul de l'intersection d'un polygone simple Q avec un plan P dans l'espace 3D. Il est évident que l'intersection recherchée est un ensemble de segments colinéaires dont les extrémités sont les intersections des arêtes de Q avec P. La recherche de $P \cap Q$ consiste donc à calculer les intersections des côtés de Q avec P et les joindre ensuite deux à deux de façon adéquate pour construire des segments :

- Supposons tout d'abord que P ne passe par aucun sommet de Q et soit $M = \{m_1, m_2, ..., m_l\}$ l'ensemble des points d'intersection des arêtes du polygone Q avec le plan P; on peut remarquer que le nombre d'éléments de M est pair ou égal à 0. Dans ce cas, les segments

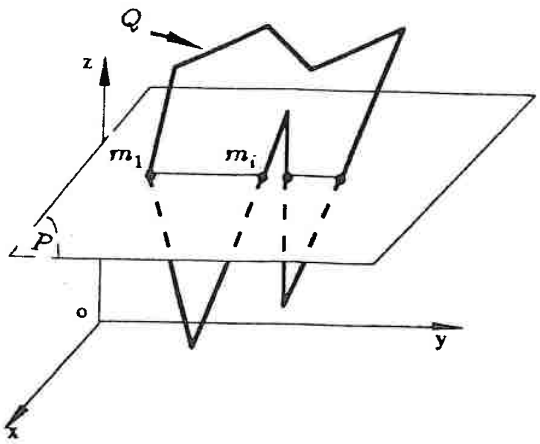


Figure 3.6 *Calcul de l'intersection d'un polygone simple Q avec un plan P.*

d'intersections de P avec Q peuvent être obtenus en joignant les points de M deux à deux de la facon suivante:

$$Q \cap P = \{m_1m_2, m_3m_4, \ldots, m_{2i+1}m_{2(i+1)}, \ldots, m_{2l-1}m_{2l}\}$$

Cette façon de procéder suppose que les points m_i sont numérotés en fonction de leur classement sur la droite intersection du plan P avec le plan P_Q contenant le polygone Q.

- considérons maintenant le cas particulier où P passe par un sommet q_i de Q. On peut alors appliquer le même algorithme que celui déjà exposé au paragraphe 3.2.2 pour déterminer la position d'un point par rapport à un polygone simple et l'on doit donc considérer deux cas :

  - si les deux sommets voisins q_{i-1} et q_{i+1} sont situés du même côté de P, alors les côtés adjacents $p_{i-1}p_i$ et p_ip_{i+1} doivent être ignorés.
  - si p_{i-1} et p_{i+1} se trouvent de part et d'autre de P, alors p_i doit être pris en compte et constitue alors l'un des points d'intersection m_j (voir [22]).

Test d'intersection d'un polygone simple avec un parallélépipède

Considérons dans l'espace 3D un polygone simple Q et un parallélépipède rectangle ayant ses côtés parallèles aux axes de coordonnées tels que :

- Q est défini par ses sommets $\{q_i\}$ de coordonnées (x_i, y_i, z_i).
- le parallélépipède est défini par les coordonnées minimales $(x_{min}, y_{min}, z_{min})$ de ses sommets et par les tailles $(scale\_x, scale\_y, scale\_z)$ de ses cotés dans les trois directions de l'espace.

Dans ce qui suit , nous nous proposons de déterminer la position relative de ce polygone et de ce parallélépipède; pour ce faire, nous suggerons de procéder de la manière suivante :

- si chaque sommet q_i de Q satisfait les conditions suivantes:

$$\begin{cases} x_i < x_{min} \\ y_i < y_{min} \\ z_i < z_{min} \end{cases} \quad \text{ou} \quad \begin{cases} x_i > x_{min} + scale\_x \\ y_i > y_{min} + scale\_y \\ z_i > z_{min} + scale\_z \end{cases}$$

  alors le polygone n'intersecte pas le parallélépipède.

- si parmi les sommets de Q, il en existe au moins un, disons q_i, tel que:

$$\begin{array}{lllll} x_{min} < x_i < x_{min} + scale\_x & \text{et} \\ y_{min} < y_i < y_{min} + scale\_y & \text{et} \\ z_{min} < z_i < z_{min} + scale\_z \end{array}$$

  alors le polygone intersecte le parallélépipède.

- si l'on ne se trouve pas dans l'un des deux cas triviaux précédents, alors on calcule l'équation du plan P_Q contenant le polygone et on teste les puissances des huit sommets du parallélépipède par rapport au plan P_Q. En désignant par $\{\varpi_1, \varpi_2, \ldots, \varpi_8\}$ ces huit puissances, il est évident que :

  - si toutes les valeurs ϖ_i sont de même signe, alors le polygone Q n'intersecte pas le plan P.
  - sinon on projette le polygone et le parallélépipède sur les trois plans (xoy), (yoz) et (zox). Si parmi ces trois projections il y en a au moins une telle que la projection du polygone simple[10] n'intersecte pas la projection du parallélépipède[11] alors le polygone Q n'intersecte pas le parallélépipède.

- dans tous les autres cas, le polygone Q intersecte le parallélépipède (voir [8]).

[10] Cette projection est aussi un polygone simple éventuellement dégénéré en un segment.
[11] Cette projection est un rectangle.

Partie II

Manipulation des T-surfaces

Chapitre

4

Opérations booléennes sur les T-solides

4.1 Introduction

La mise en oeuvre des opérations booléennes (intersection, union, différence etc..) sur des solides définis par leur surface extérieure a toujours été un problème délicat. Intuitivement, le problème est simple, mais, jusqu'à présent, il y a peu de travaux consacrés à sa résolution pratique. De plus, comme le fait remarquer P.Martin (voir [20]), les articles consacrés à ce sujet présentent des algorithmes contenant des erreurs graves[1], et interdisent la modélisation d'objets non convexes et celle d'objets ayant des trous à l'intérieur des facettes définissant leur surface extérieure (voir Figure 4.1). Indépendamment des erreurs, ces conditions sur les objets sont inacceptables pour les applications pratiques car, à partir d'objets convexes sans trous, les opérations booléenes peuvent engendrer des objets n'ayant pas de telles caractéristiques. On perdrait donc la stabilité de la classe d'objets pour ces opérations, ce qui interdirait d'appliquer celle-ci plusieurs fois consécutivement.

Généralement, les auteurs (voir [19],[20],[3]) qui se sont intéressés au problème, considèrent sa résolution pratique comme étant extrèmement complexe, et se limitent à la présentation de jeux d'essais plus ou moins formels ne prenant en considération que quelques cas particuliers[2]; les cas de figure qui ne sont pas traités correctement sont souvent ceux dont la résolution est la plus coûteuse en temps de calcul. Il s'en suit que, si l'on souhaite vraiment traiter correctement les nombreuses situations pouvant réellement se présenter en pratique, alors de tels algorithmes deviennent inutilisables (voir [29]).

Dans ce chapitre, nous proposons une méthode générale permettant de résoudre le problème posé dans le cas où les objets considérés sont des T-solides, c'est-à-dire, des solides dont la peau est constituée de facettes triangulaires planes (voir page 12). En fait, la méthode présentée

[1] Par exemple l'erreur du "test de joignabilité" cité par P.MARTIN (voir [20]).

[2] je m'excuse pour les disjugements potentiels

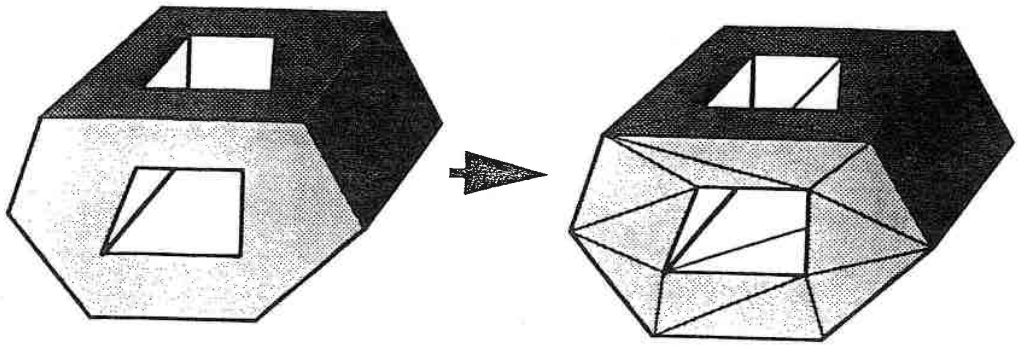


Figure 4.1 *Conversion d'un P-solide en T-solide.*

s'applique plus généralement à tout P-solide dont la peau est constituée de facettes polygonales planes (voir page 17) car nous démontrerons (voir page 75) qu'il est toujours possible de décomposer une facette polygonale en facettes triangulaires (voir Figure 4.1).

Soient V_1 et V_2 deux T-solides (cf chapitre 1.1) dont les peaux seront respectivement notées $S(V_1)$ et $S(V_2)$. Dans ce qui suit nous nous proposons de construire la peau des T-solides déduit de V_1 et V_2 par application d'opérateurs booléens binaires :

$$\left|\begin{array}{lcl} S(V_1 \cap V_2) & \equiv & \text{peau de l'intersection } (V_1 \cap V_2) \\ S(V_1 \cup V_2) & \equiv & \text{peau de l'union } (V_1 \cup V_2) \\ S(V_1 - V_2) & \equiv & \text{peau de la différence } (V_1 - V_2) \\ S(V_1 \Delta V_2) & \equiv & \text{peau de la différence symétrique } \{(V_1 - V_2) \cup (V_2 - V_1)\} \end{array}\right.$$

4.2 Algorithme et sa mise en oeuvre

Algorithme général

Nous supposerons que les deux conditions suivantes sont vérifiées :

$$\left|\begin{array}{l} S(V_1 \cap V_2) \neq \emptyset \\ \\ S(V_1) \cap S(V_2) = \text{ ensemble fini de lignes} \end{array}\right.$$

La première condition ci-dessus permet d'éliminer le cas où le problème posé est trivial, tandis que la deuxième permet d'eliminer le cas où les deux surfaces sont tangentes sur des portions de facettes d'aire non nulle; dans les applications pratiques, la deuxième condition n'est pas trés contraignante et peut toujours être satisfaite en déplaçant très légèrement certains sommets de triangles si nécessaire.
Sous ces hypothèses, le principe de la méthode proposée repose sur les trois faits suivants :

- Les T-solides V_1 et V_2 se coupent suivant une ligne polygonale $S(V_1) \cap S(V_2)$.
- Les sommets de cette ligne polygonale $S(V_1) \cap S(V_2)$ correspondent toujours à l'intersection d'un côté d'une facette triangulaire de $S(V_1)$ (ou $S(V_2)$) avec une facette de $S(V_2)$ (ou $S(V_1)$).

- Chaque côté de cette ligne polygonale $S(V_1) \cap S(V_2)$ se situe toujours à la fois à l'intérieur d'une facette de $S(V_1)$ et à l'intérieur d'une facette de $S(V_2)$.

Pour tout couple de T-surfaces (S_1, S_2), posons :

$$L(S_1, S_2) = S_1 \cap S_2$$

Compte tenu des trois faits présentés ci-dessus, notre algorithme peut être décomposé en 3 étapes successives :

1. Pour chaque facette T_1 de $S(V_1)$, on détermine le morceau (sommets et segments) de la courbe polygonale $L(S(V_1), S(V_2))$ situé à l'intérieur de T_1. En fait, ce morceau n'est autre que $L(T_1, S(V2))$. De la même façon, on détermine l'intersection de chaque facette de $S(V_2)$ avec $S(V_1)$.

2. On découpe chaque facette T_1 de $S(V_1)$ en sous-facettes triangulaires suivant la ligne polygonale $L(T_1, S(V_2))$ obtenue à l'étape précédente (si $L(T_1, S(V_2))$ existe). De la même façon, on découpe chaque facette T_2 de $S(V_2)$ suivant la ligne polygonale $L(T_2, S(V_1))$.

3. Une fois l'étape 2 réalisée, on sépare chacune des T-surfaces $S(V_1)$ et $S(V_2)$ en deux parties $S_{in}(V_1/V_2)$ et $S_{out}(V_1/V_2)$:

$$\left|\begin{array}{lcl} S_{in}(V_1/V_2) & \equiv & \text{partie de } S(V_1) \text{ située à l'intérieur de } S(V_2) \\ S_{out}(V_1/V_2) & \equiv & \text{partie de } S(V_1) \text{ située à l'extérieur de } S(V_2) \\ S_{in}(V_2/V_1) & \equiv & \text{partie de } S(V_2) \text{ située à l'intérieur de } S(V_1) \\ S_{out}(V_2/V_1) & \equiv & \text{partie de } S(V_2) \text{ située à l'extérieur de } S(V_1) \end{array}\right.$$

   On obtient ainsi quatre morceaux de T-surfaces ouvertes (cf figure 4.2) qu'il s'agit de recoller deux à deux le long de la ligne polygonale d'intersection $L(S(V_1), S(V_2))$ afin d'obtenir les T-surfaces fermées $S(V_1 \cap V_2)$, $S(V_1 \cup V_2)$ ou $S(V_1 - V_2)$, etc...

A travers ces trois étapes, les opérations booléennes génèrent des solides de même nature, c'est-à-dire, des solides répondant à notre définition du T-solide donnée au chapitre 1.1. Cette cohérence est nécessaire pour nous assurer que l'on pourra appliquer plusieurs fois de suite et combiner ces opérations afin de réaliser des opérations plus complexes.

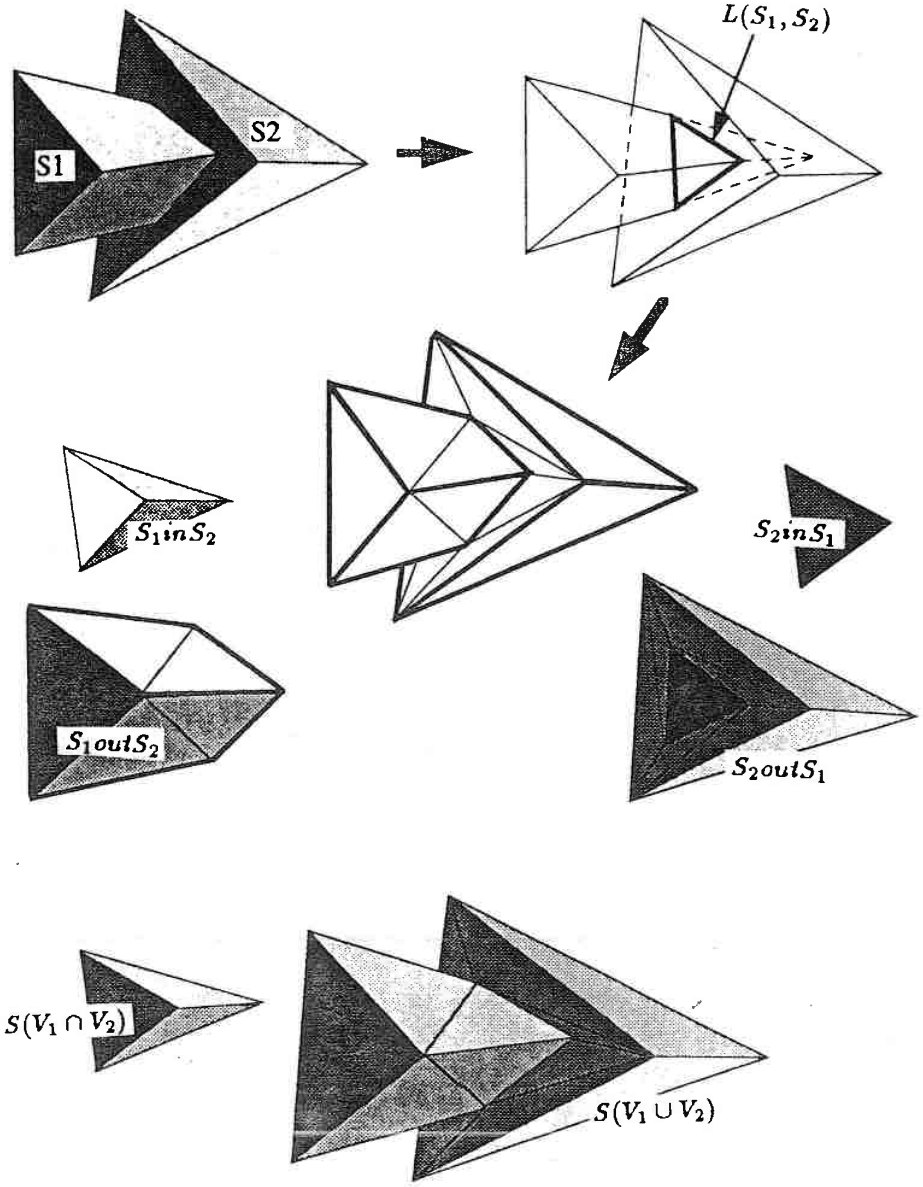


Figure 4.2 *Exemples d'opérations booléennes dans le cas où V_1 et V_2 sont deux tétraèdres.*

4.3 Détermination de l'intersection de chaque triangle de $S(V_1)$ avec $S(V_2)$

Algorithme simple

En désignant par T_1 un triangle quelconque de $S(V_1)$, et T_2 un triangle quelconque de $S(V_2)$, on peut écrire :

$$L(T_1, S(V_2)) \quad = \quad \bigcup_{T_2 \in S(V_2)} T_1 \cap T_2$$

De cette relation, on peut déduire que la détermination de $L(T_1, S(V_2))$ consiste à calculer l'intersection de T_1 avec chaque triangle de $S(V_2)$ à l'aide de l'algorithme[3] précédemment présenté au chapitre 1.2. En utilisant cet algorithme pour chaque triangle de $S(V_1)$ et chaque triangle de $S(V_2)$, le coût est de l'ordre de $O(n_1 + n_2)$ où n_1 et n_2 sont respectivement le nombre de triangles de $S(V_1)$ et de $S(V_2)$. En général, ce coût n'est pas du tout élevé et de plus l'algorithme est facile à mettre en oeuvre.

Algorithme proposé

Reprenons les symboles définis précédemment et soit $L(T_1, T_2)$ l'intersection de T_1 avec T_2. L'idée de l'algorithme proposé est basée sur les faits suivants :

$$\left| \begin{array}{l} L(T_1, T_2) = L(T_2, T_1) \\ L(T_1, T_2) \in L(T_2, S(V_1)) \\ L(T_1, T_2) \in L(T_1, S(V_2)) \end{array} \right.$$

Pour un triangle T_1 de $S(V_1)$, le calcul de l'intersection $L(T_1, S(V_2))$ de T_1 avec $S(V_2)$ consiste à calculer les intersections $L(T_1, T_2)$ de T_1 avec chaque triangle T_2 de $S(V_2)$. Il est intéressant d'attacher $L(T_1, T_2)$ avec T_2 afin d'éviter le calcul ultérieur de $L(T_2, T_1) \equiv L(T_1, T_2)$.

En procédant ainsi, il est possible d'obtenir à la fois l'intersection de chaque triangle de $S(V_1)$ avec $S(V_2)$ et l'intersection de chaque triangle de $S(V_2)$ avec $S(V_1)$ en ne calculant que le premier type d'intersection. Ceci économise à peu près la moitié des calculs par rapport à l'algorithme simple présenté ci-dessus; de plus cela permet d'éviter la construction du T-octree relatif à $S(V_1)$ normalement utilisé par l'algorithme "simple" présenté ci-dessus pour le calcul de $L(T_2, S(V_1))$.

Nous présentons ci-dessous une version écrite en pseudo-C de l'algorithme proposé ayant pour effet de :

- calculer les intersections $L(T_1, S(V_2))$ et les attacher à T_1 pour chaque triangle T_1 de $S(V_1)$
- calculer les intersections $L(T_2, S(V_1))$ et les attacher à T_2 pour chaque triangle T_2 de $S(V_2)$

```
void Computing_Intersection_Segments(S(V1), S(V2))
{
```

[3] Intersection d'un segment avec une T-surface.

```
if(le T-octree de S(V2) n'est pas construit)
Build_Toctree( S(V2) );

/* calcul de l'intersection de chaque triangle de S(V1) avec S(V2) */

 pour-tout(triangle T1 ∈ S(V1))
{
   /* construction de la liste λ1 des segments de T1 ∩ S(V2) ;
      • faire L1 = liste vide ;
      • Trgl_Inter_Tsurf(T1, S(V2)->p_toctree, L1) ;
      • Attacher L1 à T1 ;
   return ;
}


void Trgl_Inter_Tsurf(T1, B, L1)
{
   if( B est un noeud de type feuille)
   {
      soit T la liste des triangles contenus dans B ;

      pour-tout( T2 ∈ T )
      {
          if( (l = T2 ∩ T2) ≠ φ)
         {
            ajouter l dans L1 ;
            attacher l à T2;
         }
      }
      return ;
   }

   /* B est un noeud de type branche */

   if( T1 intersecte la boîte correspondant à B)
   {
      Recherche des huit noeuds fils de B : B[0],...,B[7] ;
      for(i = 0;i < 8;i++)
         Trgl_Inter_Tsurf(T1, B[i], L1) ;
   }
   return ;
}
```

Définition et construction de $\lambda(T_1, S(V_2))$ associé à $L(T_1, S(V_2))$

Les segments de $L(T_1, S(V_2))$ mis bout à bout constituent un ensemble de lignes brisées polygonales $\{\lambda_i\}$ à l'intérieur de T_1 (cf section 4.4.1) que nous désignerons par $\lambda(T_1, S(V_2))$ dans ce qui suit :

$$\lambda(T_1, S(V_2)) = \{\lambda_1, \lambda_2, \ldots\}$$

La construction de ces lignes $\{\lambda_i\}$ consiste à joindre deux à deux les segments de $L(T_1, S(V_2))$ ayant une extrémité commune. Les lignes $\{\lambda_i\}$ à construire se distinguent par leurs extrémités; en effet, deux lignes différentes doivent posséder des extrémités différentes car sinon elles devraient être fusionnées.

L'algorithme de construction de ces lignes correspond à la fonction *Set_Intersection_Lines()* présente ci-dessous en pseudo-C; cette fonction est telle que :

- l'argument d'entrée L est identique à la liste de segments $L(T_1, S(V_2))$
- le résultat retourné par cette fonction correspond à la liste des lignes $\lambda(T_1, S(V_2))$. Chacune des lignes polygonales λ_i de l'ensemble $\lambda(T_1, S(V_2))$ est supposé codée sous forme d'une liste correspondant aux sommets ordonnés de la ligne.

```
LINE_LIST_t Set_Intersection_Lines( L )
{
   /* Soit λ la liste des lignes construites à partir de L */
      • faire λ = liste vide ;

   while( L ≠ φ)
   {
         • Construire le sous ensemble L_s de L contenant les segments
           connectés directement ou indirectement au premier segment
           de L;
         • faire L = L − L_s
         • Créer la ligne λ_i correspondant à la mise bout à bout de
           tous les segments de L_s;
         • Ajouter λ_i à λ ;
   }

   return( λ ) ;
}
```

4.4 Subdivision des facettes de $S(V_1)$ (et $S(V_2)$) en triangles

Considérons un triangle T_1 de $S(V_1)$. Dans ce qui suit, nous nous proposons de subdiviser T_1 en petits triangles non recoupés par les segments de $L(T_1, S(V_2))$. Notre algorithme de subdivision opère de façon globale en prenant en compte simultanément tous les segments de $L(T_1, S(V_2))$; les deux étapes de cet algorithme sont les suivantes :

1. on subdivise T_1 en régions polygonales délimitées par $L(T_1, S(V_2))$.

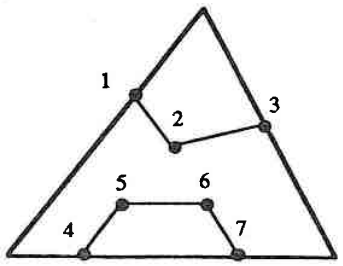


Figure 4.3 *Lignes polygonales λ_i appartenant à l'ensemble $\lambda(T_1, S(V_2))$*

2. on subdivise chaque région polygonale obtenue à l'étape 1 en petits triangles.

4.4.1 Nature de $\lambda(T_1, S(V_2))$

Comme le montre la Figure 4.3, l'ensemble $\lambda(T_1, S(V_2))$ est constitué de lignes polygonales λ_i dont les sommets sont :

1. Les intersections de T_1 avec les arêtes de $S(V_2)$ (***sommets*** 2, 5 et 6 sur la Figure 4.3).

2. Les intersections des 3 côtés de T_1 avec les facettes de $S(V_2)$ (***sommets*** 1, 3, 4 et 7 sur la Figure 4.3).

Du point de vue informatique, ces deux propriétés permettent de construire les lignes de $\lambda(T_1, S(V_2))$ sous forme d'un ensemble de lignes polygonales. Prenons par exemple la ligne λ_1 de l'exemple représenté sur la Figure 4.3 et dont les sommets sont numérotés $\{1, 2, 3\}$. Cette ligne λ_1 est constituée des segments $\{1, 2\}$ et $\{2, 3\}$ de $L(T_1, S(V_2))$ ayant le sommet $\{2\}$ pour extrémité commune. Ces deux segments correspondent à l'intersection de deux triangles T_2^{12} et T_2^{23} de $S(V_2)$ avec T_1 mais, grâce aux deux propriétés énoncées ci-dessus, il est possible d'organiser les calculs de telle façon que l'extrémité "2" ait exactement les mêmes coordonnées dans le cas du segment $\{1, 2\}$ et du segment $\{2, 3\}$. Ceci assure qu'une fois les segments de $L(T_1, S(V_2))$ calculés, on peut directement joindre les segments de cet ensemble ayant une extremité commune afin de construire les lignes polygonales $\{\lambda_i\}$ constituant $\lambda(T_1, S(V_2))$.

En fait, les lignes de $\lambda(T_1, S(V_2))$ font partie de l'intersection $\lambda(P_{T_1}, S(V_2))$ du plan P_{T_1} contenant T_1 avec $S(V_2)$ et il est intéressant d'étudier la nature de cet ensemble.

4.4.2 Nature de l'intersection du plan P_{T_1} avec $S(V_2)$

Puisque $S(V_2)$ est représentée par un ensemble de facettes triangulaires planes, on est assuré que l'intersection $\lambda(P_{T_1}, S(V_2))$ de P_{T_1} avec $S(V_2)$ est telle que :

$$\lambda(P_{T_1}, S(V_2)) = \bigcup_{T_2 \in S(V_2)} \{P_{T_1} \cap T_2\}$$

Pour toute facette T_2 de $S(V_2)$, cinq cas doivent être considérés :

- P_{T_1} n'intersecte pas T_2. Dans ce cas $(P_{T_1} \cap T_2)$ est vide.
- P_{T_1} contient seulement un sommet de T_2. Dans ce cas, $(P_{T_1} \cap T_2)$ se réduit à un point qui sera ignoré, car sa prise en compte ne modifie pas l'intersection de P_{T_1} avec $S(V_2)$.
- P_{T_1} contient un seul côté ab de T_2 (P_{T_1} passe par deux sommets a et b). Soient T_2^* le triangle adjacent de T_2 ayant ab comme côté commun avec T_2 et c (resp c^*) le troisième sommet de T_2 (resp T_2^*). Dans ce cas, suivant la position relative de c et de c^* par rapport à P_{T_1}, deux sous-cas peuvent apparaître :
  - Comme l'illustre la figure 4.4(a), soit p un point appartenant à P_{T_1}. Supposons que c et c^* soient du même côté de P_{T_1} et que p se déplace dans la direction perpendiculaire à ab. Lorsque p passe d'un côté à l'autre de ab, sa position relative par rapport à $S(V_2)$ ne change pas. Autrement dit, il est possible de traverser la surface fermée $S(V_2)$ en tout point de ab sans changer de côté relativement à $S(V_2)$. Ceci est inacceptable et c'est pourquoi ab doit être supprimé de l'intersection $(P_{T_1}, S(V_2))$.
  - Dans la figure 4.4 (b), les points c et c^* sont situés de part et d'autre de P_{T_1}. Dans ce cas, il est impossible de déplacer p d'un côté à l'autre de ab sans changer de côté par rapport à la surface fermée $S(V_2)$ et l'on en déduit que ab appartient bien à $P_{T_1} \cap S(V_2)$.
- T_2 est entièrement contenue dans P_{T_1}. Nous supposerons que ce cas ne peut pas arriver. Néanmoins, si le cas se présente, on peut toujours déplacer légèrement l'un des trois sommets de T_2 afin de se ramener au cas précédent.
- P_{T_1} intersecte deux côtés de T_2 aux points a' et b'. C'est le cas le plus général et $P_{T1} \cap S(V_2)$ est alors identique au segment $a'b'$.

Compte tennu que les cinq cas présentés ci-dessus sont les seuls possible, il est possible d'établir la propriété suivante :

Lemme 1

L'intersection $\lambda(P_{T_1}, S(V_2))$ du plan P_{T_1} avec $S(V_2)$ ne peut être représentée que par un ensemble de contours fermés polygonaux.

Démonstration

Comme indiqué précédement, $(P_{T_1} \cap T_2)$ est soit vide soit un segment, donc $\lambda(P_{T_1}, S(V_2))$ ne peut être qu'un ensemble de lignes polygonales $\{\lambda_1, \lambda_2, ..., \lambda_n\}$. Comme l'indique la Figure 4.5, supposons que parmi ces lignes, il y en ait une, λ_i, qui ne soit pas fermée et dont les sommets sont supposés numérotés de 1 jusqu'à $m+1$.

Soit (M, M^*) un segment appartenant à P_{T_1} et recoupant le segment $(m, m+1)$. Si l'on considère un point p se déplaçant dans P_{T_1} et allant de M vers M^*, deux cas doivent être considérés :

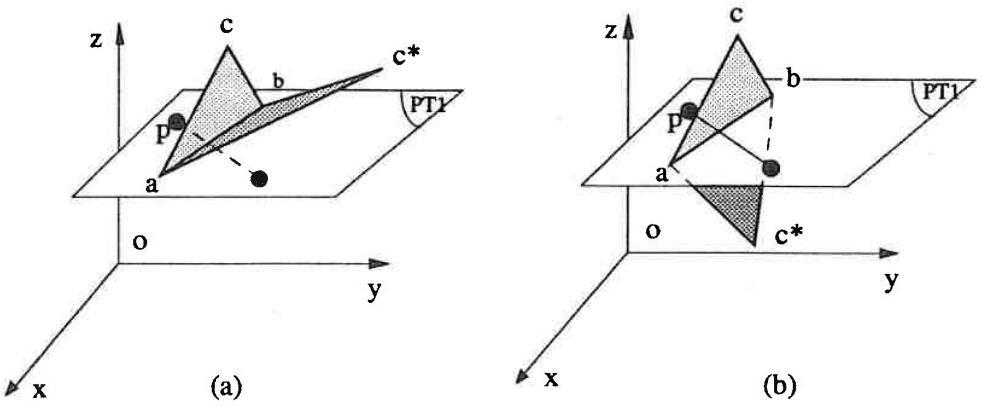


Figure 4.4 *Plan P_{T1} traverse le côté ab.*

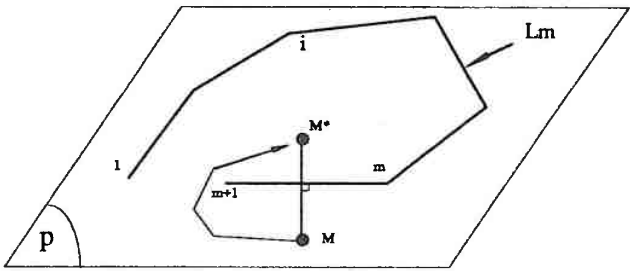


Figure 4.5 *M se déplace vers M* en deux chemins.*

1. si p se déplace sur le segment (M, M^*), alors il traverse le segment $(m, m+1)$ et change donc de position (intérieur ou extérieur) par rapport à la surface $S(V_2)$. On en déduit que M et M^* sont situés de part et d'autre de la surface $S(V_2)$.

2. compte tenu que λ_i est ouverte, p peut également se déplacer comme indiqué sur la Figure 4.5 sans couper λ_i et donc sans recouper la surface fermée $S(V_2)$. Puisque la surface $S(V_2)$ n'est pas recoupée en allant de M vers M^*, on en déduit que ces deux points sont situés d'un même côté de $S(V_2)$.

Les cas ci-dessus étant contradictoires, on en conclut que λ_i ne peut pas être ouverte.

Remarque

En pratique, deux lignes λ_i et λ_j de l'ensemble $\lambda(P_{T_1}, S(V_2))$ peuvent avoir diverses configurations l'une par rapport à l'autre :

1. λ_i peut être toute entière contenue à l'interieur de λ_j (voir Figure 4.6.a).

2. λ_i peut avoir des sommets communs avec λ_j (voir Figure 4.6.b).

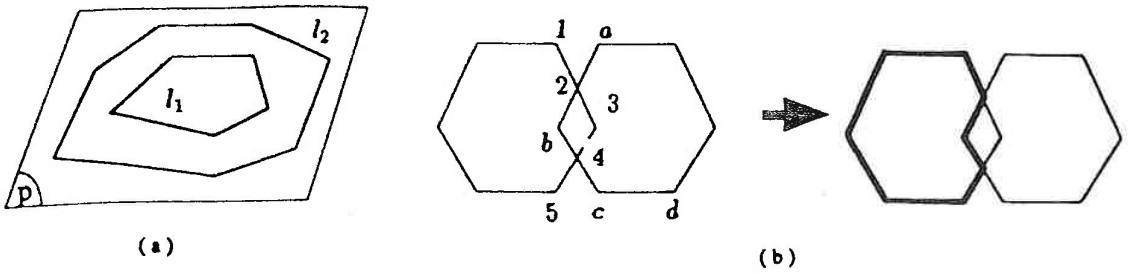


Figure 4.6 *Positions relatives des lignes $\lambda_i \in \lambda(T_1, S(V_2))$: (a) λ_1 est situé à l'interieur de λ_2; (b) deux contours croisés peuvent être convertis en deux contours non croisés mais partageant des sommets communs.*

3. λ_i peut intersecter λ_j (voir Figure 4.6.b).

Le dernier cas qui peut paraître surprenant peut arriver lorsque $S(V_2)$ est composé de T-surfaces simples fermées $\{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ se touchant par des côtés communs sans se croiser. En effet, notre algorithme de construction des lignes polygonales λ_i ne trie pas les segments en fonction des T-surfaces simples fermées σ_k dont ils proviennent.

Il est possible de démontrer que le dernier cas peut toujours être transformé en deuxième cas. Par exemple, considérons le cas représenté sur la Figure 4.6.b pour lequel les contours λ_i et λ_j sont constitués des sommets suivants :

$$\left| \begin{array}{lcl} \lambda_i & \longleftrightarrow & \{\ldots, 1, 2, 3, 4, 5, \ldots\} \\ \lambda_j & \longleftrightarrow & \{\ldots, a, 2, b, 4, c, d, \ldots\} \end{array} \right.$$

il suffit de redéfinir λ_i et λ_j de la façon suivante :

$$\left| \begin{array}{lcl} \lambda_i & \longleftrightarrow & \{\ldots, 1, 2, b, 4, 5, \ldots\} \\ \lambda_j & \longleftrightarrow & \{\ldots, a, 2, 3, 4, c, d, \ldots\} \end{array} \right.$$

4.4.3 Types des lignes λ_i appartenant à $\lambda(T_1, S(V_2))$

Dans ce paragraphe, comme le suggère la Figure 4.7, nous nous proposons d'identifier les quatre types de courbes λ_i pouvant appartenir à l'ensemble $\lambda(T_1, S(V_2))$. Pour ce faire, on s'appuie sur l'étude présentée au paragraphe précédent tout en tenant compte du fait que $L(T_1, S(V_2))$ $= \mathrm{L}(P_{T_1}, S(V_2)) \cap T_1$:

- TYPE 1 $\Longrightarrow$ La ligne polygonale traverse deux côtés différents du triangle (voir Figure 4.7(a)).

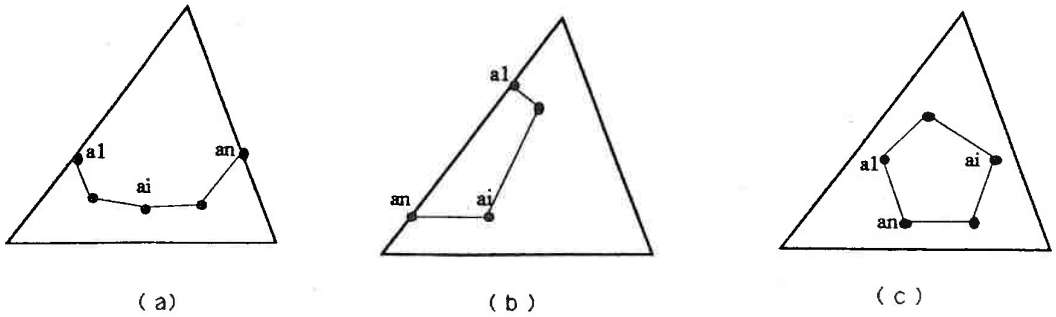


Figure 4.7 *Trois types des lignes d'intersection élémentaires* $\lambda(T_1, S(V_2))$.

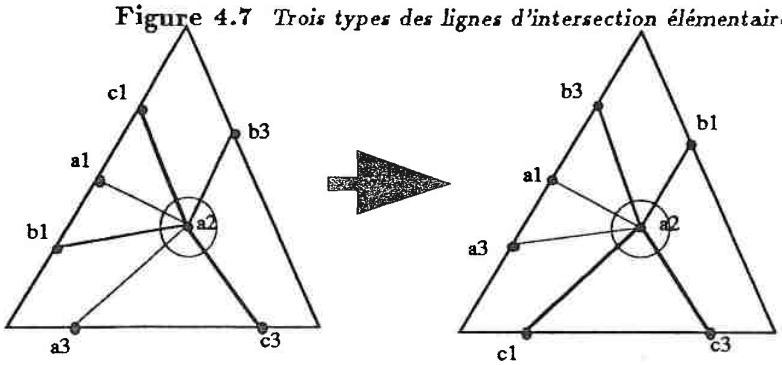


Figure 4.8 *les lignes d'intersection croisées de* $\lambda(T_1, S(V_2))$ *peuvent toujours être converties en lignes non croisées.*

- TYPE 2 $\Longrightarrow$ La ligne polygonale entre et sort sur un seul côté du triangle (voir Figure 4.7(b)).
- TYPE 3 $\Longrightarrow$ La ligne polygonale ne coupe aucun côté du triangle car elle est située à l'intérieur de celui-ci (voir Figure 4.7(c)).
- TYPE 4 $\Longrightarrow$ Deux (ou plusieurs) lignes polygonales ayant l'un des trois précédents types présentés ci-dessus se croisent à l'interieur du triangle (cf Fig 4.8).

La Figure 4.9 présente quelques exemples concrets d'ensembles $\lambda(T_1, S(V_2))$. En pratique, ils correspondent généralement à des combinaisons des trois premiers types d'intersection élémentaires présentés ci-dessus.

Nous verrons, dans les deux remarques suivantes, que le type 4 ainsi que les cas de dégénéréscence peuvent toujours se ramener aux trois premiers types présentés ci-dessus. En conséquence, seuls ces trois premiers types sont à prendre en considération et le type d'une ligne λ_i appartenant à $\lambda(T_1, S(V_2))$ peut toujours être déterminé à l'aide de la fonction suivante décrite en pseudo-C :

```
void Set_Type_of_Intersection_Line(λi, T1)
{
```

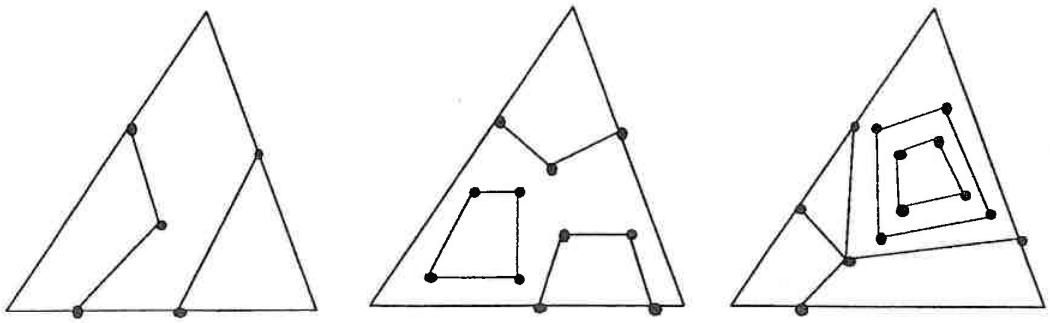


Figure 4.9 *un cas concret $\lambda(T_1, S(V_2))$ est souvent la combinaison des cas élémentaires.*

```
    Soient V_f et V_e respectivement le premier et le dernier sommet de λ_i;

    if( V_f et V_i appartiennent au même côté de T_1)
        λ_i.type = 1 ;
    else if(V_f et V_i appartienent à deux côtés différents de T_1)
        λ_i.type = 2 ;
    else
        λ_i.type = 3 ;

    return ;
}
```

A partir de maintenant et comme le suggère la fonction ci-dessus, nous supposerons que seuls les trois premiers types peuvent être rencontrés; cela suppose que le type 4 ainsi qu'un certain nombre de types "dégénérés" soient convertis en une ou plusieurs lignes de type 1, 2 ou 3.

Remarque 1: Conversion du type 4

Supposons que $S(V_2)$ soit constitué par une suite $\{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ de T-surfaces simples fermées. Suivant la position relative de ces diverses T-surfaces σ_i, une ligne de type 4 peut être générée dans les cas suivants :

- Si T_1 passe par un sommet d'une T-surface σ_i, alors le type 4 peut arriver. En modifiant légèrement la position de T_1, il est toujours possible de supprimer ce cas; comme le montre la Figure 4.10, ce type de ligne se transforme alors en plusieurs lignes ayant les types 1, 2 ou 3.

- Si deux T-surfaces σ_i et σ_j sont jointes par des arètes communes, alors le type 4 peut apparaître si T_1 traverse ces arètes. Il est encore possible de transformer ce type de ligne en plusieurs lignes ayant les types 1, 2 ou 3; pour ce faire, on procède de la manière illustrée par l'exemple représenté sur la Figure 4.8.(a) : Soient λ_1, λ_2 et λ_3 les trois lignes qui se

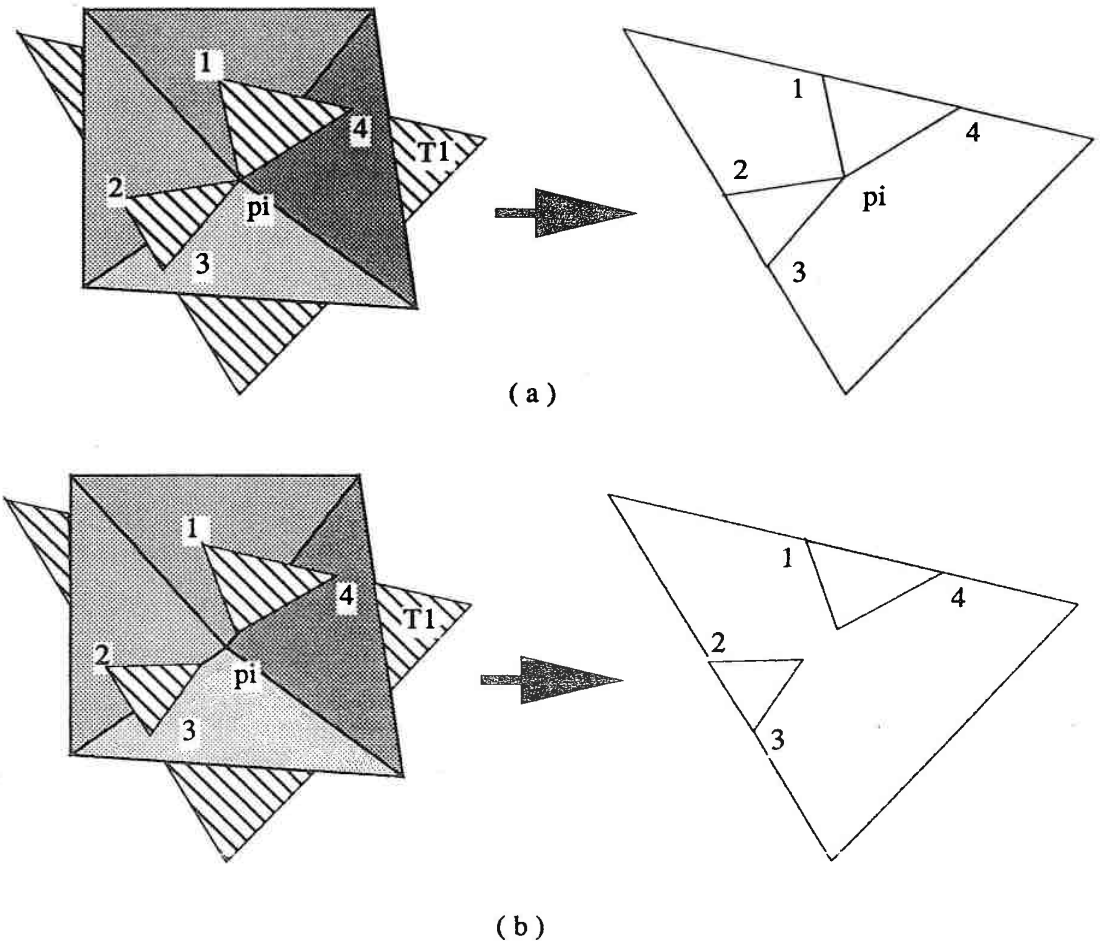


Figure 4.10 *(a) le triangle T_1 traverse un sommet p_i de $S(V_2)$; (b) un déplacement léger de T_1 peut éviter que T_1 traverse le sommet p_i de $S(V_2)$.*

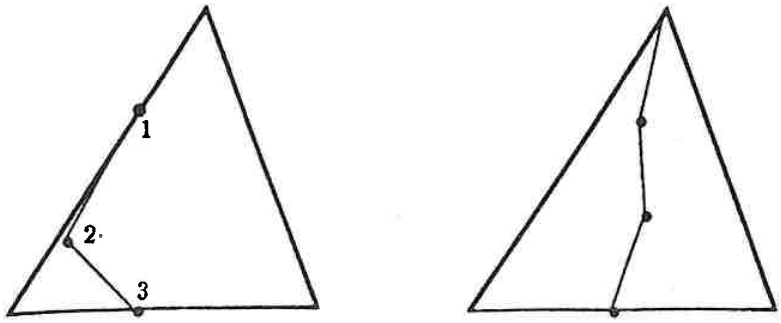


Figure 4.11 *Exemples de cas de lignes λ_i dégénérés: (a) le segment 12 est coincide avec un côté du triangle; (b) la ligne passe par un sommet du triangle.*

croisent à l'intérieur de T_1 et sont constituées des sommets suivants :

$$\left| \begin{array}{lcl} \lambda_1 & \longleftrightarrow & \{a_1, a_2, a_3\} \\ \lambda_2 & \longleftrightarrow & \{b_1, a_2, b_3\} \\ \lambda_3 & \longleftrightarrow & \{c_1, a_2, c_3\} \end{array} \right.$$

Le point a_2 partage chacune de ces trois lignes en deux parties et le nombre de ces morceaux de ligne est donc nécessairement pair. Si l'on colle deux à deux les morceaux rencontrés lorsque l'on tourne autour de a_2 dans le sens des aiguilles d'une montre, alors on obtient une série de lignes λ_1^*, λ_2^* et λ_3^* appartenant aux trois premiers types et ayant les sommets suivants :

$$\left| \begin{array}{lcl} \lambda_1^* & \longleftrightarrow & \{a_1, a_2, c_1\} \\ \lambda_2^* & \longleftrightarrow & \{b_3, a_2, c_3\} \\ \lambda_3^* & \longleftrightarrow & \{a_1, a_2, c_3\} \end{array} \right.$$

Ces trois nouvelles lignes ainsi générées ne se croisent plus mais partagent un sommet commun.

Il est possible que pour un même triangle T_1 on ait plusieurs cas de figure de type 4; on peut alors opérer itérativement comme décrit ci-dessus pour convertir les lignes correspondantes en lignes de types 1, 2 ou 3.

Remarque 2: Types dégénérés

Les quatre types élémentaires présentés ci-dessus peuvent, dans certains cas limite, dégénérer comme l'indique la Figure 4.11. En pratique, ces types dégénérés peuvent toujours être identifiés à l'un des trois premiers types élémentaires de lignes polygonales..

4.4.4 Partition de T_1 en régions polygonales

Dans ce qui suit, nous nous proposons de partitionner le triangle T_1 sous forme d'un ensemble $\Pi(T_1, S(V_2))$ de polygones simples engendrés par les lignes λ_i appartenant à $\lambda(T_1, S(V_2))$.

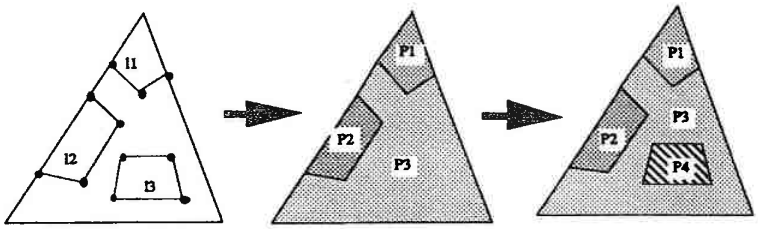


Figure 4.12 *Partition d'un triangle T_1 en régions polygonales simples définies par les lignes polygonales $\lambda_i \in \lambda(T_1, S(V_2))$.*

Les polygones simples ainsi générés peuvent, éventuellement, contenir des trous mais chaque polygone $\Pi_k \in \lambda(T_1, S(V_2))$ doit satisfaire les conditions suivantes :

- Les côtés de Π_k ne peuvent être que :
  - des lignes complètes $\lambda_i \in \lambda(T_1, S(V_2))$
  - des morceaux de côtés de T_1
- Les sommets de Π_k ne peuvent être que :
  - des sommets de T_1
  - des sommets de $\lambda(T_1, S(V_2))$

Afin de faciliter l'exposé, nous utiliserons les notations suivantes dans ce qui suit :

$$\begin{array}{lcl} \lambda^{1,2}(T_1, S(V_2)) & = & \text{lignes polygonales de } \lambda(T_1, S(V_2)) \text{ de type 1 et type 2.} \\ \lambda^{3}(T_1, S(V_2)) & = & \text{lignes polygonales de } \lambda(T_1, S(V_2)) \text{ de type 3.} \end{array}$$

Le fait que les lignes λ_i de types 1, 2 ou 3 ne se croisent pas à l'intérieur de T_1 assure que :

- T_1 peut être décomposé en régions polygonales simples par les lignes de $\lambda^{1,2}(T_1, S(V_2))$. L'ensemble de ces régions est noté $\Pi(T_1, S(V_2)) = \{\Pi_1, \Pi_2, \ldots, \Pi_n\}$ et l'on suppose que les sommets de chaque polygone Π_k appartient soit à ceux de T_1 soit à ceux de $\lambda^{1,2}(T_1, S(V_2))$.
- Chaque ligne λ_i de $\lambda^3(T_1, S(V_2))$ étant fermée peut toujours être considérée comme un trou polygonal situé à l'intérieur d'un polygone Π_k de $\Pi(T_1, S(V_2))$. En d'autres termes, on peut toujours poser :

  $$\lambda_i \in H(\Pi_k)$$

  On en conclut que T_1 peut toujours être partitionné en régions polygonales simples trouées ou non. La figure 4.12 montre un cas concret d'une telle partition.

4.4.5 Procédé de partition

Soient $m_{1,2} \geq 0$ le nombre des lignes polygonales $\lambda_i^{1,2}$ de $\lambda^{1,2}(T_1, S(V_2))$ et $m_3 \geq 0$ le nombre de lignes polygonales λ_i^3 de $\lambda^3(T_1, S(V_2))$:

$$\begin{aligned} \lambda^{1,2}(T_1, S(V_2)) &= \{\lambda_1^{1,2}, \lambda_2^{1,2}, \ldots, \lambda_{m_{1,2}}^{1,2}\} \\ \lambda^3(T_1, S(V_2)) &= \{\lambda_1^3, \lambda_2^3, \ldots, \lambda_{m_3}^3\} \end{aligned}$$

La partition de T_1 en régions polygonales simples peut être effectuée en deux étapes correspondant aux deux ensembles de lignes $\lambda^{1,2}(T_1, S(V_2))$ et $\lambda^3(T_1, S(V_2))$:

étape 1. Traitement des lignes de $\lambda^{1,2}(T_1, S(V_2))$[4]

Si $m_{1,2} > 0$, alors, T_1 peut être divisé en deux polygones simples P_1 et P_2 par la ligne $\lambda_1^{1,2} \in \lambda^{1,2}(T_1, S(V_2))$. Il est évident que $\lambda_2^{1,2}$ (si $m_{1,2} > 1$) appartient à l'un des deux polygones P_1 ou P_2 qui peut lui même être ensuite subdivisé en deux polygones. Si l'on répète cette procédure itérativement, à l'étape $i \leq m_{1,2}$, le triangle T_1 est subdivisé en $(i+1)$ polygones. Si la ligne $\lambda_{i+1}^{1,2}$ existe, elle appartient obligatoirement à l'un de ces $i+1$ polygones, qui est lui même subdivisé par $\lambda_{i+1}^{1,2}$. Finalement T_1 est partitionné par les lignes de $\lambda^{1,2}(T_1, S(V_2))$ en $(m_{1,2}+1)$ régions polygonales. Remarquons que, si $m_{1,2} = 0$, alors T_1 ne doit pas être modifié.

On conclut de ceci que, pour pouvoir effectuer le traitement décrit ci-dessus, il suffit de définir la division d'un polygone simple quelconque par une ligne $\lambda_i^{1,2}$ contenue dans ce polygone. Afin de présenter l'algorithme utilisé pour ce faire, comme le suggère la Figure 4.13, supposons que $\lambda_i^{1,2}$ comporte m sommets $\{a_j\}$ et que que le polygone simple P partitionné par $\lambda_i^{1,2}$ comporte n sommets $\{p_k\}$:

$$\begin{aligned} \lambda_i^{1,2} &= \{a_1, a_2, \ldots, a_m\} \\ P &= \{p_1, p_2, \ldots, p_n\} \end{aligned}$$

En remarquant que les extrémités a_1 et a_m de $\lambda_i^{1,2}$ appartiennent nécessairement à la frontière de P, soient (p_i, p_{i+1}) et (p_j, p_{j+1}) les côtés de P tels que :

$$\begin{aligned} a_1 &\in (p_i, p_{i+1}) \\ a_m &\in (p_j, p_{j+1}) \end{aligned}$$

La partition de P est alors réalisée de la façon suivante :

- Si $i < j$, alors les deux polygones résultant de la partition sont:

  $P_1(p_1, p_2, \ldots, p_i, a_1, a_2, \ldots, a_m, p_{j+1}, \ldots p_n)$ et $P_2(a_1, a_2, \ldots, a_m, p_{j+1}, \ldots, p_{i+1})$.

- Si $i > j$, alors les deux polygones résultant de la partition sont:

  $P_1(p_1, p_2, \ldots, p_i, a_m, a_{m-1}, \ldots, a_1, p_{i+1}, \ldots p_n)$ et $P_2(a_1, a_2, \ldots, a_m, p_j, \ldots, p_{i-1})$.

- Si $i = j$, alors a_1 et a_m appartiennent au même côté (p_i, p_{i+1}) et les deux polygones résultant de la partition sont:

  $P_1(p_1, p_2, \ldots, p_i, a_1, a_2, \ldots, a_m, p_{i+1}, \ldots p_n)$ et $P_2(a_1, a_2, \ldots, a_m)$.

[4]Cette étape nécessite de tester l'appartenance d'un point à un polygone simple (cf Chapitre 1.3)

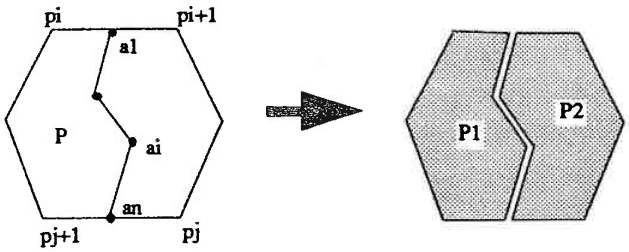


Figure 4.13 *Partition d'un polygone simple par une ligne de* $\lambda^{1,2}(T_1, S(V_2))$

étape 2. Traitement des lignes de $\lambda_3(T_1, S(V_2))$

A l'issue de l'étape 1, T_1 est décomposé en un ensemble $\Pi(T_1, S(V_2))$ de $m_{1,2}+1$ polygones P_i :

$$\Pi(T_1, S(V_2)) = \{P_1, P_2, ...P_{m_{1,2}+1}\}$$

Si $m_3 \neq 0$, alors λ_1^3 appartient à un polygone $P_i \in \Pi(T_1, S(V_2))$ et λ_1^3 divise P_i en deux régions :

- la première région est entourée par λ_1^3
- la deuxième région est le polygone P_i dont l'intérieur possède un trou délimité par le contour fermé λ_1^3.

En appliquant cette procédure pour chaque ligne de $\lambda_3(T_1, S(V_2))$, T_1 est décomposé en un ensemble de régions polygonales simples dont certaines possèdent des trous.

Algorithme proposé

Les deux étapes du processus de partition de T_1 par les lignes de $\lambda(T_1, S(V_2))$ peuvent être mises en oeuvre par la fonction suivante décrite en pseudo-C :

```
POLYGON_t Split_Trgl_to_Polygons(T1, λ(T1, S(V2)) )
{
   /* Soit Π la liste de polygones à construire */
   Π = T1 ;

   pour-tout( λi ∈ λ(T1, S(V2)))
   {
      if( λi.type == 3) continue ;

      pour-tout(Πj ∈ Π)
      {
```

- division de Π_j par λ_i en deux sous-polygones P_1 et P_2;
- ajout de P_1 *et de* P_2 à Π :

$$\Pi = \Pi \cup P_1 \cup P_2;$$

- suppression de λ_i de λ :

$$\lambda = \lambda - \lambda_i \ ;$$

```
    }
  }

  pour-tout( λi ∈ λ(T1, S(V2)) )
  {
    pour-tout( Πj ∈ Π )
    {
      if( λi.type != 3 ) continue ;

      if( λi à l'intérieur de Π)
      {
        H(Π ) = H(Π ) ∪ λi ;
        ajouter λi à Π ;
        break ;
      }
    }
  }

  return( Π ) ;
}
```

4.5 Triangulation d'un polygone simple (ayant éventuellement des trous)

4.5.1 Position du problème

La division d'un polygone plan P (avec ou sans trous) en triangles consiste à trouver un ensemble de triangles $\mathcal{T}(P) = \{T_1, T_2, \ldots, T_n\}$ tels que :

- les sommets de chaque triangle de $\mathcal{T}(P)$ appartiennent à l'ensemble des sommets de P.
- deux triangles distincts de $\mathcal{T}(P)$ ont, au plus, un côté commun ou un sommet commun.
- la somme des aires des triangles de $\mathcal{T}(P)$ est égale à l'aire de P.

Pour un polygone donné P, si l'ensemble des triangles de $\mathcal{T}(P)$ n'est pas vide, alors P sera dit ***triangulable*** (la triangulation n'est pas nécessairement unique). En général, P est plongé dans un espace tri-dimensionnel mais, puisque ses sommets sont situés dans un plan, il est possible de réaliser la décomposition en 2D. Supposons que l'on projete P sur un plan de coordonnées (xoy) et que cette projection ne se réduise pas à un segment, il est évident que si cette projection est

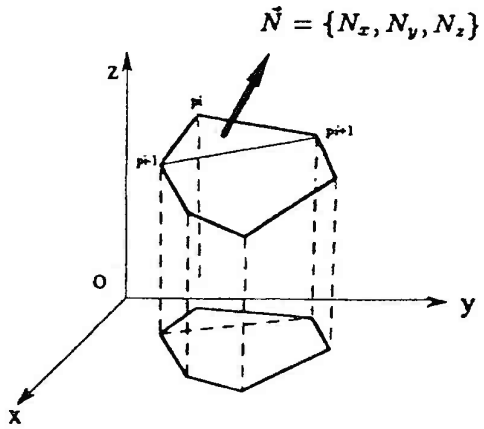


Figure 4.14 *La triangulation d'un polygone simple est equivalente à celle de sa projection sur l'un des trois plan de coordonnées* (xoy), (yoz) *et* (zox).

triangulable, alors P peut être triangulé de la même manière. En pratique, pour des raisons de précision, il est souhaitable de projeter P sur celui des trois plans de coordonnées (xoy), (yoz) ou (zox) pour lequel l'aire de la projection est maximale.

Comme l'indique la figure 4.14, soit P un polygone admettant $\{p_{i-1}, p_i, p_{i+1}\}$ comme ses trois sommets consécutifs non alignés. On peut alors calculer la normale de P :

$$\vec{N} = p_i\vec{p_{i-1}} \wedge p_i\vec{p_{i+1}}$$

soit N_x, N_y et N_z les trois composantes de $\vec{N}$. Il est évident que le plan de projection optimal est:

$$\begin{aligned}
(xoy)\ ,\ si\ |N_z| &= max\{|N_x|, |N_y|, |N_z|\} \\
(yoz)\ ,\ si\ |N_x| &= max\{|N_x|, |N_y|, |N_z|\} \\
(zox)\ ,\ si\ |N_y| &= max\{|N_x|, |N_y|, |N_z|\}
\end{aligned}$$

Une fois le plan optimal choisi, le problème de la décomposition de P consiste à décomposer la projection de P en triangles.

Dans la littérature, plusieurs méthodes (voir [5], [7], [27]) ont été proposées pour trianguler des polygones simples. En fait, la définition des polygones simples utilisée dans ces algorithmes est beaucoup plus restrictive que la nôtre et n'est pas utilisable pour les applications que nous envisageons dans ce mémoire (cf Figure 4.15).

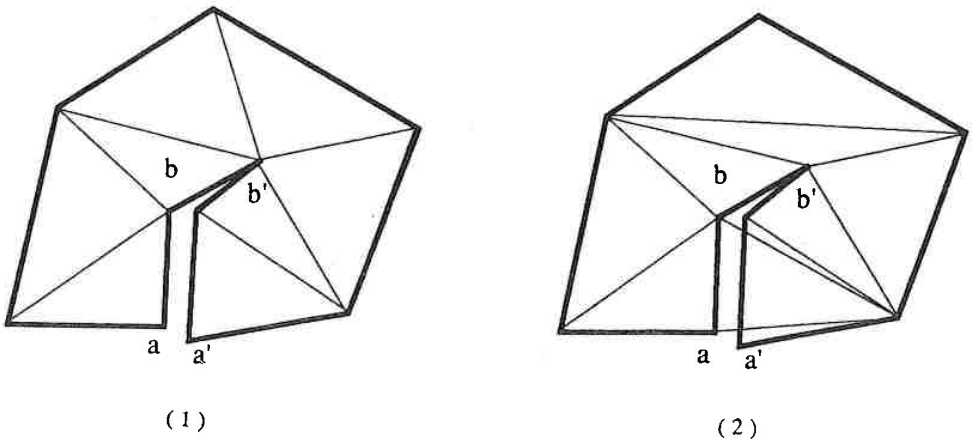


Figure 4.15 *Le sommet a coicide avec a' et b coincide avec b' : (1) Triangulation correcte (2)Triangulation incorrecte obtenue par les méthodes existantes.*

4.5.2 Décomposition d'un polygone simple sans trous

Existence de la décomposition

Lemme 2

En 2D, pour un polygone simple quelconque sans trous P ayant plus de 3 sommets, il existe au moins une diagonale joignant deux sommets non consécutifs de P qui est entièrement contenue à l'intérieur de P.

Démonstration

Soit $n > 3$ le nombre de sommets du polygone simple sans trous P. La somme des angles internes θ_i de P vérifie toujours la relation suivante :

$$\sum_{i=1}^{n} \theta_i = (n-2) \cdot 180°$$

la moyenne des angles internes est donc :

$$\theta = \frac{1}{n} \cdot \sum_{i=1}^{n} \theta_i = 180° - \frac{360°}{n}$$

Puisque θ est toujours inférieur à 180°, il existe donc au moins un angle interne, disons $\theta_i(p_{i-1}, p_i, p_{i+1})$ inférieur à 180°. Soient $m \in [0,\ n-3[$ le nombre de sommets de P situés à l'intérieur du triangle T_i dont les trois sommets sont $\{p_{i-1}, p_i, p_{i+1}\}$ et soit $I(i) = \{k_1, k_2, ..., k_m\}$ l'ensemble des indices de ces m sommets :

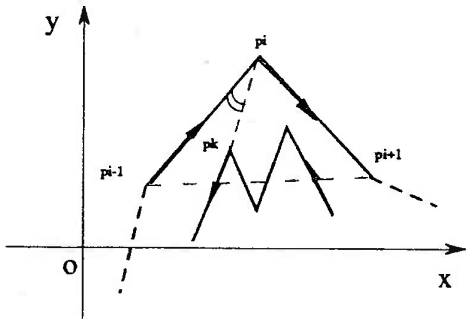


Figure 4.16 *Angle $\theta(p_{i-1},p_i,p_k)$ est l'angle le plus petit.*

- si $m = 0$, alors le segment $p_{i-1}p_{i+1}$ est la diagonale cherchée, car $\theta_i(p_{i-1},p_i,p_{i+1})$ est un angle interne de P.

- si $m \in [1,\ n-3]$, alors comme le suggère la figure 4.16, parmi les sommets $\{p_k | k \in I(i)\}$ on peut toujours trouver un sommet p_j tel que :

$$\theta(p_{i-1},p_i,p_j) = min\{\theta(p_{i-1},p_i,p_j)\} \quad k \in I(i).$$

On en déduit que le segment p_ip_j ne coupe aucun côté de P et compte tenu du fait que $\theta_i(p_{i-1},p_i,p_{i+1})$ est un angle interne de P, on est assuré que le vecteur $p_i\vec{p}_j$ est orienté vers l'intérieur de P. Il s'en suit que le segment p_ip_j n'a pas de possibilité de sortir de P, donc il est contenu entièrement dans P et constitue la diagonale cherchée.

Lemme 3

En $2D$, pour un polygone simple quelconque sans trous P ayant plus de 3 sommets, il existe au moins un sommet p_i tel que la diagonale $p_{i-1}p_{i+1}$ soit toute entière contenu à l'intérieur de P.

Démonstration

Considérons le processus iteratif suivant décrit en pseudo-C:

```
begin
   S = φ ;
   P* = P ;

   while( P* n'est pas un triangle)
   {
```

- compte tenu du Lemme 1, il existe une diagonale interne $p_i p_j$ de P^* qui divise P^* en deux polygones adjacents $P^*(1)$ et $P^*(2)$ tel que:

$$P^*(1) \cap P^*(2) = p_i p_j ;$$

- sans nuire à la généralité, on peut toujours supposer que $P^*(1)$ est celui des deux polygones vérifiant la proprieté suivante:

$$P^*(1) \cap S = \phi ;$$

- on pose

$$\begin{aligned} S &= \{p_i p_j\}; \\ P^* &= P^*(1); \end{aligned}$$

}
end

ce processus iteratif se termine en $n-2$ étapes et le segment S obtenu est précisement la diagonale interne cherchée.

Corollaire

Tout polygone simple sans trou est *triangulable*.

Algorithme de décomposition[5]

Compte tenu des lemmes 2 et 3 ci-dessus, la décomposition d'un polygone simple sans trous P en triangles peut être effectuée de deux façons différentes décrites ci-dessous en pseudo-C :

Procédure itérative en deux étapes (selon Lemme 2)

void **Triangulate_Simple_Polygon_Iter**($P, T(P)$)
{
$P^* = P$;
$T(P) = \phi$;

while(P^* **n'est pas un triangle**)
{
1 : recherche du sommet p_i de P^* tel que $p_{i-1}p_{i+1}$ soit une diagonale contenue dans P^*.

2 :

- construction du triangle $T(p_{i-1}, p_i, p_{i+1})$: $T(p) = T(p) \cup T$.
- P^* = polygone déduit de P^* par suppression du sommet p_i.

[5]Ces algorithmes nécessitent de tester l'intérieurité d'une diagonale polygone simple (cf Chapitre 1.3)

```
    }
    T(P) = T(p) ∪ P* ;
}
```

procédure récursive (selon Lemme 3)

On suppose $\mathcal{T}(P) = \phi$ en entrée :

```
void Triangulate_Simple_Polygon_Recur(P, T(P))
{
  if(P est un triangle)
     T(P) = T(p) ∪ P ;
  else
```

- recherche de la diagonale $p_i p_j$ contenue dans P.
- Division de P en $P(1)$ et $P(2)$ par $p_i p_j$.
- /* on rappelle cette fonction pour $P(1)$ et pour $P(2)$ */
  - Triangulate_Simple_Polygon_Recur($P(1), \mathcal{T}(P)$) ;
  - Triangulate_Simple_Polygon_Recur($P(2), \mathcal{T}(P)$)) ;

```
  return() ;
}
```

Comparaison de ces deux algorithmes

- Dans l'algorithme 1, ce qui est le plus coûteux est la recherche de p_i dans P^*. Pour ce faire, il faut effectuer en moyenne $2 \cdot k$ tests d'intersection[6] de deux segments en 2D. Le nombre total de tests est alors :

$$\sum_{i=0}^{n-4} \frac{n+1-i}{2} \cdot k$$

  On constate que la complexité de l'algorithme est en moyenne $O(n^2)$, mais en pratique n est souvent petit ($n < 10$) et cette méthode est malgré tout assez rapide.

- Dans l'algorithme 2, la recherche du segment $p_i p_j$ est l'opération la plus coûteuse en temps de calcul. Pour effectuer cette recherche, il faut également $2 \cdot k$ tests d'intersection de deux segments en 2D. Le nombre de niveaux de la récursion est en moyenne égale à $log(\frac{n}{3})$, donc le nombre total de tests est $\frac{n \cdot k}{2} log(\frac{n}{3})$ ($O(n \cdot log(n))$) (voir [5] [7]).

Du point de vue de la mise en oeuvre (En langage C), l'algorithme 1 est plus simple que l'algorithme 2. En revanche, le dernier est plus rapide. Pour améliorer le temps de calcul, lorsque le nombre de sommets de P devient grand, il est conseillé d'utiliser l'algorithme 2.

[6] k est une constante caractérisant la complexité de P^*. Si P^* est un polygone convexe alors $k = 1.0$ et si P^* est concave, alors k est proportionnel au "degré" de concavité de P^*

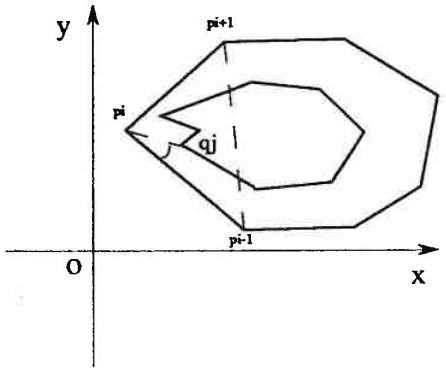


Figure 4.17 *L'angle $\theta(q_j, p_i, p_{i-1})$ est le plus petit.*

4.5.3 Décomposition d'un polygone simple ayant un seul trou

Lemme 4

Dans un plan, pour un polygone simple P ayant un seul trou polygonal Q, il existe au moins un segment p_iq_j joignant un sommet p_i de P et un sommet q_j de Q tel que p_iq_j soit situé à l'intérieur de P et à l'extérieur de Q.

Démonstration

Soient P un polygone simple avec $n(>2)$ sommets et Q un polygone simple ayant $m(>2)$ sommets qui sont situés à l'intérieur de P. D'après le corollaire du Lemme 3, si P n'avait pas de trou, alors P pourrait être triangulé par un ensemble de triangles $\mathcal{T}(P) = \{T_1, T_2, ..., T_{n-2}\}$. Parmi ces triangles, il en existe au moins un, disons T_i, dont deux côtés sont deux côtés consécutifs de P :

$$T_i = T(p_{k-1}, p_k, p_{k+1})$$

Soit $l \in [0, m]$ le nombre de sommets de Q situés à l'intérieur de T_i et soit $I(i) = \{j_1, j_2, \ldots, j_l\}$ l'ensemble des indices de ces l sommets:

- si $m \neq 0$, alors comme le suggère la figure 4.17, parmi les sommets $\{q_j | j \in I(i)\}$, il existe toujours un sommet q_j tel que :

$$\theta(p_{i-1}, p_i, q_j) = min\{\theta(p_{i-1}, p_i, q_k)\} \quad k \in I(i)$$

  Il est évident que le segment p_iq_j ne coupe aucun côté de P et de Q. Ceci implique que p_iq_j est situé à la fois à l'intérieur de P et à l'extérieur de Q et constitue le segment cherché.

- si $m = 0$, alors Q est situé entièrement à l'extérieur de T_i. Le problème se réduit donc à la recherche de la diagonale p_iq_j pour le polygone P^* qui est déduit de P en enlevant le sommet p_i.

En appliquant cette méthode itérativement, deux sous-cas apparaîssent

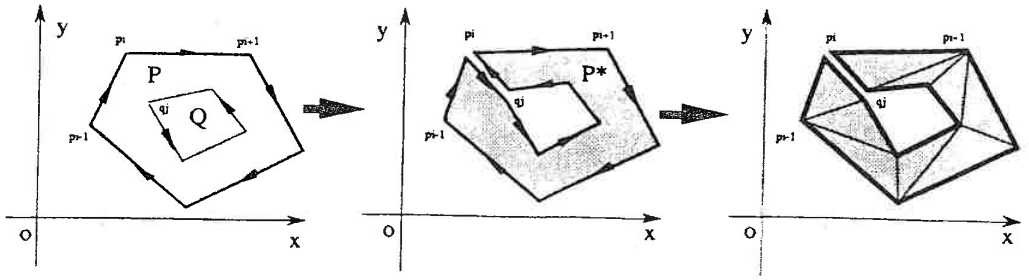


Figure 4.18 *Partition d'un polygone simple avec un seul trou.*

1. soit le segment $p_i q_j$ est trouvé à une étape $r \in]0, n-3]$.
2. soit à l'étape $r = n-3$, P devient un triangle et Q est contenu dans P. Dans ce cas, le segment existe toujours.

Corollaire

Un polygone simple P ayant un trou Q à l'intérieur peut toujours être transformé en un polygone simple P^* sans trou tel que :

- P^* entoure la région située à l'intérieur de P et à l'extérieur de Q.
- P^* possède les mêmes sommets que ceux de P (éventuellement dupliqués).

Démonstration

Supposons que P et Q aient respectivement des sommets p_i et q_j tels que :

$$P = \{p_1, p_2, ..., p_m\}$$

$$Q = \{q_1, q_2, ..., q_n\}$$

Soit $p_i q_j$ le segment défini selon le lemme 3 et soit p_i^* (resp q_j^*) le sommet dupliqué de p_i (resp q_j). Comme l'indique la Figure 4.18, si les sommets de P et de Q sont numérotés dans le sens des aiguilles d'une montre, alors le nouveau polygone P^* sans trou sera défini de la façon suivante :

$$P^* = \{p_1, p_2, ...p_i, q_j, q_{j-1}, ...q_{j+2}, q_j^*, p_i^*, p_{i+1}, p_{i+2}, ..., p_n\}$$

On constate facilement que la région entourée par P^* est identique à celle située à l'intérieur de P et à l'extérieur de Q et l'on en déduit que P peut être triangulé de la même façon que P^*.

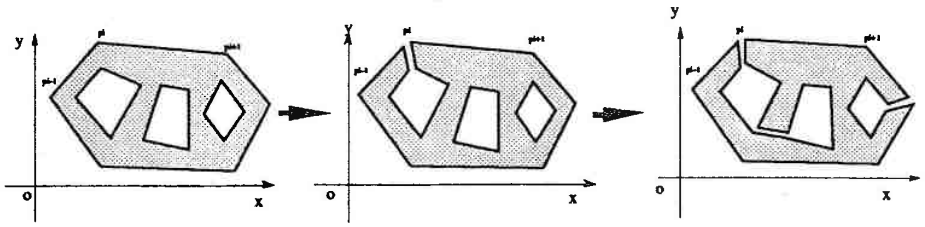


Figure 4.19 *Partition d'un polygone simple avec plusieurs trous.*

Analyse de Compléxité

Soit n_P (resp n_Q) le nombre de sommets de P (resp Q). l'algorithme peut être décomposé en 3 étapes suivantes :

1. Prenant un sommet q_i de Q, on cherche le sommet p_j de P qui est le plus proche de q_i que tous les autres sommets de P. Le coût de calcul est donc de $O(n_p + n_Q)$.
2. Tester si le segment $q_i p_j$ apartient à P et se situe à l'extérieur de Q. Le coût de calcul est également de $O(n_p + n_Q)$.
3. Transformer P en un polygone simple sans trou P^* et trianguler P^* en utilisant la procédure récursive (cf page 76). Le coût du calcul est donc de $O((n_p+n_Q)\cdot log(n_p+n_Q))$.

On conclut de ces 3 étapes ci-dessus que l'algorithme fonctionne en moyenne de $O((n_p + n_Q) \cdot log(n_p + n_Q))$ (voir [27]).

Décomposition d'un polygone ayant plusieurs trous

Soit P un polygone simple affecté par un ensemble $H(P)$ de trous polygonaux :

$$H(P) = \{Q^1, Q^2, ..., Q^m\}$$

Les polygones $Q^k \in H(P)$ sont des polygones simples non croisés entre eux mais pouvant avoir des sommets communs. Par une démonstration analogue à celle du lemme 4, on peut démontrer qu'il existe au moins un segment $p_i q_j^k$ joignant un sommet de P et un sommet d'un polygone Q^k de $H(P)$ qui est contenu à l'intérieur de P et à l'extérieur des polygones de $H(P)$.

Comme l'indique la figure 4.19(b), par une démonstration analogue à celle du corollaire du lemme 4, la région située à l'intérieur de P et à l'extérieur des polygones de $H(P)$ peut être transformée en un polygone simple P_1 tel que :

$$H(P_1) = H(P) - \{Q^k\}$$

Si l'on répète cette procédure itérativement, à l'étape m, P est transformé en un polygone simple sans trou P^* et la triangulation de P consiste donc à trianguler le polygone P^*.

L'algorithme de décomposition s'ecrit en pseudo-C sous la forme suivante :

void **Triangulate_Polygon**($P, \mathcal{T}(P)$)
{
Soit $\mathcal{T}$(P) la liste de triangles obtenue à partir de P;

$\mathcal{T}(P) = \phi$;

if(H(P) == ϕ)

Triangulate_Simple_Polygon_Recur($P, \mathcal{T}$(P))
return ;

while(H(P) $\neq \phi$)
{
- Recherche du segment $p_i q_j^k$;
- Transformer P ayant $H(P) = Q^k$ en un polygone simple P' et poser $P = P'$;
- Enlever Q^k de $H(P)$;

}

Triangulate_Simple_Polygon_Recur($P^*, \mathcal{T}(P^*)$)
$\mathcal{T}(P) = \mathcal{T}(P^*)$;
}

Analyse de Compléxité

Soient n_P (resp n_Q^k) le nombre de sommets de P (resp Q^k) et m le nombre de trous. Par une analyse identique à celle du polygone ayant un seul trou, le coût du calcul de l'élimination du premier trou Q^1 est donc de $O(n_p + n_Q^1)$. La transformation de P en un polygone simple P^* est alors :

$$\sum_{k=1}^{m} O(n_P + n_Q^k) \quad \leq \quad m \cdot O(n_P + \sum_{k=1}^{m} n_Q^k)$$

posons :

$$H = \sum_{k=1}^{m} n_Q^k$$

la borne supérieure du calcul de la transformation de P en P^* est de $O(m \cdot (n_P + H))$. En utilisant la procédure récursive de la triangulation du polygone simple sans trou (voir page 76), la triangulation de P^* est de $O((n_P + H)\log(n_P + H))$. Le coût total est inférieur à $O((n_P + H)(\log(n_P+H)+m))$, alors l'algorithme est en moyenne inférieur à $O((n_P+H)(\log(n_P+H)+m))$ qui est à-peu-près de même ordre de grandeur que la méthode existante ($O(n_P \cdot log(n_P) + m \cdot log(n_P) + H \cdot log(H)$)(voir [27]).

Commentaire sur la méthode de décomposition

La méthode de décomposition que nous proposons possède les avantages suivants :

- l'algorithme n'ajoute pas de points pour reconstruire les triangles.
- l'algorithme est très général et fonctionne dans tous les cas.
- l'algorithme est très efficace en terme de temps de calcul ($O((n_p+H)(\log(n_p+H)+m))$).

En contrepartie de ces avantages, on peut reprocher à cet algorithme de produire des triangles qui ne soient pas nécessairement proches de triangles équilatéraux, ce qui est aussi le cas des autres méthodes (voir [25] [19]).

4.6 Détermination des positions des facettes de $S(V_1)$par rapport à $S(V_2)$

Après la décomposition des facettes de $S(V_1)$ et de $S(V_2)$ (voir section précédente), les triangles de $S(V_1)$ (ou de $S(V_2)$) peuvent être divisés en deux ensembles :

1. l'ensemble Ω_1^0 des triangles de $S(V_1)$ qui n'ont pas été décomposés; en d'autres termes, Ω_1^0 est l'ensemble des triangles de $S(V_1)$ qui ne sont pas intersectés par $S(V_2)$.
2. l'ensemble Ω_1^1 des triangles de $S(V_1)$ qui ont été décomposés en sous-triangles; en d'autres termes, Ω_1^1 est l'ensemble des triangles de $S(V_1)$ qui sont intersectés par $S(V_2)$.
3. l'ensemble Ω_2^0 des triangles de $S(V_2)$ qui n'ont pas été décomposés; en d'autres termes, Ω_2^0 est l'ensemble des triangles de $S(V_2)$ qui ne sont pas intersectés par $S(V_1)$.
4. l'ensemble Ω_2^1 des triangles de $S(V_2)$ qui ont été décomposés en sous-triangles; en d'autres termes, Ω_2^1 est l'ensemble des triangles de $S(V_2)$ qui sont intersectés par $S(V_1)$.

Le problème posé consiste à déterminer la position relative de chaque triangle de Ω_1^0 et de Ω_1^1 (resp de Ω_2^0 et de Ω_2^1) par rapport à $S(V_2)$ (resp $S(V_1)$).

4.6.1 Détermination de la position d'un point par rapport à $S(V_2)$

Soit T_1^0 un triangle de Ω_1^0. Puisque T_1^0 n'est pas intersecté par $S(V_2)$, la position d'un point quelconque appartenant à T_1^0 est identique à celle de T_1^0 (à l'intérieur ou à l'extérieur de $S(V_2)$). Suivant que les facettes de $S(V_2)$ sont orientées ou non, nous proposons deux algorithmes pour la détermination de la position d'un point par rapport à une T-surface fermée $S(V_2)$.

Algorithme 1

Soit M un point quelconque en 3D dont on veut déterminer la position relativement à une T-surface fermée $S(V)$. Le principe de la méthode proposée consiste à :

- lancer une demi-droite Δ à partir de M dans la direction positive de l'axe $o\vec{y}$.
- compter le nombre $\nu(M,S)$ d'intersections de Δ avec $S(V)$.

Si Δ ne passe pas par les arêtes ni les sommets de $S(V)$, on vérifie facilement que:

- si $\nu(M,S)$ est pair ou nul, alors M est à l'extérieur de $S(V)$.
- si $\nu(M,S)$ est impair, alors M est à l'intérieur de $S(V)$.

La détermination de l'intersection d'une demi-droite (ou un segment fini) avec une T-surface fermée a été présentée au chapitre 1.2 (cf page 33) et le seul problème restant à résoudre est celui de l'incrémentation de $\nu(M,V)$ lorsque Δ traverse une arête ou un sommet d'un triangle $T \in S(V)$:

Cas où Δ passe par une arête de $S(V)$

Selon la définition d'une T-surface et compte tenu du fait que $S(V)$ est fermée, on est sûr que l'arête ab du triangle $T \in S(V)$ est aussi l'arête d'un autre triangle (et seulement un) $T^* \in S(V)$. Ceci nous permet de définir les segments $\alpha\beta = T \cap P$ et $\beta\gamma = T^* \cap P$ où P est le plan horizontal contenant Δ. Suivant la position relative de $\alpha\beta$ et $\beta\gamma$ par rapport à Δ, les quatre cas suivants doivent être pris en considération :

1. (cf Fig 4.20(a)) si $\alpha\beta$ et $\beta\gamma$ se trouvent de part et d'autre de Δ, alors le nombre d'intersections est égal à 1 pour T et T^* (cette façon de procéder est analogue à celle décrite page 38 pour le test de l'appartenance d'un point à un polygone).
2. (cf Fig 4.20(b)) si $\alpha\beta$ et $\beta\gamma$ se trouvent du même côté de Δ, alors le nombre d'intersections est égal à 0 ou 2 pour T et T^* .
3. (cf Fig 4.20(c)) si $\alpha\beta$ appartient à Δ, alors la facette T est ignorée. Le nombre d'intersections est égal à 2 pour T et T^* .
4. (cf Fig 4.20(d et e)) si $\alpha\beta$ et $\beta\gamma$ coincident avec ab, c'est-à-dire si l'arête ab est horizontale, alors, suivant la position relative de T et de T^* par rapport à P, deux sous-cas doivent être considérés :
   - si T et T^* sont au-dessus (ou au dessous) de P, alors le nombre d'intersections est égal à 0 (T et T^* sont ignorées).
   - si T et T^* sont de part et d'autre de P, alors le nombre d'intersections est égal à 1 sauf si ab appartient à Δ, auquel cas, T et T^* doivent être ignorés (cf Fig 4.20 (f)).

Dans les trois premiers cas ci-dessus, l'incrémentation du nombre d'intersections se fera comme suit:

- si l'extrémité α a une ordonnée supérieure à celle de Δ, alors on incrémente le nombre d'intersections de 1.
- sinon, le nombre d'intersections reste constant.

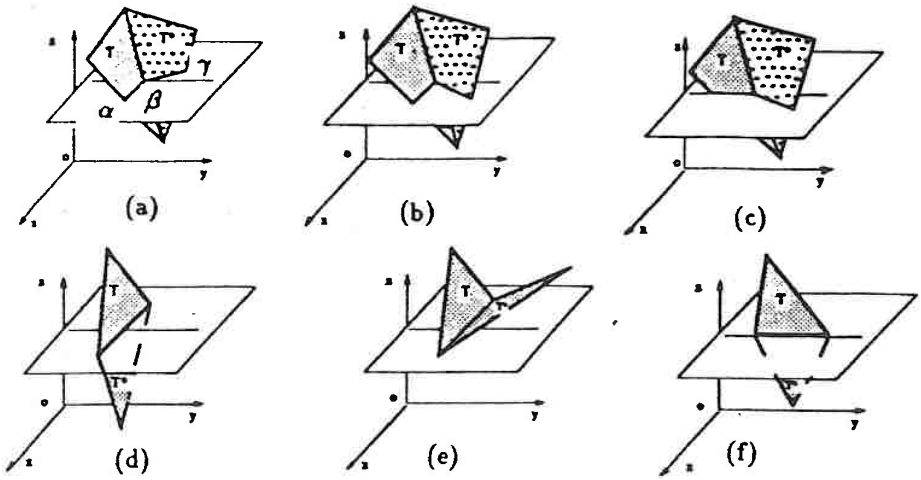


Figure 4.20 *Cas où Δ passe par une arête ab partagée par T et T*.*

Cas où Δ passe par un sommet de $S(V)$

Supposons que Δ passe par un sommet p_i de $S(V)$ et soit :

- P le plan horizontal passant par p_i,
- $\mathcal{T}(p_i) = \{T_1, T_2, \ldots, T_m\}$ l'ensemble des facettes triangulaires ayant p_i comme sommet commun,
- $\mathcal{L}(p_i) = \{p_iq_1, p_iq_2, ..., p_iq_m\}$ l'ensemble des intersections des triangles de $\mathcal{T}(p_i)$ avec P.

Comme l'indique la figure 4.21, supposons que l'on déplace un point z dans le plan P autour du point p_i dans le sens des aiguilles d'une montre. Lorsqu'on croise un segment p_iq_j de $\mathcal{L}(p_i)$, la position de z par rapport à $S(V)$ change. Si l'on note la position de z au départ comme "+", alors la position courante de z, au cours de ce déplacement, est $signe((-1)^k)$ où k est le nombre de segments de $\mathcal{L}(p_i)$ rencontrés depuis la position de départ. En supposant que, au départ, z est sur la droite Δ, on en conclut que le nombre k de segments de $\mathcal{L}(p_i)$ situés au-dessus de Δ peut être considéré comme le nombre d'intersections de Δ avec tous les triangles de $\mathcal{T}(p_i)$.

```
BOOLEAN_t Position_of_Point1(M, S(V))
{
   Soit Δ la demi-droite parallèle à oy et passant par M ;
   Soit ν(M,Δ) le nombre des intersections de Δ avec S(V) ;

   ν(M,Δ) = 0 ;

   pour-tout( triangle T ∈ S(V))
   {
```

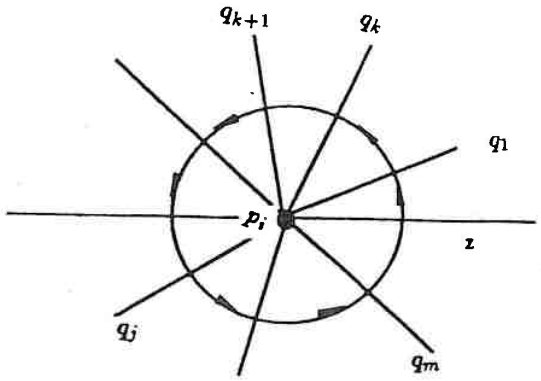


Figure 4.21 *Cas où Δ passe par un sommet p_i de $S(V)$.*

```
        if( Δ passe par l'intérieur de T)
            ν(M,Δ) = ν(M,Δ) + 1 ;
        else if( Δ passe par un côté de T)
            ν(M,Δ) est incrémenté 1 ou 2 ;
        else if( Δ passe par un sommet p_i de T)
            ν(M,Δ) est incrémenté de k (≥0)
            suivant la configuration des triangles autour de p_i;
    }

    if( ν(M,Δ) est pair ou égal à 0)
        return( à l'extérieur) ;
    else
    return( à l'intérieur) ;
}
```

Algorithme 2

Soit M un point quelconque en 3D dont on veut déterminer la position relativement à une T-surface fermée $S(V)$ composée de triangles $T(a_0, a_1, a_2)$ ayant les sommets a_0, a_1 et a_2 ordonnés de telle façon que :

$$\vec{a_0a_1} \wedge \vec{a_1a_2} = \text{ vecteur orienté vers l'extérieur de } S(V)$$

Comme le montre la Figure 4.22, le principe de la méthode proposée consiste à :

- lancer une demi-droite Δ à partir de M et orientée dans une direction quelconque, par exemple la direction positive de l'axe $o\vec{x}$.
- déterminer le premier point d'intersection O de Δ avec $S(V)$ et le triangle $T(a_0, a_1, a_2)$ s'ils existent.

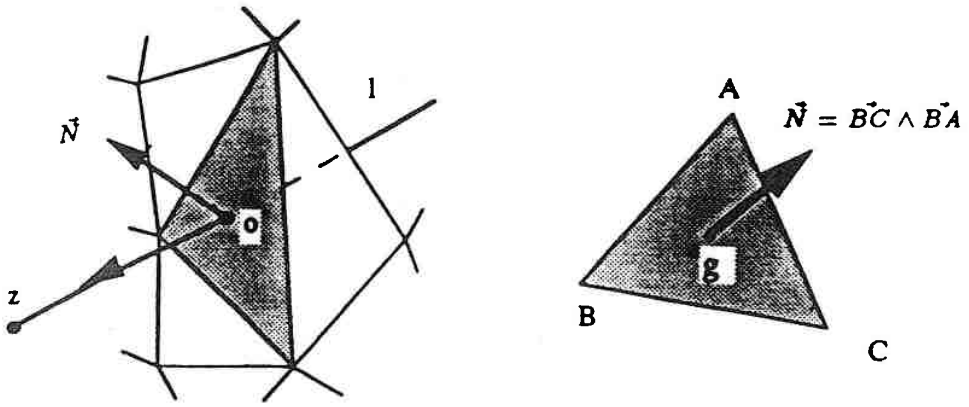


Figure 4.22 *(a) La demi-droite intersecte la T-surface au point o; (b) un triangle est supposé orienté par l'ordre de ses 3 sommets.*

- examiner le produit scalaire de $\vec{MO}$ avec la normale $\vec{N}$ au triangle $T(a_0, a_1, a_2)$

Si Δ n'intersecte pas $S(V)$, alors M est situé à l'extérieur de $S(V)$; dans le cas contraire, plusieurs sous-cas doivent être considérés :

1. si Δ ne passe ni par les arêtes ni par les sommets de $T(a_0, a_1, a_2)$, alors la position de M par rapport à $S(V)$ peut être déterminée par le signe du produit scalaire $(\vec{OM} \wedge \vec{N})$:
   - M est à l'intérieur de $S(V)$, si $\vec{OM} \wedge \vec{N}$ est positif.
   - M est à l'extérieur de $S(V)$, si $\vec{OM} \wedge \vec{N}$ est négatif.
   - M est sur $S(V)$, si $\vec{OM} \wedge \vec{N}$ est nul.
2. si Δ passe par une arête ou un sommet de $T(a_0, a_1, a_2)$, alors on teste si le segment MG joignant M au barycentre G de $T(a_0, a_1, a_2))$ intersecte une facette de $S(V)$ différente de $T(a_0, a_1, a_2)$:
   - si MG n'intersecte aucune autre facette de $S(V)$, alors la position de M peut être déterminée comme dans le cas 1 ci-dessus.
   - si MG intersecte $S(V)$, désignons par $T^* \neq T$ le premier triangle intersecté par MG. On peut alors reprendre itérativement le processus de détermination au cas numéro 1 ci-dessus.

L'algorithme ainsi proposé peut être décrit à l'aide de la fonction suivante décrite en pseudo-C :

```
BOOLEAN_t Position_of_Point2(M, S(V))
{
   Soit Δ la demi-droite parallèle à ox⃗ et passant par M ;
      Soit O le premier point d'intersection de Δ avec un triangle T ∈ S(V);


   while( T ≠ φ )
   {
      if( O est à l'intérieur de T)
      {
         N⃗ = a0a1⃗ ∧ a1a2⃗ ;

      if( N⃗ · OM⃗ > 0.0)
         return( à l'intérieur) ;
      else
         reurn( à l'extérieur) ;
      }

      if( O est sur un côté ou coincide avec un sommet de T)
      {
         soit G le barycentre de T ;

         if( MG n'intersecte aucune facette de S(V))
         {
            N⃗ = a0a1⃗ ∧ a1a2⃗ ;

            if( N⃗ · OM⃗ > 0.0)
               return( à l'intérieur) ;
            else
               reurn( à l'extérieur) ;
         }
         else
         {
            soit T* le premier triangle intersecté par MG ;
            on pose: O = T* ∩ MG ;
            T = T* ;
         }
      }
   }
   return( à l'extérieur) ;
}
```

Remarque

L'algorithme 2 nécessite l'orientation de la T-surface $S(V)$ (voir Fig 4.23(a)). Dans les cas où $S(V)$ n'est pas orientée, cette méthode est tout de même envisageable car il suffit d'abord d'orienter $S(V)$ (voir page 101). L'algorithme 2 a pour avantage d'éviter le cas où Δ passe par les sommets ou les arêtes de $S(V_2)$, il est alors plus simple à mettre en oeuvre (En langage C).

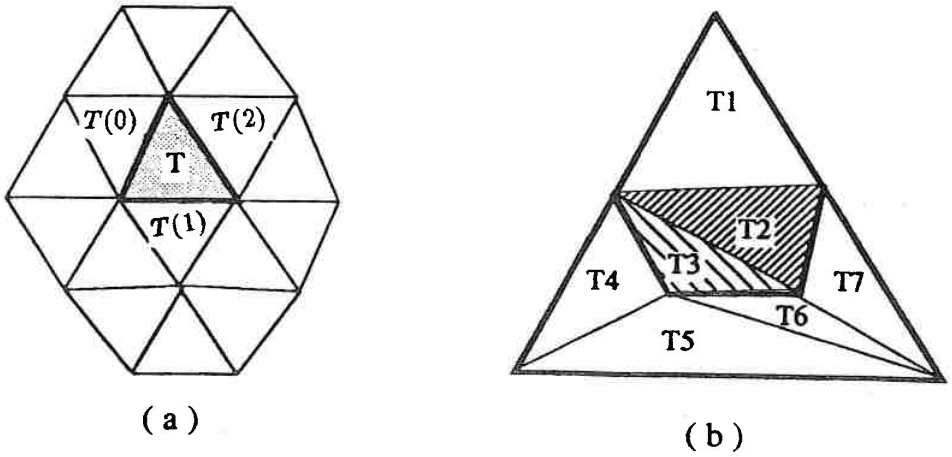


Figure 4.23 *(a) les triangles de Ω_1^0 liés avec T; (b) T a été décomposé.*

4.6.2 Détermination des positions des facettes de Ω_1^0 et de Ω_1^1 par rapport à $S(V_2)$

Position des facettes de Ω_1^0

Soient $T \in \Omega_1^0$ et $T(0)$, $T(1)$ et $T(2)$ ses trois triangles adjacents. Il est évident que si $T(0)$, $T(1)$ et $T(2)$ appartiennent à Ω_1^0, alors leurs positions par rapport à $S(V_2)$ sont identiques à celle de T; on en déduit que la position de toutes les facettes de Ω_1^0 qui sont liées avec T peut être déterminée à l'aide d'une procédure récursive.

Si l'on applique cette méthode, alors la position de toutes les facettes de Ω_1^0 peut être déterminée très rapidement. En remarquant que les sommets des facettes de Ω_1^0 ont la même position que les facettes elles-mêmes, on en déduit également la position de ces sommets (cf Fig 4.23(a)).

Position des facettes de Ω_1^1

Comme l'indique la Figure 4.23(b), les facettes de Ω_1^1 peuvent être classées en deux types :

1. celles qui ont au moins un sommet initial de $S(V_1)$.
2. celles qui n'ont aucun sommet initial de $S(V_1)$. Leurs sommets ont été crées au cours du processus de décomposition présenté précédement.

Soit T_1^1 une facette de type 1 et T_1^2 de type 2. On peut constater que T_1^1 possède la même position relative que ses sommets initiaux par rapport à $S(V_2)$ dont la position a déjà été déterminée au paragraphe précédent. Dans le cas de T_1^2, les sommets n'ont pas de position connue et nous proposons de déterminer cette position en calculant la position du barycentre de T_1^2.

Algorithme

void **Determining_Position_of_Trgls(** $S(V_1), S(V_2)$**)**
{
pour-tout(triangle $T \in S(V_1)$**)**
{
if(T **est un triangle initial && sa position n'est pas determinée**
{
- soit g le barycenter de T ;
- poser:
  - position de T = Position_of_Point1(g, $S(V_2)$) ;
- pour tout triangle T^* lié directement ou indirectement à T, poser :
  - position de T^* = position de T;

}
}

pour-tout(triangle $T \in S(V_1)$**)**
{
if(T **est un triangle non initial)**
{
if(T **possède un sommet initial)**
son signe est identique à celui de son sommet initial ;
else
{
poser:

position de T **= Position_of_Point1(**g, $S(V_1)$**) ;**
}
}
}
return ;
}

4.7 Complexité et résultats expérimentaux

Analyse de complexité

Soient V_1 et V_2 deux T-solides auxquels on se propose d'appliquer des opérations booléennes. Pour ce faire, on construit tout d'abord un T-octree pour chacune des deux T-surfaces associées et le calcul de ces opérations se déroule ensuite en trois étapes principales:

1. Calcul de $T_1 \cap S(V_2)$ pour tout $T_1 \in S(V_1)$ et le calcul de $T_2 \cap S(V_1)$ pour tout $T_2 \in S(V_2)$.
2. Subdivision en triangles de tout $T_1 \in \Omega_1^1$ et de tout $T_2 \in \Omega_2^1$.
3. Détermination de la position de tout triangle de $S(V_1)$ (resp de $S(V_2)$) par rapport à $S(V_2)$ (resp $S(V_1)$).

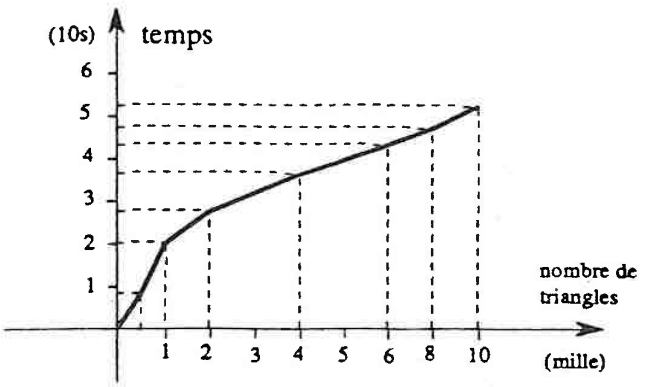


Figure 4.24 *Temps de calcul mesuré pour les opérations booléennes sur deux sphères.*

Soient N_1 *(resp* N_2*)* le nombre de triangles de $S(V_1)$ (resp $S(V_2)$) et M_1 (resp M_2) le nombre de triangles de $S(V_1)$ (resp $S(V_2)$) intersectés par $S(V_2)$ (resp $S(V_1)$). Afin de faciliter la discussion, on suppose que $N_1 \approx N_2$. Avec ces notations, les trois étapes mentionnées ci-dessus peuvent être analysées de la façon suivante :

- ETAPE 1 : le calcul de $T_1 \cap S(V_2)$ est composé de deux sous-étapes:
  - Parcourir le T-octree de $S(V_2)$. Ceci coûte $O(N_1)$
  - Tester l'intersection de T_1 avec les triangles de $S(V_2)$. Ceci coûte également $O(N_1)$.
- ETAPE 2 : le coût de cette étape est à priori $O(M_1+M_2)$. En remarquant que, en moyenne, $M_1 + M_2 = \sqrt{N_1 + N_2}$, on en déduit que le coût de l'étape 2 est $O(\sqrt{N_1 + N_2})$.
- ETAPE 3 : la détermination de la position des triangles initiaux de $S(V_1)$ (resp $S(V_2)$) par rapport à $S(V_2)$ (resp $S(V_1)$) est faite par une procédure récursive très rapide. Dans cette procédure, le calcul principal est la détermination de la position des triangles non initiaux et le coût correspondant est en $O(M_1 + M_2)$.

On peut constater que, aux étapes 2 et 3, l'algorithme est en $O(\sqrt{M_1 + M_2})$ et à l'étape 1 en $O(N_1 + N_2)$. En pratique, les calculs aux étape 1 et 3 sont très petits par rapport à celui de l'étape 2, donc, on peut dire que, en moyenne, notre algorithme est en $O(\sqrt{N_1 + N_2})$ à condition que les T-octrees soient déja construits. Dans le pire cas où tous les triangles de $S(V_1)$ (resp $S(V_2)$) sont intersectés ($M_1 = N_1$ *et* $M_2 = N_2$), l'algorithme fonctionne en $O(M_1 + M_2) = O(N_1 + N_2)$ (linéaire).

En pratique, le volume de calcul à l'étape 1 est très petit par rapport à celui de l'étape 2, et l'on peut donc dire que :

- En moyenne, notre algorithme est en $O(\sqrt{N_1 + N_2})$ à condition que les T-octrees soient déjà construits.

- Dans le pire cas où tous les triangles de $S(V_1)$ (resp $S(V_2)$) sont intersectés ($M_1 = N_1$ et $M_2 = N_2$), l'algorithme fonctionne en $O(M_1 + M_2) = O(N_1 + N_2)$ et peut donc être considéré comme linéaire.

La figure ci-dessous illustre la performance de l'algorithme sur une machine fonctionnant à la vitesse de 7 Mips.

4.8 Quelques extentions

4.8.1 Opérations booléennes sur les P-solides

Introduction

Comme nous l'avons présenté dans l'introduction de ce chapitre, la résolution pratique des opérations booléennes des solides définis par leurs bords (P-solides) n'est pas une chose facile si l'on souhaite traiter correctement les nombreux cas qui peuvent réellement se présenter en pratique.

En s'inspirant des recherches effectuées sur les T-solides, on peut imaginer que la mise en oeuvre des opérations sur les P-solides (voir page 17) doit être semblable à. En fait, en construisant des P-octree et en subdivisant certains polygones, il est tout à fait possible d'effectuer les opérations booléennes sur des P-solides de la même façon que pour les T-solides.

Dans ce qui suit, nous proposons une autre approche du problème consistant à transformer **virtuellement** les P-solides en T-solides. Grâce aux études faites sur les T-solides, cette méthode permet de traiter correctement tous les cas particuliers. Les solides résultants **ne sont pas** des T-solides mais des P-solides comme ceux obtenus par les méthodes classiques.

Algorithme général

Soient V_1 et V_2 deux P-solides. Dans ce chapitre, on a démontré qu'un polygone simple quelconque pouvait être subdivisé en triangles (voir page 75), donc les peaux $S(V_1)$ et $S(V_2)$ peuvent être **virtuellement** transformées en T-surfaces fermées en divisant virtuellement toutes les facettes de $S(V_1)$ et de $S(V_2)$ en triangles.

Comme pour les T-solides, la mise en oeuvre des opérations booléennes sur les P-solides est effectuée également en trois étapes :

1. ETAPE 1 : Pour chaque facette polygonale P_1 de $S(V_1)$, on détermine l'ensemble $\lambda(P_1, S(V_2))$ des lignes polygonales d'intersection de P_1 avec $S(V_2)$. De la même façon, on détermine l'intersection de chaque facette de $S(V_2)$ avec $S(V_1)$.

2. ETAPE 2 : On subdivise chaque facette P_1 de V_1 par les lignes polygonales de $\lambda(P_1, S(V_2))$ obtenue à l'étape 1 (si $\lambda(P_1, S(V_2))$ existe); à l'issue de cette subdivision, les facettes P_1 concernées sont remplacées par des sous facettes polygonales. On opère de façon symétrique pour chaque facette de $S(V_2)$.

3. ETAPE 3 : Une fois l'étape 2 réalisée, on détermine la position relative de chaque facette de $S(V_1)$ (resp de $S(V_2)$) par rapport à $S(V_2)$ (resp $S(V_1)$) et on effectue les assemblages des facettes afin d'obtenir $S(V_1 \cap V_2)$ $S(V_1 \cup V_2)$ etc...

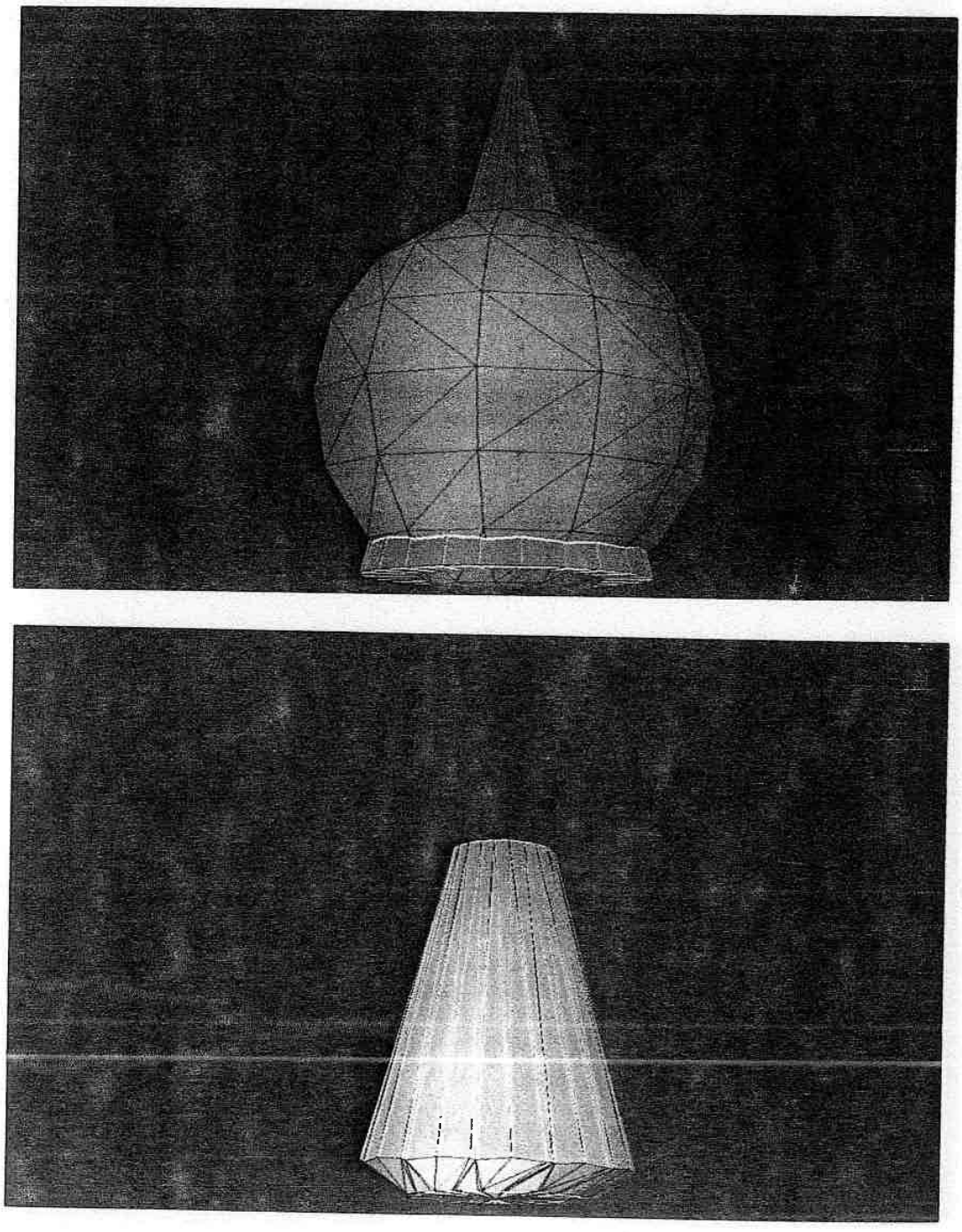


Figure 4.24 *Intersection d'une cone avec une sphére.*

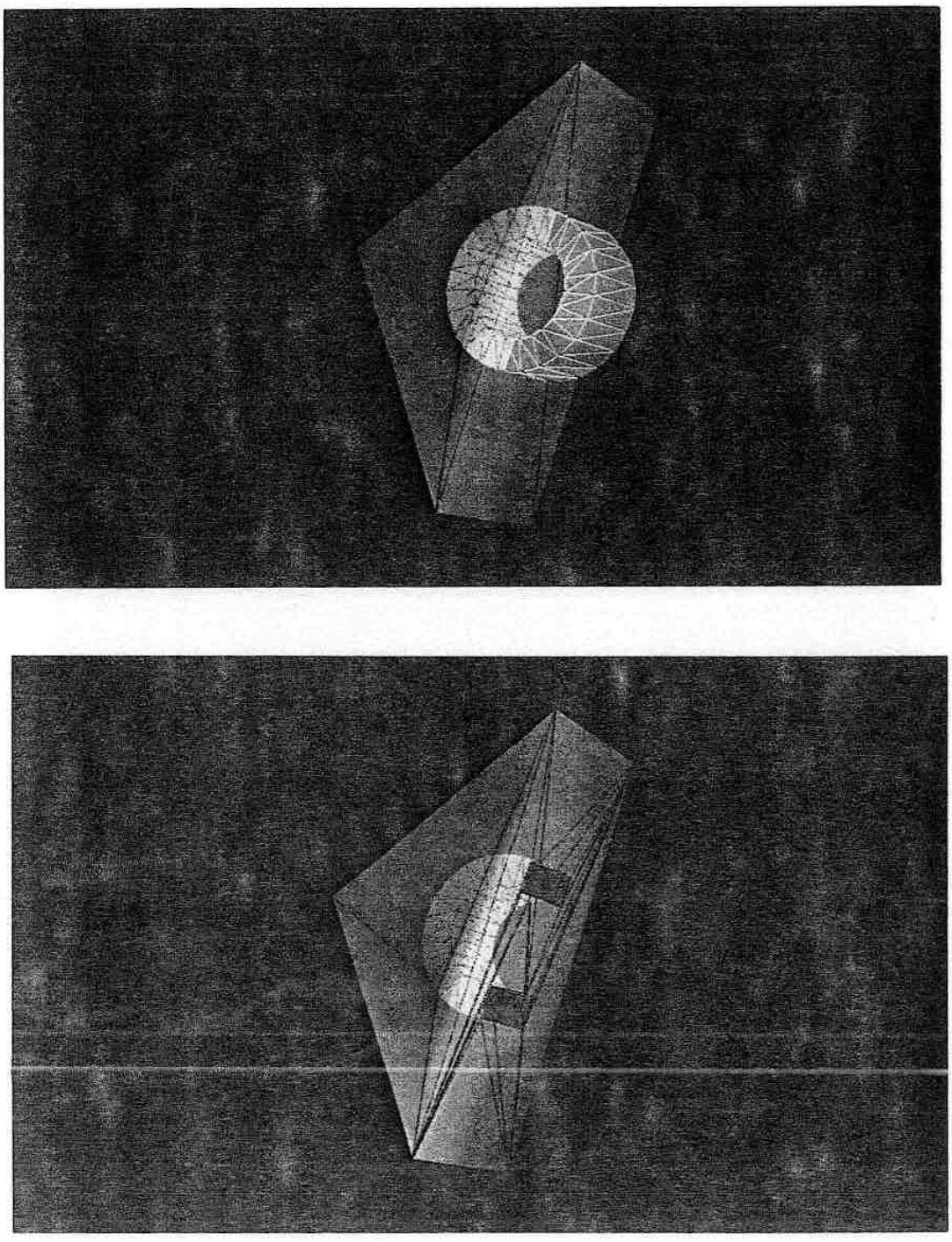


Figure 4.25 *Différence d'un prisme avec un tore.*

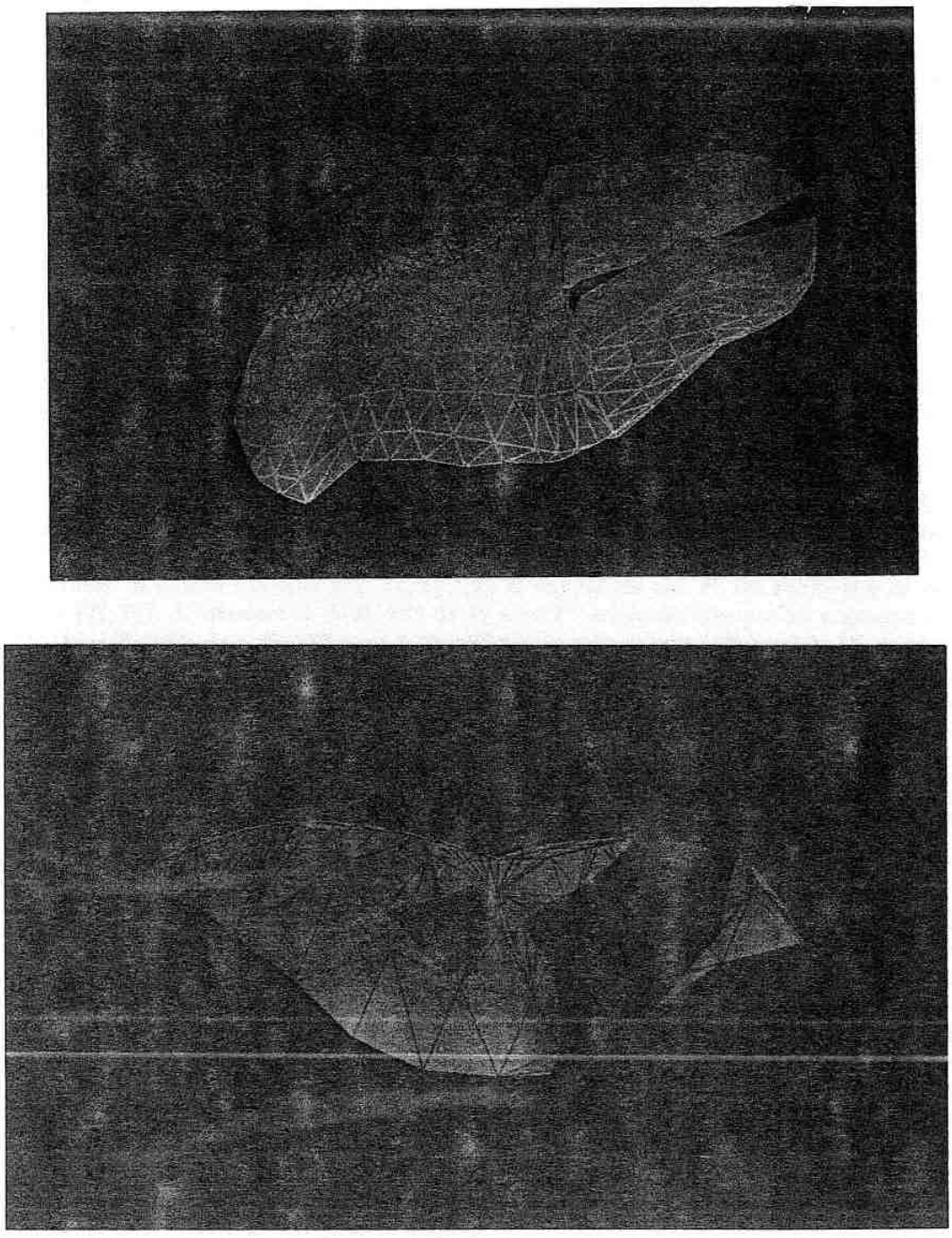


Figure 4.26 *Intersection de deux gisements.*

Recherche de $\lambda(P_1, S(V_2))$[7]

Soit $\mathcal{T}(P_1)$ l'ensemble des triangles **virtuels** remplissant une facette P_1 de $S(V_1)$ et soit $\lambda(P_1, S(V_2))$ les lignes polygonales d'intersection de P_1 avec $S(V_2)$. Si $\lambda(P_1, S(V_2))$ n'est pas vide, alors il est constitué par l'ensemble des intersections de P_1 avec toutes les facettes polygonales de $S(V_2)$; les lignes polygonales de $\lambda(P_1, S(V_2))$ sont alors constituées par une série de segments dont les extrémités sont :

- soit l'intersection de P_1 avec les arêtes de $S(V_2)$.
- soit l'intersection des arêtes de P_1 avec les facettes de $S(V_2)$.

Soit $\mathcal{T}(P_1)$ l'ensemble des triangles virtuels décomposant la facette polygonale P_1. On peut constater que :

- $\lambda(P_1, S(V_2))$ est géométriquement identique à $\lambda(\mathcal{T}(P_1), S(V_2))$, pourtant ce dernier peut posséder plus de segments (plus d'extrémités) que le premier.
- les extrémités de $\lambda(P_1, S(V_2))$ sont aussi celles de $\lambda(\mathcal{T}(P_1), S(V_2))$.

On en déduit que la recherche de $\lambda(\mathcal{T}(P_1), S(V_2))$ consiste à regrouper les segments de $\lambda(\mathcal{T}(P_1), S(V_2))$ afin d'éliminer les sommets de $\lambda(P_1, S(V_2))$ qui appartiennent à des côtés virtuels des triangles.

Comme l'indique la figure 4.28, soit P^* une facette polygonale de $S(V_2)$ qui est subdivisée virtuellement en quatre triangles $\{T_1^*, T_2^*, T_3^*, T_4^*\}$ et supposons que P_1 est subdivisée en deux triangles $\{T_1, T_2\}$. L'intersection $L(P_1, P^*)$ de P_1 avec P^* est constituée par les segments $ab = T_1 \cap T_1^*$, $bc = T_1 \cap T_2^*$, $de = T_1 \cap T_4^*$ et $ef = T_2 \cap T_4^*$ (ces segments sont colineaires). Les segments de $L(P_1, P^*)$ peuvent être obtenus par la méthode présentée au chapitre 1.2 (cf page 33) en utilisant les T-octrees. Une fois ces segments obtenus, il suffit de combiner deux à deux ceux qui ont une extrémité commune. Les segments ab et bc (resp de et ef) sont regroupés en un seul segment ac (resp df). La combination résultante est :

$$L(P_1, P^*) = \{ac, df\} = \{ab, bc, de, ef\}$$

Décomposition de P_1 par $\lambda(P_1, S(V_2))$ en sous-polygones

Par analogie avec l'étude effectuée au paragraphe 4.4.3 et comme le suggère la Figure 4.29, on peut identifier trois types élémentaires de lignes polygonales $\lambda(P_1, S(V_2))$. En plus des trois types élémentaires présentés sur la Figure 4.29, on peut rencontrer d'autres types de configurations pour les lignes $\lambda(P_1, S(V_2))$ mais ceux ci peuvent toujours être convertis en l'un des trois types élémentaires. En pratique, les cas réels rencontrés dans les applications correspondent souvent à une combinaison des trois types élémentaires (voir page 65). On peut toujours considérer que les lignes polygonales de $\lambda(P_1, S(V_2))$ ne se croisent pas mais possèdent, éventuellement, des sommets communs. En utilisant la même méthode que celle présentée au paragraphe 4.4.4 pour la partition d'un triangle en régions polygonales, on peut toujours partitionner P_1 en régions polygonales par les lignes de $\lambda(P_1, S(V_2))$; les régions polygonales ainsi obtenues peuvent, éventuellement, posséder des trous polygonaux.

La figure 4.30 illustre un exemple de division.

[7] Cette recherche nécessite de calculer l'intersection de deux facettes polygonale (cf Chapitre 1.3)

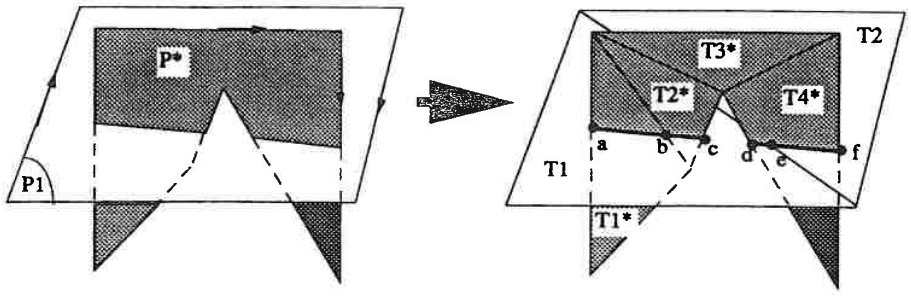


Figure 4.28 *Intersection de deux polygones P^* et P_1: $P_1 \cap P^* = \{ab, bc, de, ef\} = \{ac, df\}$*

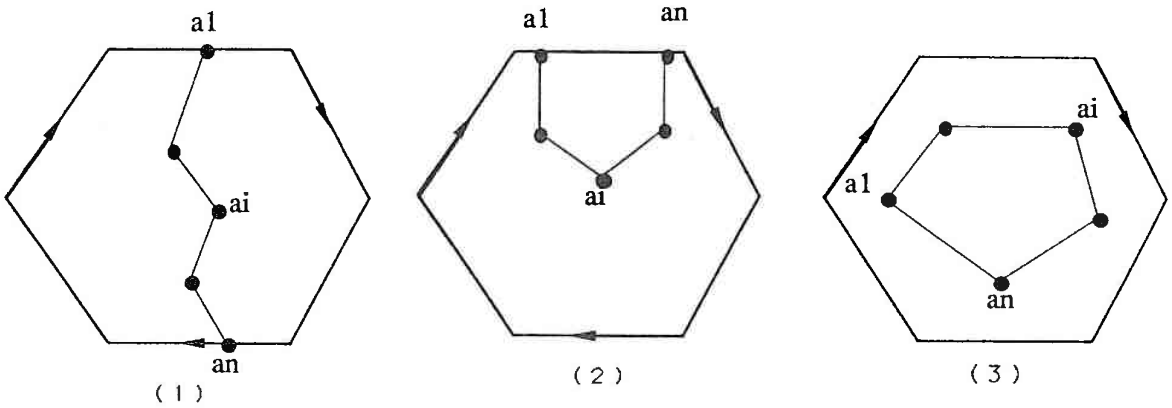


Figure 4.29 *Trois types de lignes d'intersection élémentaires.*

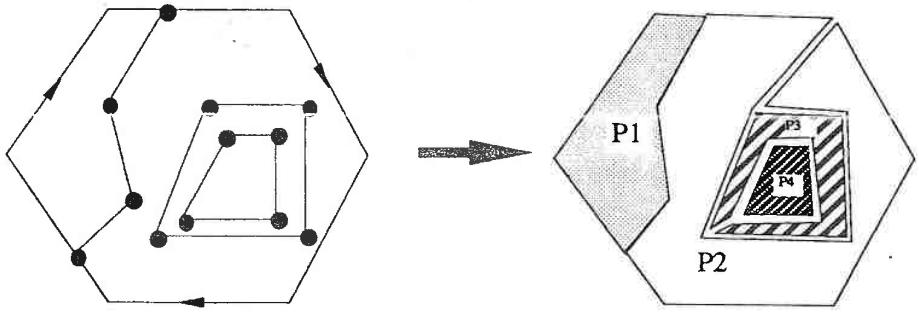


Figure 4.30 *Un exemple de la partition d'un polygone simple.*

Détermination de la position de chaque facette de $S(V_1)$ (resp $S(V_2)$ par rapport à $S(V_2)$ (resp $S(V_1)$)

En utilisant la même méthode que celle présentée au paragraphe 4.6, on peut déterminer la position par rapport à $S(V_2)$ de chaque triangle **virtuel** de $S(V_1)$ qui n'a pas été intersecté par $S(V_2)$. En fait, les facettes de $S(V_1)$ peuvent être classées en deux types :

- les facettes initiales (possédant 3 sommets initiaux) de $S(V_1)$, c'est-à-dire, celles qui n'ont pas été intersectées par $S(V_2)$ (l'ensemble de ces facettes est noté Ω_1^0)
- les facettes créees pendant la subdivision (l'ensemble de ces facettes est noté Ω_1^1).

Il est évident qu'une facette polygonale de Ω_1^0 possède la même position par rapport à $S(V_2)$ que l'un de ses triangles. Les facettes de Ω_1^1 peuvent se séparer en deux groupes :

- les facettes qui ont au moins un sommet initial de $S(V_1)$. Dans ce cas, la facette possède la même position que ses sommets.
- les facettes qui n'ont aucun sommet initial et dont tous les sommets ont été créés au cours de la partition. Dans ce cas, la détermination de la position d'une facette de ce groupe ne peut alors être effectuée que par identification de la position de l'un quelconque de ses points, par exemple le barycentre.

Algorithme

void **Set_Operations_of_Psolid**($S(V_1), S(V_2), S(V_1 in V_2), S(V_1 out V_1),\ S(V_2 in V_1), S(V_2 out V_2)$)
{
Division de chaque facette de $S(V_1)$ en triangles virtuels;
Division de chaque facette de $S(V_2)$ en triangles virtuels;

pour-tout(polygone $P \in S(V_1)$)
{
Calcul de $\lambda(P, S(V_2))$ à l'aide du T-octree de $S(V_2)$;
Division de P en sous-polygones par $\lambda(P, S(V_2))$;
}

pour-tout(polygone $P \in S(V_2)$)
{
Calcul de $\lambda(P, S(V_1))$ à l'aide du T-octree de $S(V_1)$;
Division de P en sous-polygones par $\lambda(P, S(V_1))$;
}

Déterminer la position de tout polygone de $S(V_1)$ par rapport à $S(V_2)$;
Déterminer la position de tout polygone de $S(V_2)$ par rapport à $S(V_1)$;
Constitution de $S(V_1 in V_2), S(V_1 out V_2), S(V_2 in V_1) et S(V_2 out V_1)$;

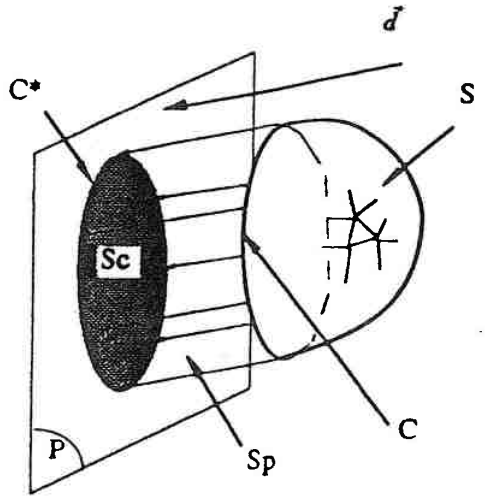


Figure 4.31 *Ferméture de S en direction $\vec{d}$.*

```
    return ;
}
```

4.8.2 Fermeture d'une T-surface ouverte dans une direction $\vec{d}$

Définition

Comme indiqué sur la figure 4.31, soit S une T-surface ouverte ne se recoupant pas et dont la frontière est C, soit $\vec{d}$ un vecteur donné et soit P un plan très éloigné de S et perpendiculaire à $\vec{d}$. Posons :

- S_P^{∞} = cylindre infini de génératrices parallèles à $\vec{d}$ s'appuyant sur le contour C
- S_P = partie de S_P^{∞} comprise entre C et P
- S_C = partie de P entourée par la projection de C suivant la direction $\vec{d}$
- $S^{\vec{d}} = S \cup S_P \cup S_C$

Si les conditions suivantes sont réalisées

$$\left| \begin{array}{l} S \cap (S_P \cup S_C) \equiv C \\ S^{\vec{d}} \text{ est fermée} \end{array} \right.$$

alors nous dirons que $S^{\vec{d}}$ est la fermeture de S dans la direction $\vec{d}$.

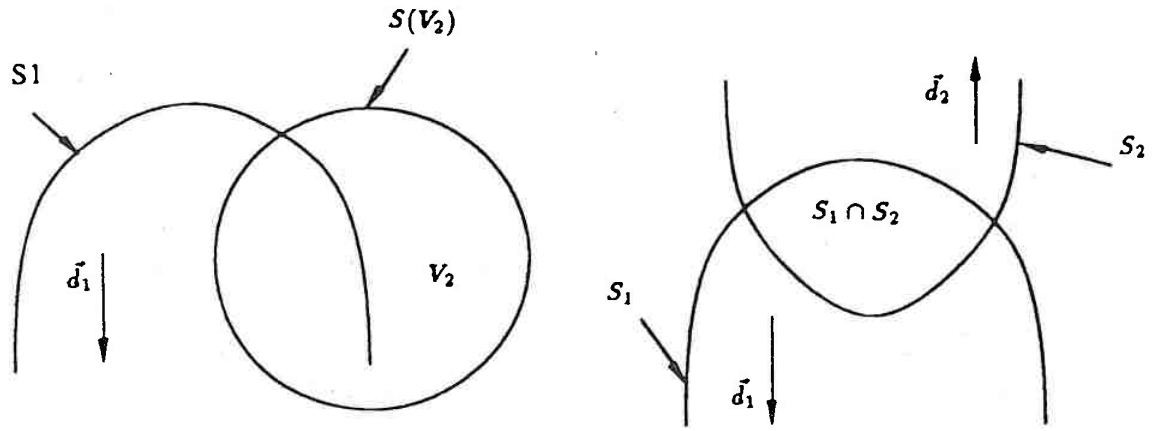


Figure 4.32 *Opérations booléennes sur les fermetures de T-surfaces: (a) $S_1 \cap S_2$ est faisable; (b) $S_1 \cap S_2$ n'est pas faisable.*

Opérations booléennes d'une fermeture d'une T-surface avec un T-solide

Soit $S_1^{\vec{d}}$ la fermeture d'une T-surface ouverte S_1 et soit V_1 le solide associé dont la peau est $S_1^{\vec{d}}$. Compte tenu que les parties de $S_1^{\vec{d}}$ différentes de S_1 ne sont pas définies comme étant composées de facettes triangulaires, il convient de remarquer que :

- $S_1^{\vec{d}}$ n'est pas une T-surface
- V_1 n'est pas un T-solide

Il s'ensuit que, si l'on combine V_1 avec un T-solide V_2 à l'aide d'opérations booléennes, alors, comme le suggère la Figure 4.32, le résultat n'est pas nécessairement un T-solide. Pour garder la cohérence du modèle, si la peau du solide résultant de l'une de ces opérations est constituée (entièrement ou partiellement) par S_C ou S_P, alors cette peau n'est pas une T-surface et nous considèrerons l'opération comme impossible à réaliser (voir figure 4.32(a)).

Dans les cas où ces opérations sont possibles, la difficulté majeure lors de la mise en oeuvre est en fait la détermination de la position de chaque facette de $S(V_2)$ par rapport à S_1, car $S_1^{\vec{d}}$ n'est pas une T-surface fermée. On peut cependant remarquer que si les facettes de S_1 sont orientées, alors la résolution du problème est facile.

Notons que l'on peut généraliser ce qui vient d'être dit au cas ou la peau du solide V_2 est elle même constituée par la fermeture d'une T-surface S_2 (cf Fig 4.32 (b)).

4.9 Calcul du volume d'un T-solide complexe

Soit V un T-solide complexe quelconque dont la peau S est supposée orientée positivement de l'intérieur vers l'extérieur de V. Le principe du calcul du volume $|V|$ de V est basé sur la formule

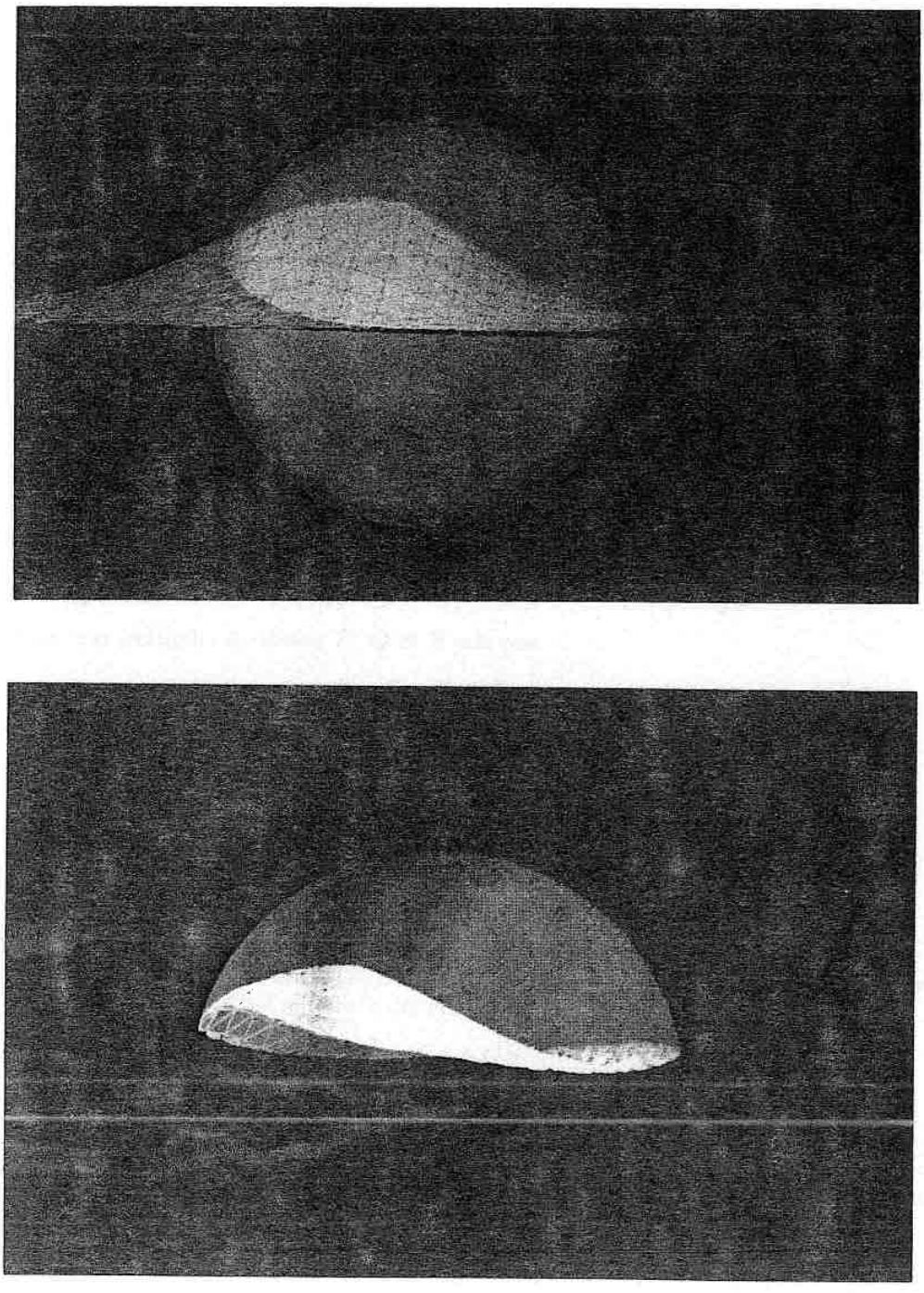


Figure 4.32 *Exemple d'ensemble $S(V_1) - S_2$ lorsque S_2 est une T-surface ouverte.*

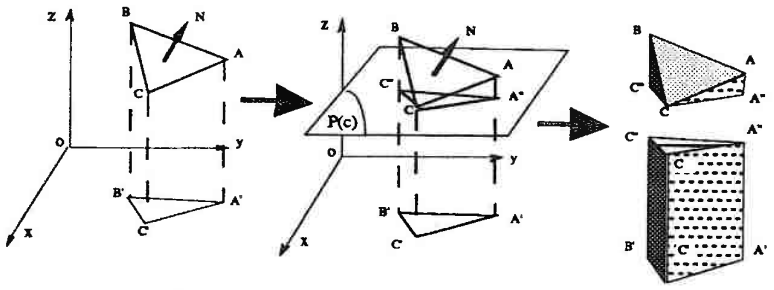


Figure 4.34

suivante où P, Q, R sont trois fonctions de (x, y, z) :

$$\int\int\int_V (\frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y} + \frac{\partial R}{\partial z}) dxdydz = \int\int_{S(V)} Pdxdy + Qdxdz + Rdxdy$$

En effet, on peut remarquer que :

$$\frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y} + \frac{\partial R}{\partial z} = 1 \Longrightarrow \int\int\int_V (\frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y} + \frac{\partial R}{\partial z}) dxdydz = \int\int\int_V dxdydz = |V|$$

il suffit donc, par exemple, de choisir P, Q et R tels que :

$$P = Q = 0 \text{ et } R = z$$

On peut alors calculer le volume $|V|$ à l'aide de la formule suivante :

$$|V| = \int\int_{S(V)} z dxdy = \sum_{T \in S(V)} \int\int_T z dxdy$$

où T désigne une facette triangulaire quelconque de S orientée vers l'extérieur de V.

Calcul de $\int\int_T z dxdy$

Comme l'indique la figure 4.34, soit $T(A, B, C)$ un triangle quelconque de S dont la normale $\vec{N}$ orientée vers l'extérieur de S est supposée définie par :

$$\vec{N} = \vec{CA} \wedge \vec{CB}$$

Désignons par (x_A, y_A, z_A), (x_B, y_B, z_B) et (x_C, y_C, z_C) les coordonnées des trois sommets du triangle.

La valeur absolue de l'intégrale $\int\int_{T(A,B,C)} z dxdy$ correspond au volume $vol(ABC)$ compris entre la projection de $T(A, B, C)$ dans le plan xoy et $T(A, B, C)$. Pour faciliter la discussion, on suppose que le sommet C est celui ayant la coordonnée z la plus petite :

$$z_C = min(z_A, z_B, z_C)$$

Soit $P(C)$ le plan horizontal passant par C. Comme l'illustre la figure 4.34 $T(A'', B'', C)$ est l'intersection de $vol(ABC)$ avec $P(C)$. Il est évident que $P(C)$ divise $vol(ABC)$ en deux parties :

- l'une, située au dessus de $P(C)$, est une pyramide de base quadrangulaire $\square(A, B, B'', A'')$ et de sommet C dont le volume sera noté $vol_1(ABC)$
- l'autre, située au dessous de $P(C)$, est un prisme droit de base triangulaire $\triangle(A"B"C)$ dont le volume sera noté $vol_2(ABC)$

Soient |T(A",B",C)| l'aire du triangle $T(A'', B'', C)$, $|\square(A, B, B'', A'')|$ l'aire du rectangle $\square(A, B, B", A$ et h la distance perpendiculaire entre le point C et le plan contenant $\square(A, B, B'', A'')$. On vérifie facilement que l'on a :

$$\begin{aligned}
vol_1(ABC) &= \frac{|\square ABB"A"|\cdot h}{3} \\
&= \frac{(|BB"|+|AA"|)\cdot|B"A"|}{2} \cdot \frac{h}{3} \\
&= \frac{|BB"|+|AA"|}{2} \cdot |B"A"|.(\frac{h}{3}) \\
&= \frac{|BB"|+|AA"|}{2} \cdot \frac{2\cdot|\triangle A"B"C|}{3} \\
&= \frac{(|BB"|+|AA"|)\cdot|\triangle A"B"C|}{3} \\
vol_2(ABC) &= |CC'| \cdot |\triangle A''B''C|
\end{aligned}$$

On peut donc écrire :

$$\left|\begin{array}{l}
vol(ABC) = vol_1(ABC) + vol_2(ABC) = \frac{1}{3}(|BB''| + |AA''| + 3\cdot|CC'|)\cdot|\triangle A"B"C| \\
\text{avec :} \\
\left\{\begin{array}{ll}
|\triangle A''B''C| & = \frac{|(x_A-x_C)(y_B-y_C) - (x_B-x_C)(y_A-y_C)|}{2} \\
(|BB''| + |AA''| + 3\cdot|CC'|) & = z_A + z_B + z_C
\end{array}\right.
\end{array}\right.$$

$$\Longrightarrow \left|\begin{array}{l}
\int\int_T z\,dx\,dy = \epsilon|\frac{1}{3}(z_A + z_B + z_C)\cdot((x_A - x_C)(y_B - y_C) - (x_B - x_C)(y_A - y_C))| \\
\text{avec : } \epsilon = \left\{\begin{array}{ll} +1 & \text{si } \vec{N}\cdot\vec{oz} > 0 \\ -1 & \text{si } \vec{N}\cdot\vec{oz} < 0 \end{array}\right.
\end{array}\right.$$

Orientation d'une T-surface fermée

Pour orienter une T-surface fermée S il suffit d'orienter chacune de ses facettes T. Si les facettes triangulaires sont codées à l'aide de la structure de données TRGL_t présentée au chapitre 1.1, il est facile d'accéder aux trois facettes adjacentes d'une facette donnée T et, récursivement, à tous les triangles qui ont un lien avec T. L'orientation d'une surface fermée peut donc être réalisée par les deux procédures suivantes décrites en pseudo-C :

```
void Orientating_Tsurf( S )
{
   pour-tout(triangle T ∈ S)
   {
      if( T n'est pas orienté )
      {
         Orientating(T) ;
         Orientating_Trgls_Recursively( T ) ;
      }
   }
   return ;
}

void Orientating_Trgls_Recursively( T )
{
   pour-tout( T* = triangles adjacents à T)
   {
      if(T* ≠ φ &&T* n'est pas encore orienté )
      {
         Orienter T* dans la même direction que T ;
         Orientating_Trgls_Recursively( T* ) ;
      }
   }

}
```

On constate que la première fonction ci-dessus nécessite de savoir orienter un triangle T de S de telle façon que sa normale soit dirigée vers l'extérieur de S. Comme l'indique la figure 4.22(b), soit $T(A,B,C)$ une facette de S de sommets A, B et C. Posons :

$$\vec{N} = \vec{BC} \wedge \vec{BA}$$

Par convention, l'orientation de $T(A,B,C)$ consiste à ordonner les trois sommets de A, B et C de telle façon que $\vec{N}$ soit dirigé vers l'extérieur de S. Soit G le barycentre de $T(A,B,C)$ considéré comme l'origine de $\vec{N}$ et soit G' un point voisin de G appartenant à $\vec{N}$ et tel que S ne soit pas recoupé par le segment GG'. L'orientation de $T(A,B,C)$ peut alors se déduire de la position de G' de la façon suivante :

- l'orientation de $T(A,B,C)$ est définie par l'ordre $\{A,B,C\}$, si G' est à l'intérieur de S.
- l'orientation de $T(A,B,C)$ est définie par l'ordre $\{A,C,B\}$, si G' est à l'extérieur de S.

La méthode de la détermination de la position d'un point par rapport à une T-surface fermée a déja été présentée au paragraphe 4.6.1.

Chapitre

5

Opérations chirurgicales sur les T-surfaces complexes

Introduction

Les opérations sur les T-surfaces, qui vont être présentées dans ce chapitre, sont très différentes des opérations booléennes sur les T-solides présentées au chapitre précédent. Plus précisément, ce sont des opérations semblables à celles effectuées par les chirurgiens sur le corps humain mais les objets à "opérer" sont ici des T-surfaces quelconques définies au chapitre 1.1. En utilisant ces opérations chirurgicales, on peut par exemple :

- couper une T-surface à l'aide d'un couteau ou d'une paire de ciseaux informatiques.
- enlever un morceau d'une T-surface pour trouer la T-surface.
- coller un morceau de T-surface à une T-surface déjà existante.
- etc ...

Si les T-surfaces manipulées représentent la peau d'organes biologiques, alors les opérations présentées dans ce qui suit sont littéralement des opérations "chirurgicales"!.

5.1 Découpage d'une T-surface par une autre T-surface

5.1.1 Position du problème

Considérons tout d'abord les deux exemples représentés sur la Figure 5.1 :

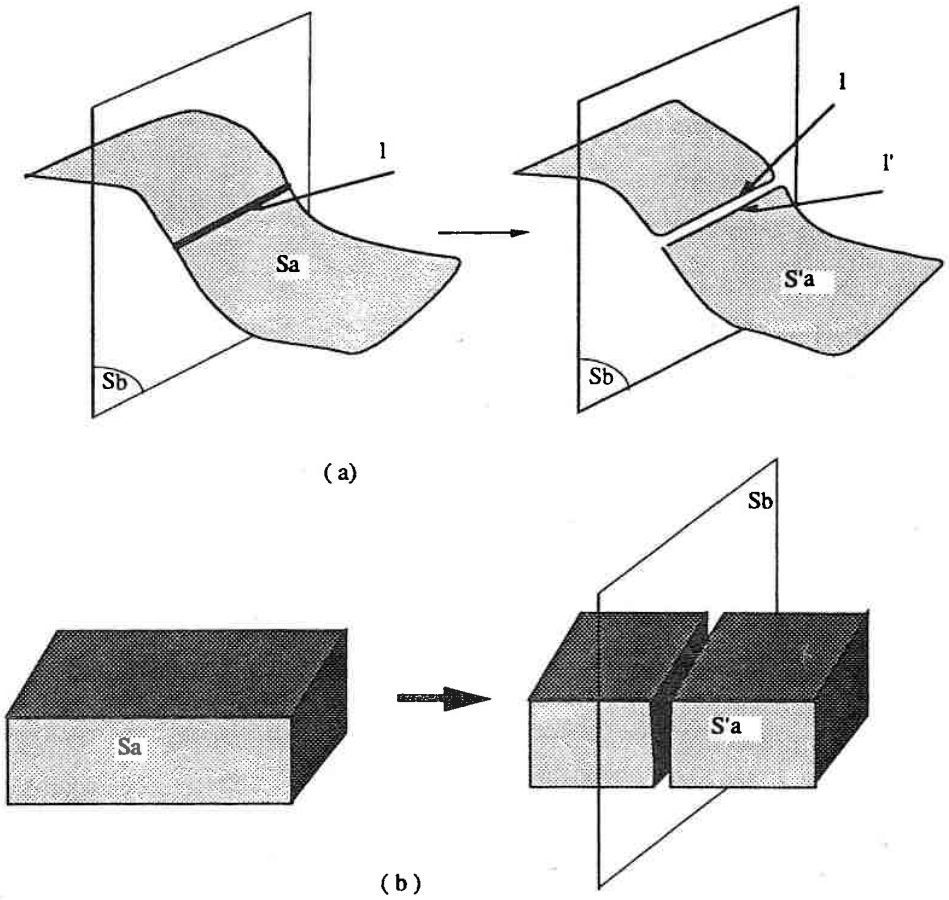


Figure 5.1 *S_a est découpée par S_b*

- La T-surface S_a représentée sur la Figure 5.1.(a) est supposée représenter une couche géologique. Au cours du processus de modélisation, le géologue reçoit de nouvelles informations indiquant que cette surface S_a doit être recoupée par une deuxième T-surface S_b correspondant à une faille dont la présence était jusque là ignorée. La prise en compte de la faille S_b consiste ici à découper S_a en deux morceaux séparés S_a^1 et S_a^2 situés de part et d'autre de S_b.

- La Figure 5.1(b) illustre un autre exemple classique en $CFAO$ où S_a représente une surface manufacturée que l'on souhaite séparer en deux morceaux S_a^1 et S_a^2 afin de pouvoir enlever par exemple S_2^a. Dans ce cas, il est nécessaire de découper S_a par le plan S_b.

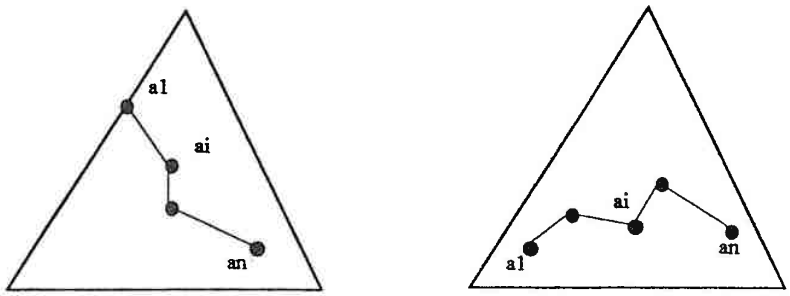


Figure 5.2 *Deux nouveaux types de lignes polygonales*

Dans les deux exemples ci-dessus, après découpage par S_b, la surface S_a est remplacée par une surface S'_a telle que :

S'_a	est géométriquement identique à S_a
S'_a	est topologiquement différent de S_a

Pour garder la cohérence du modèle de T-surface, c'est à dire, pour que S_a' soit aussi une T-surface, il est nécessaire de :

1. retrianguler les facettes de S_a qui sont intersectées par S_b et de
2. réarranger les liens entre les facettes de S'_a de telle façon que les facettes ayant la ligne $l = S_a \cap S_b$ comme côtés communs ne soient plus topologiquement adjacentes (voir [28]).

5.1.2 Configuration de l'intersection d'une facette de S_a avec S_b

Soient T_a une facette de S_a qui est intersectée par S_b et soit $L(T_a, S_b)$ l'ensemble des segments définis par :

$$L(T_a, S_b) = \bigcup_{T \in S_b} T \cap T_a$$

Soit $\lambda(T_a, S_b)$ l'ensemble des lignes polygonales qui peuvent être construites à partir des segments de $L(T_a, S_b)$. Dans le cas particulier où S_b est fermée, nous avons vu au chapitre 2.1 (cf page 63) que $\lambda(T_a, S_b)$ pouvait toujours être identifier à une combinaison des trois types canoniques. Dans le cas où S_b est ouverte, ces trois types déjà étudiés, on doit rajouter les deux nouveaux types représentés sur la Figure 5.2. Si nécessaire, comme nous l'avons fait au chapitre 2.1 pour les surfaces fermées, dans le cas où S_b est ouverte, on peut toujours réarranger les segments des lignes de $\lambda(T_a, S_b)$ de telle façon que ceux-ci ne se coupent jamais tout en ayant, éventuellement, des sommets communs (voir Figure 5.4).

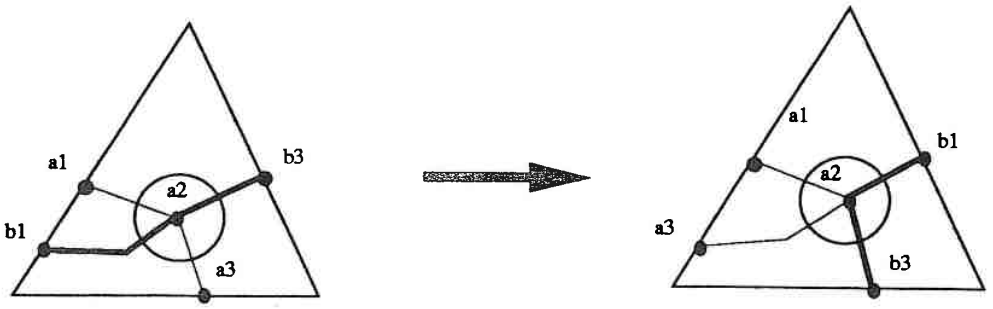


Figure 5.3 *Les lignes croisées de $\lambda(T_a, S_b)$ peuvent être converties en lignes non croisées.*

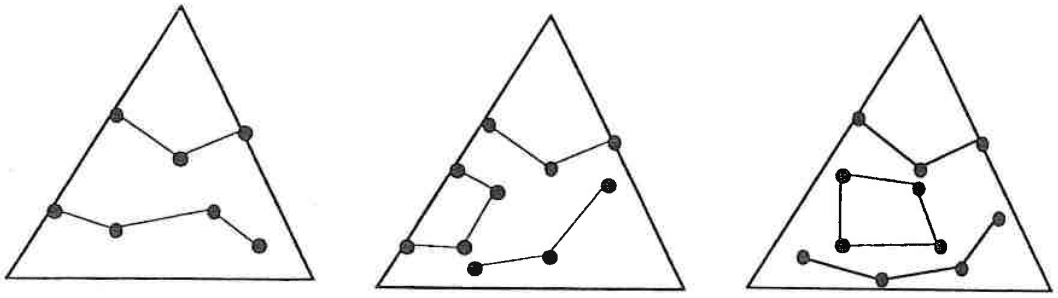


Figure 5.4 *Des cas concrets de lignes de $\lambda(T_a, S_b)$ obtenues en pratique.*

5.1.3 Partition de T_a par $\lambda(T_a, S_b)$ en régions polygonales

Afin de faciliter la discussion, nous utiliserons les notations suivantes :

$$\begin{aligned}
\lambda_{1,2}(T_a, S_b) &= \text{lignes polygonales de } \lambda(T_a, S_b) \text{ de type 1 et type 2.} \\
\lambda_3(T_a, S_b) &= \text{lignes polygonales de } \lambda(T_a, S_b) \text{ de type 3} \\
\lambda_4(T_a, S_b) &= \text{lignes polygonales de } \lambda(T_a, S_b) \text{ de type 4} \\
\lambda_5(T_a, S_b) &= \text{lignes polygonales de } \lambda(T_a, S_b) \text{ de type 5}
\end{aligned}$$

Le fait que les lignes d'intersection appartenant aux quatre ensembles ci-dessus ne se croisent pas a l'intérieur de T_a assure que la partition de T_a peut être être réalisée à l'aide du processus suivant lui même décomposé en quatre étapes :

1. T_a peut être décomposé en régions polygonales simples par $\lambda_{1,2}(T_a, S_b)$. L'ensemble de ces régions est noté $R(T_a) = \{P_1, P_2, \ldots, P_n\}$ tel que les sommets de chaque polygone P_i appartiennent soit à ceux de T_a soit à ceux de $\lambda_{1,2}(T_a, S_b)$. Considérons deux régions polygonales P_i et P_j appartenant à $R(T_a)$ et possédant des lignes de $\lambda_{1,2}(T_a, S_b)$ comme frontière commune. Pour que les lignes de $\lambda_{1,2}(T_a, S_b)$ fassent partie de la frontière de S'_a après l'opération de découpage, il est nécessaire que P_i et P_j soient topologiquement disjointes. Pour ce faire, comme le suggère la Figure 5.5, nous proposons de réaliser une

copie $\lambda'_{1,2}(T_a, S_b)$ de l'ensemble $\lambda_{1,2}(T_a, S_b)$ pour que $\lambda_{1,2}(T_a, S_b)$ et $\lambda'_{1,2}(T_a, S_b)$ subdivisent T_a en régions polygonales topologiquement disjointes.

2. Une fois l'étape 1 terminée, les lignes de $\lambda_3(T_a, S_b)$ appartiennent forcément aux éléments de $R(T_a)$. Considérons une ligne l appartenant à $\lambda_3(T_a, S_b)$ et située à l'intérieur d'un polygone $P_i \in R(T_a)$. Comme le suggère la Figure 5.5.b, pour que l fasse partie de la frontière de S'_a, on procède de la façon suivante :

   (a) Soient $\{a_1, a_2, \ldots, a_m\}$ les sommets de l dont on supposera que a_1 est le point situé sur le côté $p_j p_{j+1}$ de P_i et a_m celui situé à l'intérieur de P_i. On duplique tous ces sommets excepté a_m.

   (b) Soient $\{p_1, p_2, \ldots, p_n\}$ les sommets de P_i. On remplace P_i par le polygone P_i^* dont les sommets sont les suivants :

$$P_i^* = \{p_1, p_2, \ldots, p_j, a_1, a_2, \ldots, a_m, a'_{m-1}, a'_{m-2}, \ldots, a'_1, p_{j+1}, p_{j+2}, \ldots, p_n\}$$

3. Soit l une ligne polygonale de $\lambda_4(T_a, S_b)$ appartenant au polygone $P_i \in R(T_a)$. Pour que l fasse partie de la frontière de S'_a, on duplique l exceptées ses deux extrémités. Comme l'illustre la Figure 5.5.c, l et sa copie l' constituent un contour fermé définissant un polygone d'aire nulle qui peut être considéré comme un trou situé à l'intérieur de P_i. Selon le lemme présenté au paragraphe 4.5.3, P_i peut alors être transformé en un polygone simple sans trou.

   Les lignes de $\lambda_5(T_a, S_b)$ peuvent être traitées de façon analogue que celle pour $\lambda_4(T_a, S_b)$ (cf Figure 5.5.d).

A l'issue des trois étapes décrites ci-dessus, T_a est décomposée en régions polygonales simples. Pour obtenir la partition recherchée, il suffit donc maintenant de décomposer chacun de ces polygones en triangles.

5.1.4 Réarrangement des liens entre les facettes de S'_a

Après la décomposition présentée ci-dessus, les facettes triangulaires de S_a peuvent se séparer en deux groupes :

- celles qui n'ont pas été décomposées, c'est à dire, celles dont les triangles n'ont pas été intersectés par $S(V_2)$. Soit Ω_0 l'ensemble de ces facettes.

- celles qui ont été décomposées en sous-triangles. Soit Ω_1 l'ensemble de ces facettes.

Les facettes de Ω_1 seront remplacées par leurs triangles résultant de la décomposition. Il est donc nécessaire de réarranger les liens entre les facettes de S'_a (S'_a = subdivision de S_a). Ce réarrangement est effectué en deux étapes présentées ci-dessous; Soient T une facette triangulaire de Ω_1 et T^* l'un de ses trois triangles voisins :

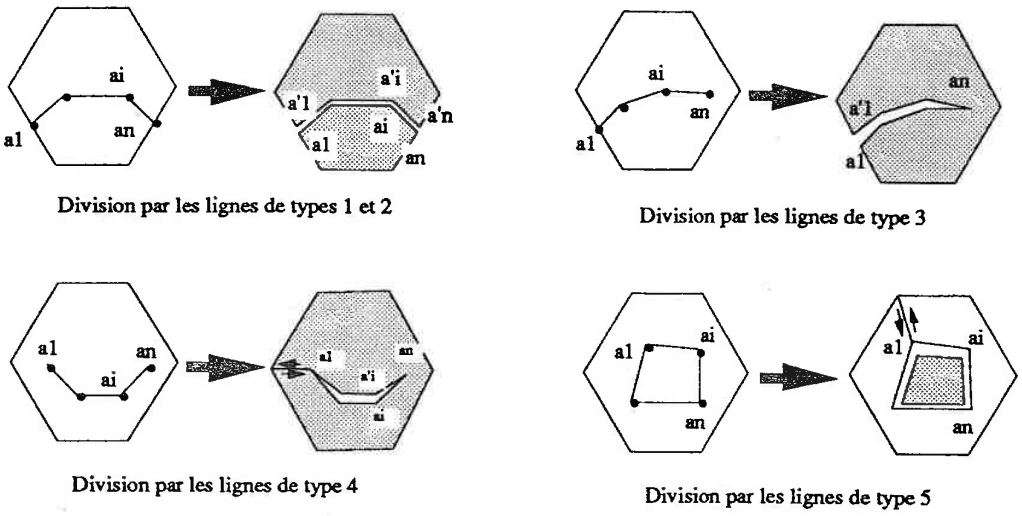


Figure 5.5 *Division d'un polygone simple par les différents types de lignes.*

Etape 1 : Liens entre les triangles résultant de la subdivision de T

Comme l'indique la figure 5.6.a, Supposons que T soit subdivisée par la ligne $l(a_1, a_2, a_3)$ en cinq triangles $\{T_1, T_2, T_3, T_4, T_5\}$. Par suite de la duplication des lignes d'intersection $l(a_1, a_2, a_3)$ au cours de la décomposition et d'après la définition de la relation $adj(A, B)$ présentée au chapitre 1.1[1], on peut affirmer que T_1 n'est pas le triangle adjacent de T_3 . Cette non adjacence de T_1 et T_3 reste vraie même si T_1 et T_3 ont un côté géométriquement confondu car, par suite de la duplication de $l(a_1, a_2, a_3)$, ce côté est lui aussi dupliqué.

Les liens corrects entre ces cinq triangles résultant de la subdivision représentée sur la Figure 5.6.a sont donc :

$$adj(T_1, T_2), \quad adj(T_3, T_4) \quad et \quad adj(T_4, T_5)$$

Etape 2 : Liens entre T et un triangle adjacent T^*

Suivant que T^* est subdivisée ou non, deux sous-cas doivent être pris en considération :

- $T^* \in \Omega_0$: Comme indiqué sur la figure5.6.b, puisque T sera supprimé de S_a pour construire S'_a, il ne peut donc plus être le triangle adjacent de T^* dans S'_a. Cependant, dans ce cas, il convient de noter qu'il existe toujours un triangle résultant de la subdivision de T qui est adjacent à T^*.

- $T^* \in \Omega_1$: Comme indiqué sur la figure5.6.c, supposons que T^* soit subdivisé en $\{T_1^*, T_2^*, T_3^*\}$. Dans ce cas, le lien entre T et T^* est remplacé par les liens entre les triangles résultant de la subdivision de T et T^*. On obtient donc les liens suivants :

$$adj(T_2, T_1^*) \quad \text{et} \quad adj(T_5, T_2)$$

[1]Si les deux triangles A et B ont un côté commun, alors le relation $adj(A, B)$ est vraie

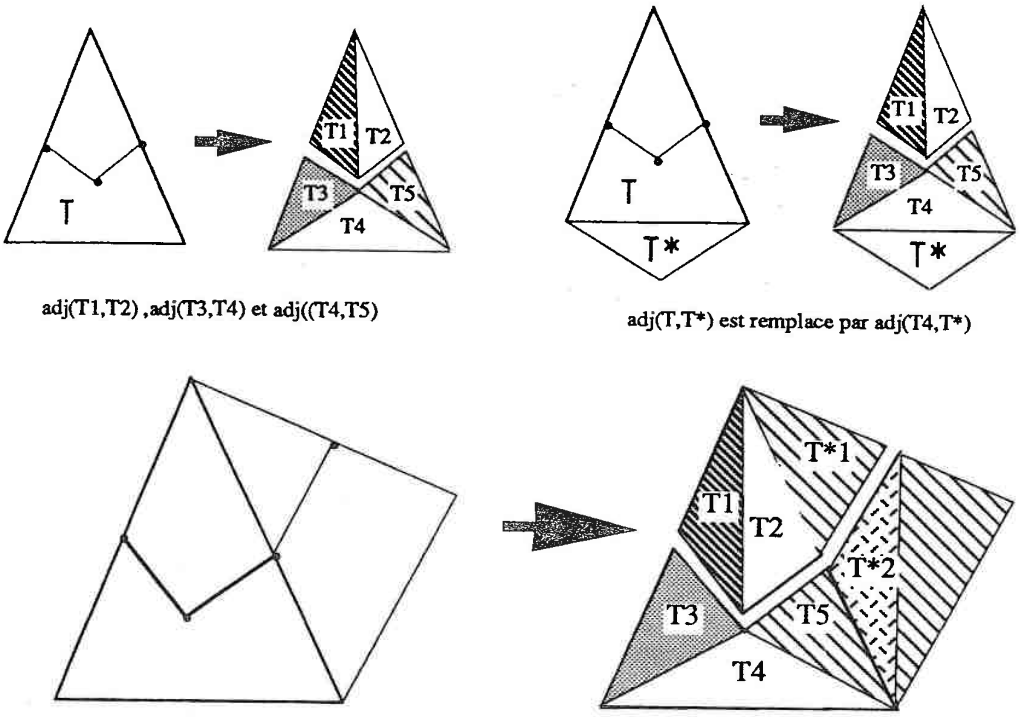


Figure 5.6 *Réarrangement des liens entre les triangles de S'_a.*

Remarque

Notre algorithme de découpage de T-surfaces peut s'étendre facilement aux P-surfaces[2]. Les problèmes qui peuvent être rencontrés pour la mise en oeuvre du découpage de P-surfaces sont pratiquement les mêmes que ceux des T-surfaces. En fait, il suffit de construire un P-octree pour la P-surface à couper et ensuite le processus de découpage peut être déroulé de la même façon que celui présenté pour les T-surfaces.

```
void Cut_Tsurf_by_Tsurf(S_a, S_b)
{
   Construction du T-octree de S_b ;

   pour-tout(triangle T ∈ S_a)
   {
```

[2]Voir définition page ??.

- Calcul de $L = T \cap S_b$:
  $L =$ ***Computing_Intersection_Segments***(T, S_b) ;
- Construction de l'ensemble Λ des lignes déduites de l'ensemble de segments L : $\Lambda =$ ***Set_Intersection_Lines***(L);
- Détermination du type de chaque ligne de Λ ;
- Division de T par les lignes de Λ en fonction de leur type ;
- Réarrangement des liens entre les triangles résultant du découpage de T;

}

pour-tout(triangle $T \in S_a$)
{
  if(T a été découpé)
    Réarrangement des liens de T avec ses triangles adjacents ;
}

return ;
}

La fonction ci-dessus appelle les fonctions suivantes qui ont déjà été présentées :

- *Computing_Intersection_Segments*( )
- *Set_Intersection_Lines*( )

5.2 Opérations interactives

Introduction

Dans ce qui suit, nous supposerons que les objets géométriques[3] à manipuler interactivement sont stockés dans une base de données géométriques dont le contenu est "filmé" par une "caméra informatique". Cett camera informatique projette l'image des objets filmés dans une fenêtre rectangulaire dessinée sur l'écran d'une station de travail (voir Figure 5.9) et nous nous proposons d'agir sur ces objets à travers des manipulations effectuées sur leurs images.

Afin de simplifier l'exposé, dans ce qui suit, comme le suggère la Figure 5.7, nous supposerons que l'espace 3D est repéré par rapport à un système de coordonnées orthonormé inverse $O_{cop}(x^v, y^v, z^v)$ attaché à la caméra et tel que :

- l'origine O_{cop} soit confondue avec le centre de projection de la caméra
- le vecteur unitaire $\vec{n}$ de l'axe $O_{cop}z^v$ soit orienté suivant l'axe de visée de la caméra
- le vecteur unitaire $\vec{v}$ de l'axe $O_{cop}y^v$ soit orienté suivant la verticale de la caméra

[3] Par exemple des T-surfaces.

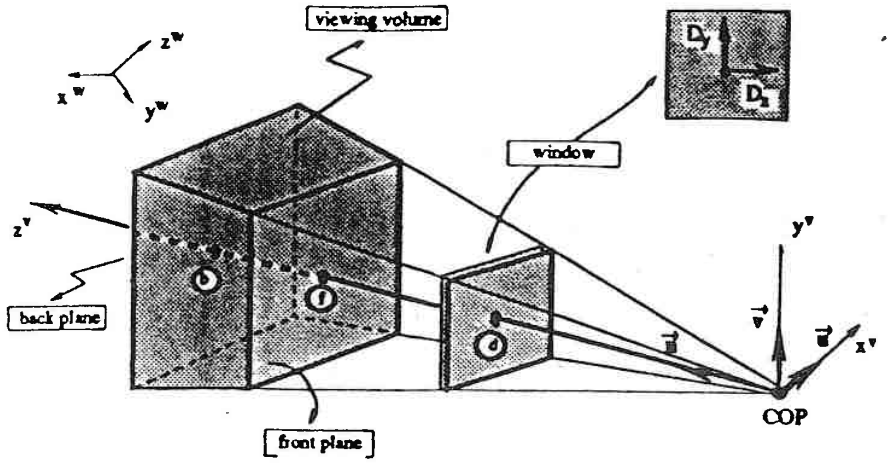


Figure 5.7 *Système de caméra informatique.*

- le vecteur unitaire $\vec{u}$ de l'axe $O_{cop}x^v$ soit tel que

$$\vec{u} = \vec{n} \times \vec{v}$$

Pour déterminer l'ouverture de la caméra, on utilise une fenêtre de vision homothétique de la fenêtre correspondant à l'image projetée sur l'écran et telle que :

- la fenêtre de vision a ses côtés parallèles aux axes $O_{cop}x^v$ et $O_{cop}y^v$
- la fenêtre de vision est centrée sur l'axe $O_{cop}z^v$ et est située à une distance d du centre de projection O_{cop}

Enfin, comme le suggère la Figure 5.7, la profondeur de champ correspondant au volume effectivement vu par la caméra[4] est définie par deux plans orthogonaux à l'axe $O_{cop}z^v$ appelés respectivement ***front plane*** et ***back plane*** (voir [8] [16]) et tels que :

- le front plane coupe l'axe $O_{cop}z^v$ à une distance f de $O_{cop}z^v$ et l'on suppose que les objets situés avant le front plane ne sont pas vus par la caméra
- le back plane coupe l'axe $O_{cop}z^v$ à une distance b de $O_{cop}z^v$ et l'on suppose que les objets situés après le back plane ne sont pas vus par la caméra

Vision perspective

Afin de simplifier l'exposé, on peut sans nuire à la généralité, supposer que la fenêtre de vision et la fenêtre correspondante sur l'écran sont confondues. Comme le montre la Figure 5.8, on constate alors que l'image p^* d'un point p appartenant au volume visible correspond à l'intersection

[4] On remarquera sur la Figure 5.7 que ce volume est en fait un tronc de pyramide.

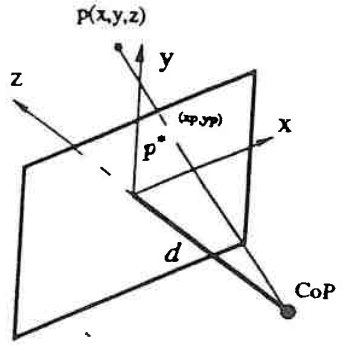


Figure 5.8 *Vision perspertive .*

de la droite O_{copp} avec la fenêtre de vision. Cette transformation appelée "transformation perspective" n'est pas linéaire mais peut tout de même être modélisée à l'aide d'opérations matricielles en utilisant des coordonnées homogènes (voir [8]).

Le fait important que nous retiendrons de la vision perspective ainsi définie est que la donnée d'un point p^* dans l'espace image (écran de la station de travail) définie un segment de droite $p_f p_b$ tel que, comme le suggère la Figure 5.9, on ait :

- le point p_f correspond à l'intersection de la droite O_{copp^*} avec le front plane
- le point p_b correspond à l'intersection de la droite O_{copp^*} avec le back plane

Tous calculs faits, les coordonnées (x_f^v, y_f^v, z_f^v) et (x_b^v, y_b^v, z_b^v) de ces deux points se déduisent des coordonnées (x^v, y^v, z^v) de p^* à l'aide des relations suivantes :

$$\begin{array}{lcllcl} x_f^v & = & \frac{x^v \cdot f}{f + z^v} & x_b^v & = & \frac{x^v \cdot b}{f + z^v} \\ y_f^v & = & \frac{y^v \cdot f}{f + z^v} & y_b^v & = & \frac{y^v \cdot b}{f + z^v} \\ z_f^v & = & f & z_b^v & = & b \end{array}$$

5.2.1 Construction d'un couteau informatique

Soient p^* et q^* deux points sélectionnés sur la fenêtre de vision à l'aide, par exemple, d'une souris et soient (p_f^*, p_b^*) et (q_f^*, q_b^*) les segments de droites associés définis au paragraphe précédent. Comme le suggère la Figure 5.10, on peut toujours considérer que ces deux points p^* et q^* définissent une T-surface $S(p^*, q^*)$ composée des deux triangles $T(p_f, p_b, q_f)$ et $T(p_b, q_f, q_b)$. Si nous posons $S_b = S(p^*, q^*)$ et si nous calculons la T-surface S_a' résultant de la découpe de S_a par $S_b = S(p^*, q^*)$, alors $S(p^*, q^*)$ joue le rôle de la lame du couteau.

Il est possible de généraliser ce processus de découpe en considérant non plus deux points, mais toute une suite de points $L = \{p^*, q^*, \ldots, r^*\}$ générant une T-surface $S(L)$ correspondant à la lame du couteau. L'algorithme de construction de $S(L)$ peut être décrit à l'aide de la fonction suivante présentée en pseudo-C :

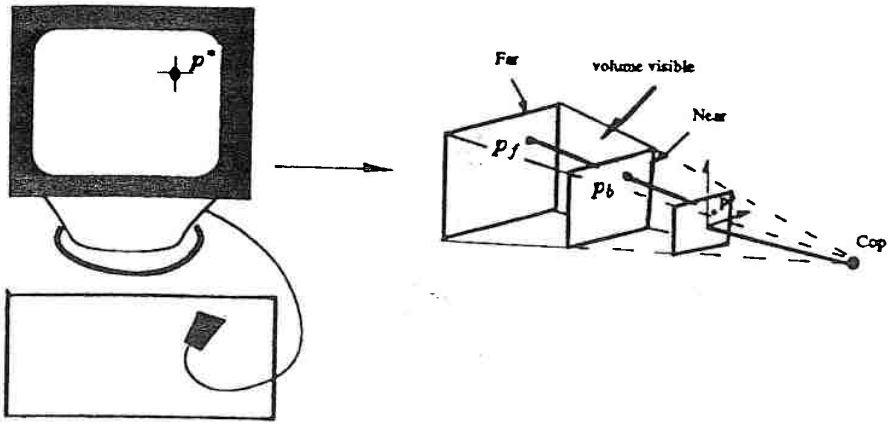


Figure 5.9 *Un point p^* sur l'écran définit un segment $p_f p_b$ en 3D.*

```
TSURF_t Get_Knife(L, camera)
{
   Soit S(L) la T-surface à construire à partir de L;
   Soient p le premier sommet de L et q le deuxième ;

   S(L) = φ ;
   while( q ≠ φ)
   {
      Recherche des segments p_f p_b et q_f q_b en fonction de camera;
      Construction des triangles T(p_f, p_b, q_f) et T(p_b, q_f, q_b)
      S(L) = S(L) ∪ T(p_f, p_b, q_f) ∪ T(p_b, q_f, q_b) ;

      p = q ;
      q = q->next ;
   }

   return( S(L) ) ;
}
```

Comme on le voit, cette fonction possède deux arguments d'entrée correspondant à la liste L des points sélectionnés sur l'écran et aux paramètres courants définissant la caméra utilisée. En sortie, la fonction retourne la T-surface $S(L)$ correspondant à la lame du couteau.

5.2.2 Construction d'une paire de ciseaux informatique

On peut remarquer que la lame du couteau définie au paragraphe pécédent recoupe complètement la partie de la surface S_a se trouvant dans le volume de vision y compris les parties cachées de S_a. Bien souvent, au lieu de tout découper de façon aveugle, on préfère ne découper que les

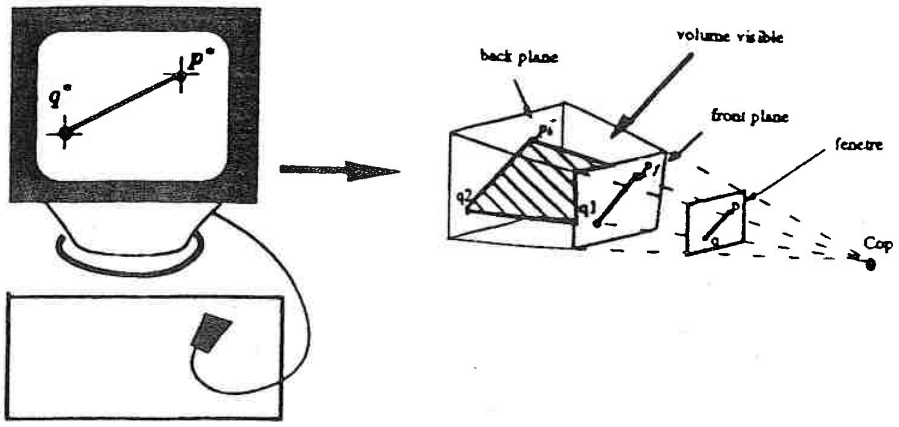


Figure 5.10 *Construction d'un couteau informatique (une T-surface en 3D).*

parties directement visibles à partir du point de vue défini par la position de la caméra un peu comme le ferait une paire de ciseaux découpant une pièce de tissu. Pour ce faire, il est nécessaire de modifier la position des points $\{p_b, q_b, \ldots, r_b\}$ utilisés pour construire la lame de couteau au paragraphe précédent afin que cette lame ne coupe que la partie visible de la surface S_b découpée.

5.2.3 Application à la modélisation des failles en géologie

Position du problème

En Géologie, la faille est un phénomène accidentel qui, au cours de l'évolution de la terre, provoque des cassures dans les surfaces correspondant aux couches géologiques; en pratique, ces cassures peuvent elles-mêmes être assimilées à des surfaces découpant les surfaces géologiques comme le suggère la Figure 5.11.

Généralement, les failles ont des formes géométriques complexes et sont très différentes de l'une à l'autre dans l'espace. Par ailleurs, la pluspart du temps, on a peu d'informations pour les caractériser, et leur modélisation est souvent réalisée en plusieurs étapes par approximations successives. On imagine facilement que les couteaux et ciseaux informatiques peuvent être un outil très intéressant pour simuler la mise en place interactive de ces failles. Dans ce qui suit, nous présentons en détail les différentes opérations correspondant à la mise en place d'une faille recoupant une surface donnée.

Sélection des objets sur l'écran

Soit $\sigma = \{S_1, S_2, ..., S_n\}$ un ensemble de T-surfaces affichées sur l'écran à l'aide d'une caméra informatique. Ces surfaces sont composées de triangles eux-mêmes composés de sommets et c'est pourquoi on est amené à distinguer deux catégories d'objets pouvant être sélectionnés par exemple à l'aide d'une souris :

- Les objets principaux correspondant par exemple aux T-surfaces affichées à l'écran

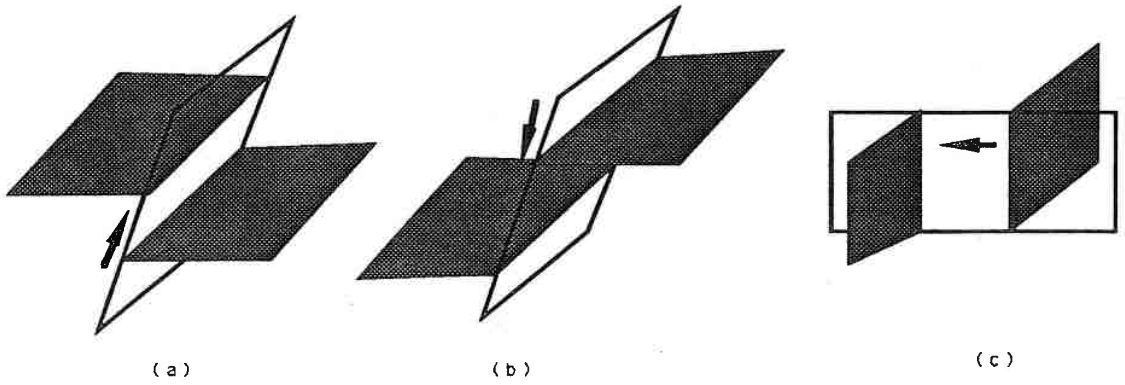


Figure 5.11 *Trois exemples de failles géologiques simples.*

- Les sous-objets qui comprennent les sommets, les arêtes ou les facettes d'une T-surface.

Sélection d'une T-surface

Soit $p_f p_b$ le segment associé à un point p^* sélectionné sur l'écran et soit Q l'ensemble des intersections $q(S)$ de $p_f p_b$ avec chacune des surfaces S affichées à l'écran. Deux cas sont à considérer :

- Si Q est vide, alors la tentative de sélection a échoué et aucune surface n'est sélectionnée.
- Si Q n'est pas vide, alors la tentative de sélection a réussi et la surface S sélectionnée est celle pour laquelle le point $q(S)$ associé est le plus proche du centre optique de la caméra.

Sélection d'un sommet de triangle de T-surface

La sélection d'un sommet de triangle de T-surface est effectuée avec une tolérance de séléction notée ϵ correspondant au "rayon" d'une petite fenêtre carrée entourant le point désigné sur l'écran à l'aide de la souris. Seuls les sommets p de triangles dont l'image sur l'écran tombe dans cette fenêtre sont considérés comme des candidats à la sélection (cf fig 5.12); soit A l'ensemble de ces sommets sélectionnés. Deux cas sont à considérer :

- Si A est vide, alors la tentative de sélection a échoué et aucun sommet n'est sélectionné.
- Si A n'est pas vide, alors la tentative de sélection a réussi et le sommet sélectionné est celui de A qui est le plus proche du centre optique de la caméra.

Désignation d'un vecteur dans l'espace

En appliquant la méthode de la sélection des objets sur l'écran présentée ci-dessus, on peut désigner des points de l'espace 3D comme par exemple des sommets de triangles de T-surface.

Soit q un premier point sélectionné à l'intérieur du volume visible et soit p^* un deuxième point désigné sur l'écran à l'aide de la souris. On notera que :

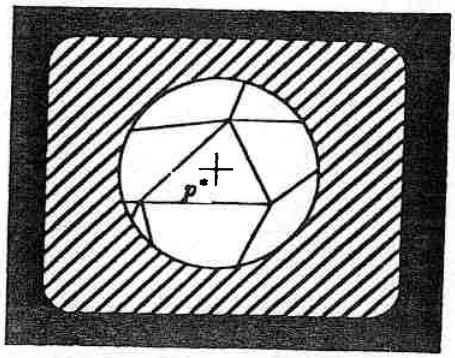


Figure 5.12 *Sélection d'un sommet de triangle*

- Le premier point q correspond à un point caractéristique de l'un des objets affichés sur l'écran, par exemple le sommet de l'un des triangles d'une T-surface.

- Le second point p^* est à priori indépendant des points caractéristiques des objets affichés sur l'écran. En fait, p^* correspond à un point situé dans la fenêtre de vision et n'est utilisé que pour définir le segment $p_f^* p_b^*$ associé.

Soit Q le plan passant par q et parallèle au plan $(O_{cop}x^v, O_{cop}y^v)$; en pratique si l'on assimile l'image de q à q lui même, alors Q peut être considéré comme un plan parallèle à l'écran. Il est naturel de définir le vecteur $\vec{qp}$ comme étant le vecteur d'origine q correspondant au premier point et d'extrémité p tel que :

$$p = p_f^* p_b^* \cap Q$$

Translation d'une T-surface

Etant donné un objet dans l'espace, la translation de celui-ci consiste à déplacer chaque point de cet objet d'un vecteur donné. Reprenons la classification des objets dans la sous-section 1 :

- Pour un sous-objet quelconque, la translation peut être effectuée de façon triviale, il suffit de translater tous les sommets de ce sous-objet.

- Pour un objet principal comme une T-surface, il est évident que sa translation peut s'effectuer de la même façon que celle d'un sous-objet. Or la exécution de la translation d'un morceau d'une T-surface n'est pas triviale, car les facettes d'une T-surface sont arrangées de façon quelconque. Il est donc nécessaire de trouver les facettes appartenant au morceau sélectionné. Tout d'abord, on sélectionne une facette T_c sur le morceau S_c. On marque toutes les facettes T qui ont la relation $\mathit{link}(T_c, T)$ à l'aide d'une procédure récursive. L'ensemble de ces facettes marquées constitue réelement le morceau S_c de T-surface à sélectionner; ensuite on translate toutes ces facettes du vecteur désigné.

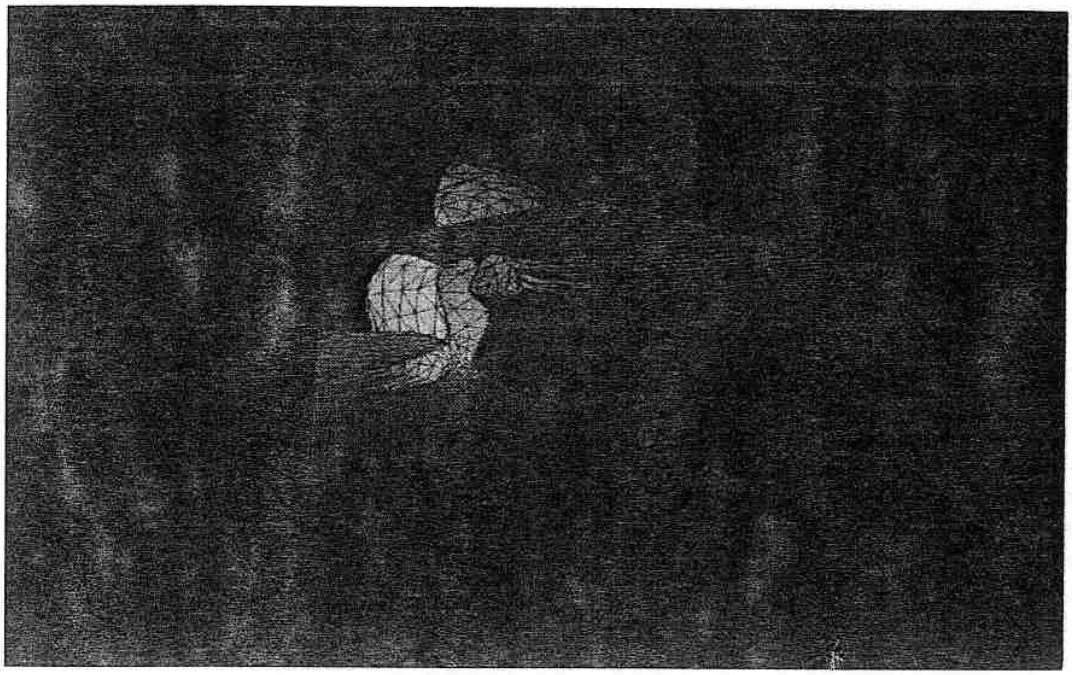


Figure 5.13 *Une simulation de faille sur une couche géologique en forme de diapir.*

Simulation de failles

Ainsi que nous l'avons dit en introduction, en géologie, une faille peut être considérée comme une surface de rupture complexe de l'espace 3D suivant laquelle les couches géologiques se sont rompues. En pratique, comme le montre la Figure 5.13, la modélisation d'une faille doit être effectuée en trois étapes :

1. modélisation de la surface complexe correspondant à la faille
2. coupure des surfaces géologiques par la faille
3. translation relative des morceaux de surfaces géologiques situés de part et d'autre de la surface de la faille

Les vecteurs de translation sont soit donnés soit construits interactivement à l'aide de la procédure décrite au paragraphe précédent.

5.2.4 Application à la modélisation des dômes de sel en géologie

Au départ, lors de leur formation, les couches géologiques sédimentaires sont toutes horizontales. Les couches de sel beaucoup plus légères que les autres ont tendance à remonter localement sous forme de gouttes de sel intersectant les couches supérieures plus denses comme le montre la

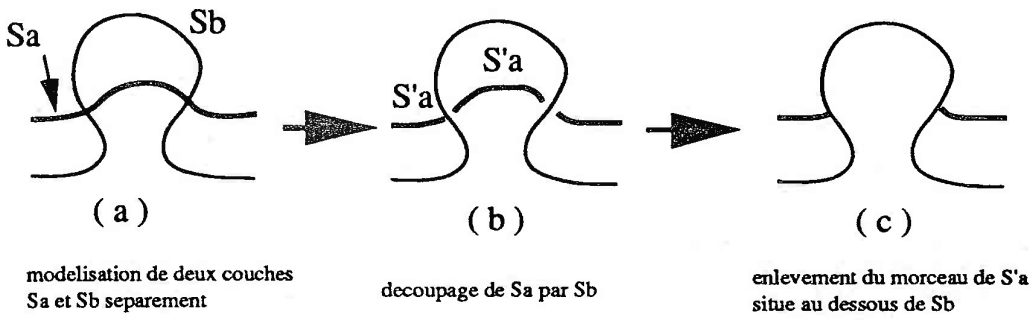


Figure 5.14 *Modélisation du dôme de sel .*

Figure 5.14.c. Sur cette Figure, la surface S_b correspond à l'interface séparant la couche de sel et une couche plus dense située au dessus et elle même limitée vers le haut par une deuxième interface S_a. Au cours du processus de modélisation, les surfaces S_a et S_b sont tout d'abord modélisées séparément et l'on obtient ainsi la Figure 5.14.a qui physiquement inacceptable pour un géologue. Pour passer de la Figure 5.14.a, à la Fighure 5.14.c, il est nécessaire de passer par l'étape représentée sur la Figure 5.14.b. Les deux opérations représentées par des flèches sur la Figure 5.14 sont les suivantes :

1. calcul de la découpe S'_a de S_a par S_b.
2. suppression du morceau de S'_a situé au dessous de S_b

Pour réaliser l'étape 2 ci-dessus, on sélectionne d'abord une facette T appartenant à la partie à enlever S'_a et on supprime ensuite toutes les facettes T^* topologiquement liées avec T.

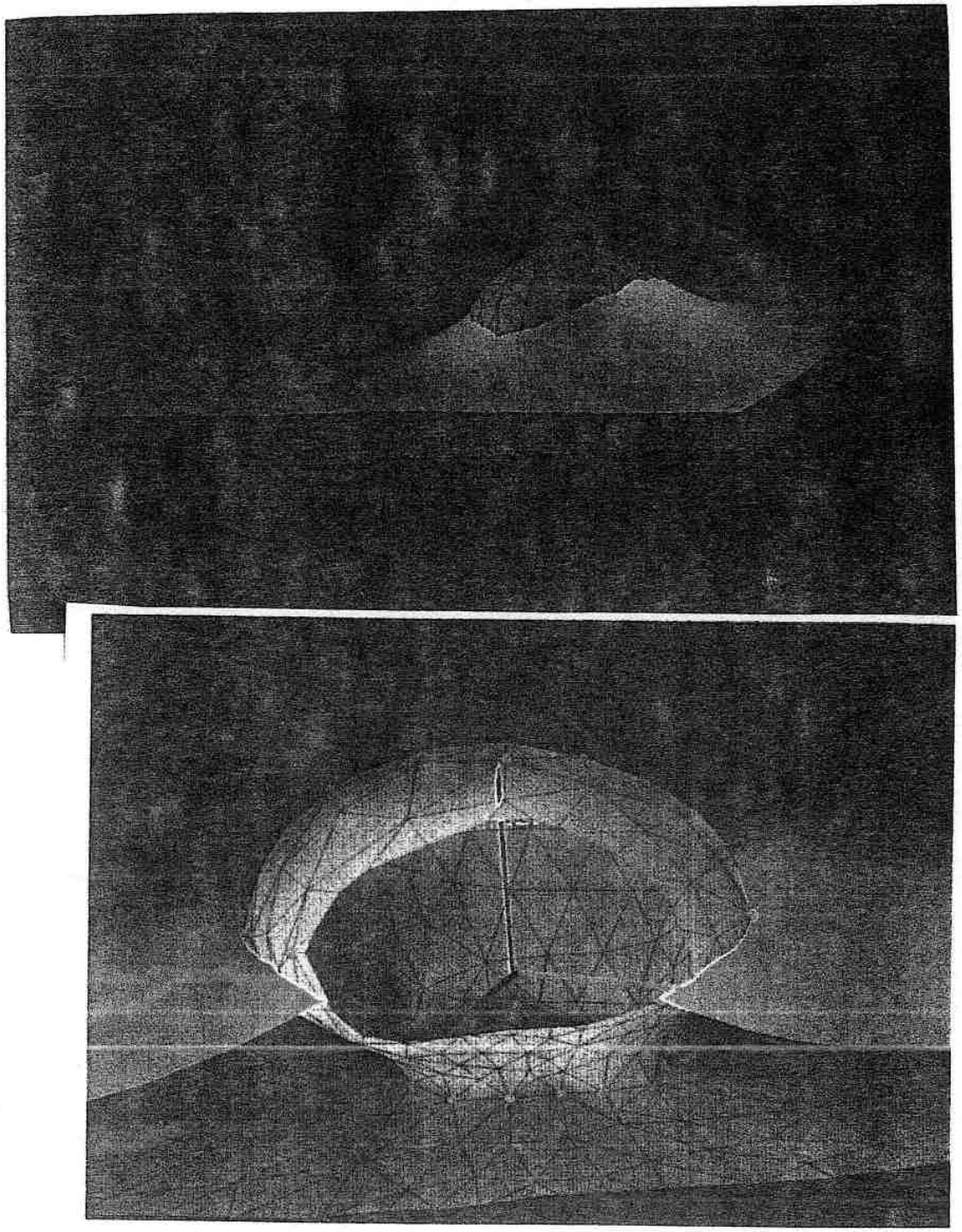
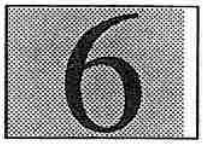


Figure 5.15 *Modélisation d'un dôme de sel à l'aide du logiciel GOCAD.*

Partie III

Méthodes de Modélisation de T-surfaces

Chapitre



Triangulation de Delaunay

Nous rappelons ici brièvement le principe de la triangulation de Delaunay et deux algorithmes pour sa mise en oeuvre. Pour une description plus approfondie et plus complète, on pourra consulter (voir [9]) et les références qui y sont citées. Dans ce qui suit, nous nous intéresserons à la triangulation de Delaunay en 2D et en 3D.

6.1 Diagramme de Voronoï

6.1.1 Définition

Soient $P = \{p_1, p_2, \ldots, p_N\}$ un ensemble de points dans l'espace Euclidien E^m et $d(p_i, p_j)$ la distance euclidienne entre deux points p_i et p_j. A chaque point p_i, nous pouvons associer une région $R(i)$ définie par :

$$R(i) = \{p \in E^m \quad | \quad d(p, p_i) \leq d(p, p_j)\} \quad j \in [1, N]$$

qui constitue le lieu géométrique des points qui sont plus proches du point p_i que de tous les autres points de P. Cette région est appelée le polyèdre de Voronoï associé au point p_i. L'ensemble de ces polyèdres associés à tous les points de P forme alors une partition de l'espace E^m, appelée partition de Voronoï.

Prenons $m = 3$. On peut interpréter géométriquement les polyèdres de Voronoï en utilisant une analogie biologique. Imaginons que chaque point représente le noyau d'une cellule et que toutes les cellules se développent simultanément vers l'extérieur à partir de leur noyau, et avec la même vitesse de croissance. Dés que deux cellules se touchent, elles s'arrêtent au point de contact. Au bout d'un certain temps, chaque cellule sera en contact avec ses voisines, et seules celles dont le noyau se trouve sur l'enveloppe convexe de P pourrons croître à l'infini. La forme résultante de chaque cellule est exactement le polyèdre de Voronoï correspondant à son noyau.

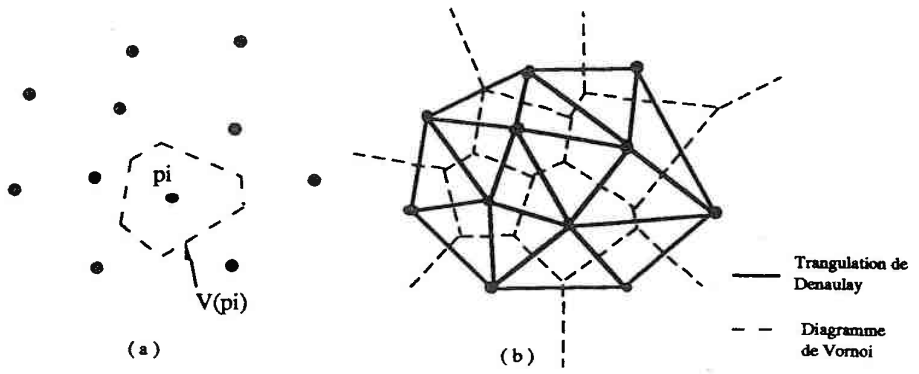


Figure 6.1 *(a) Polygone de Voronï; (b) Le diagramme de Voronoï et la triangulation de Delaunay.*

A partir des polyèdres de Voronoï, on peut définir une relation de voisinage entre les points de P :

Deux points p_i et p_j sont dit voisins si leurs polyèdres de Voronoï respectifs sont adjacents.

Avec cette notion de voisinage, en reliant les points voisins de P, on obtient alors une triangulation de P, connue sous le nom de triangulation de Delaunay (dual du Voronoï).

Dans le cas $m = 2$, intuitivement s'il n'y a que deux points, le demi-plan $H(p_i, p_j)$ défini par la médiatrice de $p_i p_j$ contenant p_i est justement le polygone de Voronoï associé à p_i. Donc, généralement le polygone de Voronoï de p_i est donné par $V(i) = \cap H(p_i, p_j)$ avec $j \in [1, N]$ et $j \neq i$ (cf Fig 6.1.a). Le diagramme de Voronoï peut être donné par $V = \cup V(i)$ avec $i \in [1, N]$. On obtient son dual en reliant une paire de points de P qui partagent une même arête du diagramme. En général, il contient toutes les informations de proximité. Ceci joue un rôle fondamental et puissant dans l'algorithmique géométrique (cf Fig6.1.b).

6.1.2 Propriètés en 2D d'un diagramme de Voronoï

Les propriètés importantes d'un diagramme de Voronoï en 2D sont les suivantes :

1. le polygone $V(i)$ est ouvert si et seulement si p_i est un point qui est sur l'enveloppe convexe de P.
2. Le diagramme de Voronoï d'un ensemble de N points posséde au plus $2N - 5$ sommets et $3(N - 2)$ arêtes.
3. Pour toutes triangulations d'un ensemble P de N points dont K points sont sur l'enveloppe convexe de P, il y a $2(N - 1) - K$ triangles et $3(N - 1) - K$ arêtes.
4. la triangulation de Delaunay vérifie le critère "Max-Min angle" (voir [9]). Ce critère maximise le plus petit angle de chacun des triangles de la triangulation, ce qui permet de minimiser le nombre de triangles allongés, propriété importante pour les calculs numériques.

De plus, en supposant qu'il n'existe pas 4 points cocirculaires on a :

- Chaque sommet du diagramme de Voronoï d'un ensemble de points P est exactement l'intersection de trois arêtes du diagramme (cf Fig 6.1.b), le cercle dual défini par le triangle de Delaunay ne contient aucun point de P. De plus la triangulation de Delaunay sur l'ensemble de points P est unique.

6.2 Construction de la triangulation de Delaunay en 2D

6.2.1 Complexité du problème

En 2D, Shamos (voir [22]) a démontré le théorème ci-dessous qui établit la borne inférieure de la complexité du probléme dans le pire des cas.

Théorème 1

La construction du diagramme de Voronoï pour n points dans un plan est en $O(n * log(n))$.

Dans l'espace, on a le théorème suivant dû à Klee (voir [9]).

Théorème 2

Le coût de la construction de la triangulation de Delaunay pour n points dans l'espace est $O(n^2)$.

Les algorithmes de la construction des diagrammes de Voronoï et de Delaunay connus à ce jour appartiennent à deux catégories :

- les algorithmes du type "diviser pour régner". On dispose de l'ensemble des données .
- les algorithme dits incrémentaux. Ils construisent les diagrammes dynamiquement en fonction d'une nouvelle donnée.

Dans la suite, nous présentons l'algorithme de Shamos (diviser pour régner) et un algorithme classique (voir [9]) qui construit de façon incrémentale la triangulation.

6.2.2 Algorithme de Shamos

L'idée essentielle est d'utiliser la technique "diviser pour régner" afin d'optimiser l'algorithme.

Soit $P = \{p_1, p_2, \ldots, p_n\}$ un ensemble de points en 2D, la procédure $Vor(P)$ peut s'écrire en trois étapes :

1. Partitionner P en deux sous-ensembles P_1 et P_2 de taille égale (si possible).
2. Construire $Vor(P_1)$ et $Vor(P_2)$ récursivement.

3. Fusionner $Vor(P_1)$ et $Vor(P_2)$ pour obtenir $Vor(P)$.

L'étape 1 et L'étape 3 se font en $O(n)$ (voir [23]), on arrive donc à un algorithme optimal en $O(n \cdot log(n))$.

Dans l'étape 1, la partition se fait généralement suivant les coordonnées en x; l'étape 2 est simple, car la construction de $Vor(P_1)$ (et $Vor(P_1)$) se fera lorsque l'on n'a plus que 2 points, dans ce cas $Vor(P_1)$ (et $Vor(P_1)$) est la médiatrice de ces deux points. Pour expliquer l'étape 3, nous allons présenter les lemmes suivants :

Soit $\sigma(P_1, P_2)$ l'ensemble des arêtes de $Vor(P)$ qui sont partagées par $V(i)$ et $V(j)$ pour $pi \in P1$ et $pj \in P2$.

Lemme 1

$\sigma(P_1, P_2)$ est l'ensemble des arêtes d'un sous-graphe de $Vor(P)$ (voir [22]).

Lemme 2

Si $\sigma(P_1, P_2)$ n'a qu'une arête, alors $\sigma(P_1, P_2)$ est une droite; sinon, ses deux arêtes extrêmes sont des demi-droites.

Supposons que P_1 et P_2 soient séparés par une ligne verticale. Désignons par π_l la partie du plan située à gauche de σ et par π_r la partie du plan située à droite.

Lemme 3

$Vor(P)$ est l'union de $Vor(P_1) \cap \pi_l$ et $Vor(P_2) \cap \pi_r$.

La fusion de $Vor(P_1)$ et $Vor(P_2)$ peut s'effectuer en trois sous-étapes :

1. Construire la chaine polygonale σ séparant P_1 et P_2.
2. Enlever les arêtes ou parties d'arêtes de $Vor(P_2)$ situées à gauche de σ et enlever les arêtes ou parties d'arêtes de $Vor(P_1)$ situées à droite de σ.
3. $Vor(P)$ est l'union de $Vor(P_1)$ et $Vor(P_2)$ mise à jour à l'étape précédente.

L'exemple de la figure 6.2 illustre le processus de la construction de σ :

Soit t_1 la tangente commune en haut à P_1 et P_2, dans l'exemple, c'est la droite passant par le point 3 de p_1 et le point 7 de P_2. σ est initialisé par la médiatrice de t_1. on parcour cette médiatrice de haut au bas, on rencontre un côté de $Vor(P_2)$ (ici le côté séparant le point 7 et le point 8 de P_2) et on construit σ avec la médiatrice du point 3 de P_1 et le point 8 de P_2. On répète ce processus jusqu'à trouver la médiatrice de la tangente commune en bas qui est ici la droite passant par le point 6 de P_1 et le point 10 de P_2.

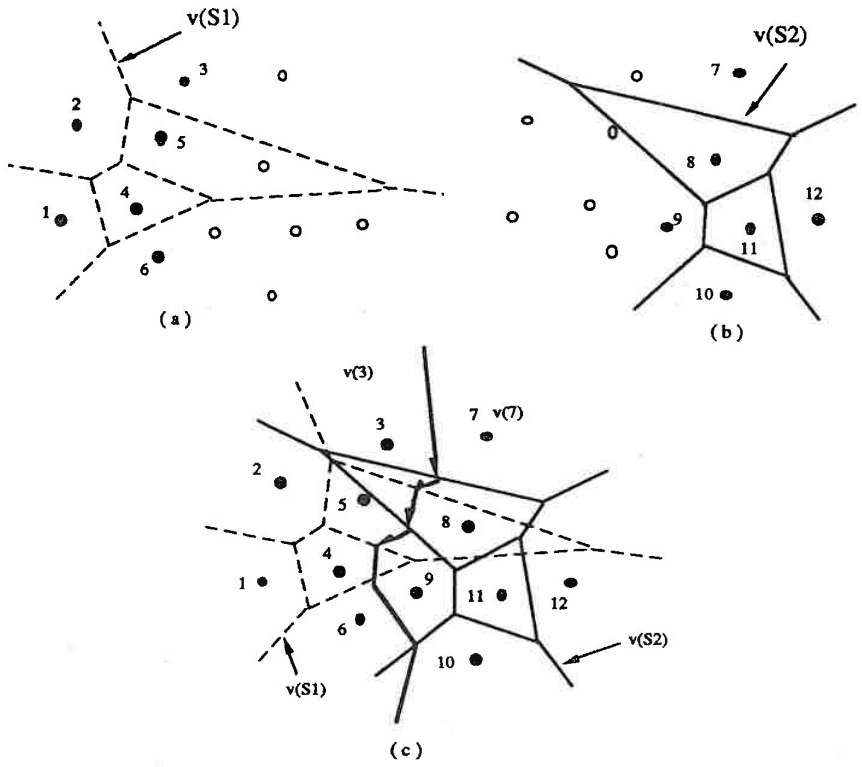


Figure 6.2 *Construction du diagramme de Voronoï par l'algorithme de Shamos.*

6.2.3 Algorithme incrémental

La construction dynamique du diagramme de Voronoï signifie que chaque point est introduit l'un après l'autre au cours de la construction. Ceci répond à beaucoup de besoins pratiques. La complexité de cet algorithme est en $O(n * n)$ dans un plan, sa génération en 3D coute $O(n^3)$ dans le pire des cas.

On construit d'abord un triangulation de Delaunay sur un ensemble réduit de points qui peut se limiter à trois points non-alignés. Quand on ajoute un nouveau point dans une triangulation de Delaunay, celle-ci n'est en général modifiée que localement. Notons τ_m la triangulation de Delaunay associée aux m premiers points de P, $\mathcal{T}$ l'ensemble des triangles de τ_m et C l'union des cercles circonscrits aux triangles de $\mathcal{T}$. La modification nécessaire dans la triangulation τ_m pour obtenir une triangulation de Delaunay τ_{m+1} dépend de la position du nouveau point ajouté p_{m+1}. On peut distinguer trois types de situations géométriques de ce point par rapport aux ensembles $\mathcal{T}$ et C :

- p_{m+1} appartient à $\mathcal{T}$.
- p_{m+1} n'appartient pas à $\mathcal{T}$ ni à C.

- p_{m+1} n'appartient pas à $\mathcal{T}$ mais à C.

La figure 6.3.a montre un exemple où les points p, q et r sont respectivement en situation 1, 2 et 3.

Traitement du cas 1

La modification nécessaire dans ce cas consiste à chercher tous les triangles dans $\mathcal{T}_m$ dont le cercle circonscrit contient le point p_{m+1}. Soient Γ l'ensemble de ces triangles et $F_1, F_2, \ldots, ..., F_n$ les côtés des triangles de Γ non communs à deux triangles de Γ. La triangulation de Delaunay τ_{m+1} est alors l'union de $\tau_m - \Gamma$ avec $(p_{m+1}, F_j | j = 1, ..., n)$ qui sont les nouveaux triangles, construits en reliant les côtés F_j avec le point p_{m+1}. La Figure 6.3.b montre une telle construction.

Traitement du cas 2

Dans ce cas, on obtient la triangulation τ_{m+1} en joignant simplement le point p_{m+1} avec les côtés de l'enveloppe convexe de τ_m qui sont vus par le point p_{m+1} (cf Figure 6.3.c).

Traitement du cas 3

On cherche d'abord les triangles dans τ_m dont le cercle circonscrit contient le point p_{m+1}. Soient Γ l'ensemble de ces triangles et $F_1, F_2, \ldots, F_n$ les côtés des triangles de Γ non communs à deux triangles de Γ. On cherche ensuite les côtés de l'enveloppe convexe de τ_m qui n'appartiennent pas à Γ mais sont vus par le point p_{m+1}. Soient $F_{n+1}, \ldots, F_l$ de tels côtés. La triangulation τ_{m+1} est alors l'union de $(\tau_m - \Gamma)$ avec $(p_{m+1}, Fj | j = 1, \ldots, n, n+1, \ldots, l)$ qui sont les nouveaux triangles. construits en reliant les côtés e_j avec le point p_{m+1}. L'exemple illustré sur la Figure 6.3.d permet d'observer le processus d'une telle construction.

En poursuivant ainsi l'ajout des points dans P un par un jusqu'au dernier point, on obtient la triangulation de Delaunay complète de tous les points de P.

6.2.4 Obtention de T-surface à partir de la triangulation de Delaunay 3D

Soit un objet V dans l'espace, dont la frontière $S(V)$ est une surface sur laquelle un ensemble de n points $P = \{p_1, \ldots, p_n\}$ sont connus par leurs coordonnées. L'objetif est de représenter la forme $S(V)$ de cet objet par une T-surface.

On construit d'abord la triangulation de Delaunay en 3D qui est un ensemble de tétraèdres, $\mathcal{T}(P)$ sur P, soit $V(P)$ le volume représenté par l'ensemble de $\mathcal{T}(P)$. Si tous les points de P sont sur l'enveloppe convexe, alors la peau $S(V(P))$ du volume $V(P)$ est la T-surface cherchée; sinon on doit procéder en éliminant des tétraèdres les uns après les autres de manière à respecter la règle suivante jusqu'à ce que tous les points de P soient sur la frontière.

règle

les tétraèdres que l'on peut éliminer sont ceux qui ont exactement une face, 3 arêtes et trois points sur $S(V(P))$ ou ceux qui ont deux faces , cinq arêtes et quatre points sur $S(V(P))$.

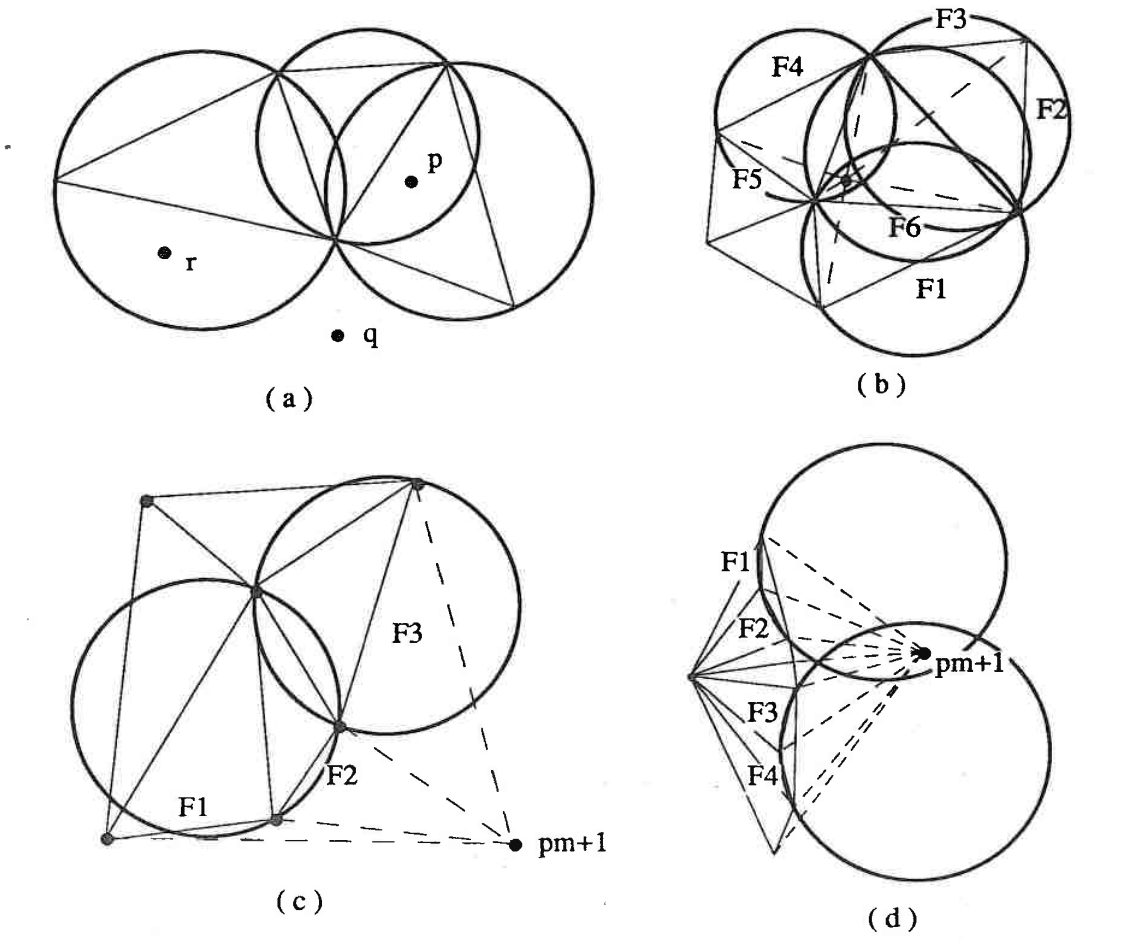


Figure 6.3 *Construction de la triangulation tau_{m+1} à partir de celle de τ_m.*

Ce processus est semblable à une sculpture. A la sortie de ce processus $S(V(P))$ est la T-surface cherchée.

Chapitre

7

Conversion de grille (2D et 3D) en T-surfaces

7.1 Conversion de grille 2D

Soit $z = \varphi(x, y)$ une fonction connue par les valeurs aux noeuds d'une grille régulière (rectangulaire) et dont le graphe est constitué par une surface S plongée dans l'espace (ox, oy, oz). On se propose d'approcher S par une T-surface en n'utilisant pour ce faire que les seules données fournies par la grille. Si la fonction $\varphi(x, y)$ est supposée continue, la solution est triviale car il suffit de découper chaque cellule de la grille en deux triangles.

Dans les applications géologiques, $\varphi(x, y)$ est souvent affectée par des lignes de discontinuités correspondant à des "failles", il est alors nécessaire de redécouper les triangles traversés par les discontinuités. En pratique, nous proposons d'opérer de la façon suivante :

- on triangule S en supposant qu'il n'y a pas de discontinuité
- chaque discontinuité est approchée par un ensemble de lignes polygonales que l'on peut utiliser pour générer une surface cylindrique verticale D (voir Figure 7.1) elle même triangulée
- on effectue le découpage de la surface triangulée S par la surface triangulée D à l'aide de la méthode " découpage de T-surfaces" présentée au chapitre 2.2.

7.2 Conversion de grille 3D

Introduction

Dans certaines applications, une surface complexe est définie par une grille régulière 3D. Par exemple, en exploitation minière, le corps du gisement est souvent découpé en petits blocs

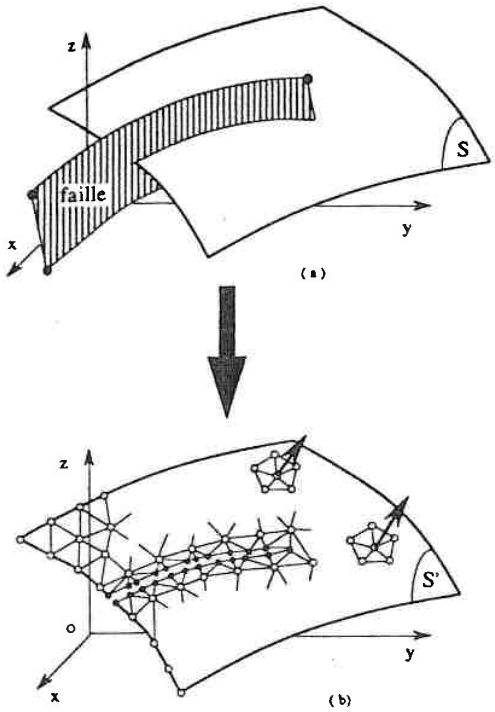


Figure 7.1 *(a) construction de la surface cylindrique verticale représentant la faille; (b) Génération de T-surface à partir de grille 2D avec discontinuité.*

réguliers, si on attache la teneur moyenne de chaque bloc à son centre, alors on obtient des données définies sur une grille régulière 3D. La teneur $t = \varphi(x, y, z)$ en chaque noeud de la grille est estimée par la méthode de Krigeage et le corps de gisement G est défini par l'ensemble de points possédant la teneur plus grande qu'une teneur de coupure t_0. Bien souvent, G est composé de plusieurs morceaux isolés de forme "lentilles", donc le bord de G est en fait une surface très complexe.

Dans ce qui suit, nous proposons une méthode qui convertit une surface définie par une grille 3D et une teneur de coupure t_0 en une T-surface. Les données d'entrée sont une liste de valeurs correspondant aux valeurs en chaque noeud de la grille. Tout d'abord, notre méthode calcule les points d'intersection de la surface avec la grille; et ensuite, nous effectuons une triangulation sur ces points d'intersection. Cette triangulation peut être faite en n'utilisant à chaque étape que les valeurs associées aux 8 noeuds voisins dans la grille. Donc cette méthode a l'avantage d'utiliser peu de mémoire et d'être efficace.

Lorsque la T-surface obtenue présente des rugosités importantes, la méthode DSI présentée dans la section suivante est appliquée pour la lisser.

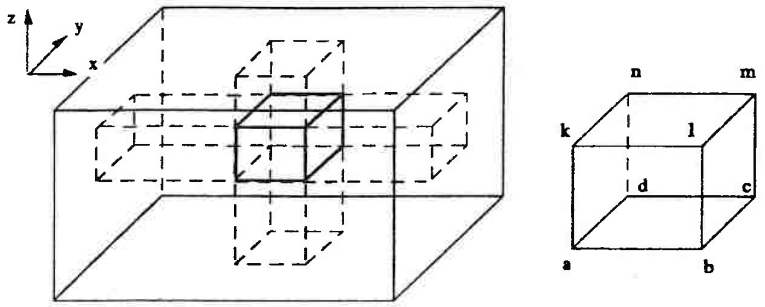


Figure 7.2 *Le domaine d'étude est partitionné en petits cubes élémentaires.*

7.2.1 Notion de surface isovaleur dans un gisement

Soit $t = \varphi(x, y, z)$ une fonction connue par ses valeurs aux noeuds d'une grille régulière couvrant un domaine parallélépipédique de l'espace 3D et soit t_0 une valeur donnée appelée "valeur de coupure". En supposant que la fonction $\varphi(x, y, z)$ n'est jamais constante dans le domaine d'étude, on définit alors la surface isovaleur $S(t_0)$ (voir [14]) comme l'ensemble des points de l'espace tels que :

$$(x, y, z) \in S(t_0) \Longleftrightarrow \varphi(x, y, z) = t_0$$

Nous proposons dans ce qui suit un algorithme de construction de $S(t_0)$ sous forme d'une T-surface en n'utilisant pour ce faire que les seules valeurs de $\varphi(x, y, z)$ connues aux noeuds de la grille régulière R couvrant le domaine d'étude.

7.2.2 Triangulation automatique de $S(t_0)$

Comme l'indique la Figure 7.2, afin de simplifier la discussion nous supposerons que le domaine d'étude est un parallélepipède rectangle ayant ses axes parallèles aux axes de coordonnées et que le réseau régulier couvrant ce domaine divise celui-ci en "cubes" élémentaires notés □. Par ailleurs, nous supposerons que les hypothèses suivantes sont toujours vérifiées :

- La grille R est suffisament fine pour que $S(t_0)$ intersecte un côté d'un cube élémentaire □ au plus une fois.
- Si deux sommets (x',y',z') et (x",y",z") d'un cube élémentaire □ satisfont:

  $$\varphi(x',y',z') > t_0 \text{ et } \varphi(x",y",z") < t_0$$

  alors ce cube est intersecté par $S(t_0)$.
- Si aucun côté d'un cube élémentaire □ n'est intersecté par $S(t_0)$, alors $S(t_0)$ est toute entière située à l'extérieur de ce cube.
- La surface $S(t_0)$ ne passe strictement par aucun sommet du réseau R (les sommets des cubes élémentaire □).

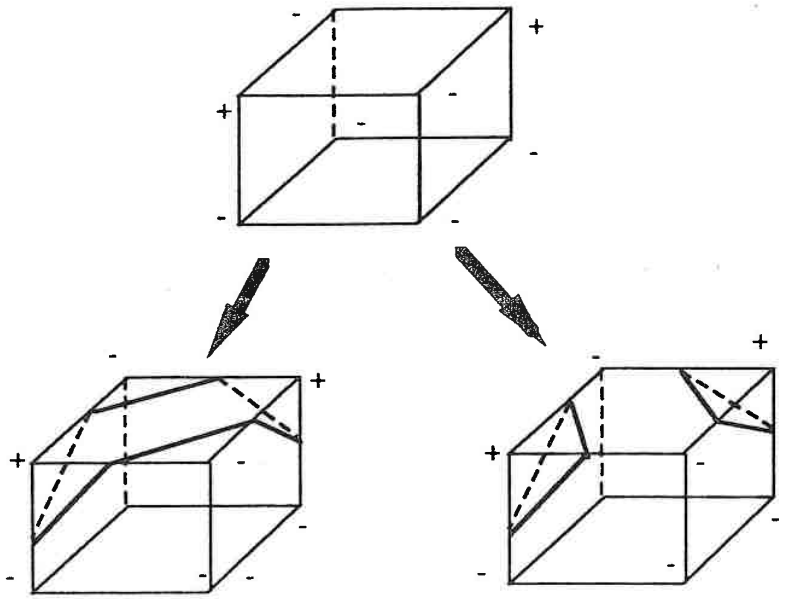


Figure 7.3 *Il y a deux possibilités de construire C car la face en haut de la cube $\square$ contient deux segments de C.*

Comme l'illustre la figure 7.2, soient $\{a, b, c, d, k, l, m, n\}$ les huit sommets d'un cube élémentaire $\square$ du réseau et soit P la partie de $S(i_0)$ située à l'intérieur de $\square$:

$$P = \square \cap S(i0)$$

Il est possible que P soit composé de plusieurs morceaux disjoints mais dans tous les cas, nous désignerons par C sa frontière constituée par l'ensemble des intersections des six faces $\{F_i(\square) : i = 1, 6\}$ de $\square$ avec $S(i_0)$:

$$C = \bigcup_{i=1}^{6} \{F_i(\square) \cap S(i_0)\}$$

Approximer $S(i_0)$ par une T-surface revient alors à approcher P par un morceau de cette T-surface pour chaque cube élémentaire $\square$ du réseau. Pour ce faire, on commence par supposer que la frontière C de P est en fait une ligne polygonale composée éventuellement de plusieurs morceaux.

Compte tenu des hypothèses simplificatrices formulées au paragraphe précédent, l'intersection d'une face $F_i(\square)$ de $\square$ avec $S(i_0)$ est au plus constituée de deux morceaux de courbes que l'on peut alors approcher par deux segments de droite; en opérant ainsi, on approche C par un ensemble de contours polygonaux. En pratique, comme l'indique la Figure 7.4, on peut recenser 13 cas topologiquement différents correspondant aux différents types de frontières C possibles (voir [34]); parmi ces cas possibles, nous distinguerons des cas simples et des cas complexes :

- Les cas simples sont ceux pour lesquels chaque face $_i(\square)$ du cube $\square$ contient au plus un segment de C. Le contour C est alors soit vide soit composé d'une seule courbe polygonale fermée.
- Les cas complexes sont ceux pour lesquels au moins une face $F_i(\square)$ du cube $\square$ contient deux segments de C. Le contour C peut alors être constitué de 1 ou 2 courbes polygonales fermées.

Dans les cas simples, le contour C peut être obtenu en joignant la paire de segments possédant un sommet commun. Dans les cas complexes, le problème majeur consiste à choisir la manière correcte pour construire les côtés de C lorsque la même face contient deux segments de C (Fig 7.4). Nous proposerons une méthode pour résoudre ce problème dans la section suivante.

Une fois le contour polygonal C déterminé, il ne reste plus qu'à contruire le ou les morceaux de T-surface $S(\square)$ tels que :

- C est la frontière de $S(\square)$,
- les sommets de $S(\square)$ sont ceux de C,
- $S(\square)$ est contenu dans $\square$,
- $S(\square)$ ne se recoupe pas elle même

Dans tous les cas, C comporte au maximum 12 sommets. C ne peut être qu'un contour simple comme présenté sur la figure 7.5 grâce au fait que P est un morceau de $S(t_0)$. Nous pouvons conclure qu'il existe toujours une T-surface $S(\square)$ satisfaisant les contraintes ci-dessus. On peut l'obtenir à l'aide de la procédure suivante (en pseudo-C) :

```
LIST_OF_TRIANGLES_t Triangulating_Contour(C)
{

   soit T la liste de triangles à construire;
   soient p_{i-1}, p_i et p_{i+1} les 3 sommets consécutifs de C;

   T = φ ;
   while( C n'est pas un triangle)
   {
      pour-tout(sommet p_i de C )
      {
         if( p_{i-1} et p_{i+1} ne sont pas situés dans la même face de F_i(□))
         {
            Construire le triangle T(p_{i-1},p_i,p_{i+1});
            Ajouter T(p_{i-1},p_i,p_{i+1}) à T: T+ = T(p_{i-1},p_i,p_{i+1});
            Supprimer le sommet p_i de C : C - = p_1;
         }
      }
```

```
        }

        Ajouter C à T ;

        return( T ) ;
    }
```

A l'issue de cette procédure, la T-surface S (voir Fig 7.6) est une T-surface satisfaisant les contraintes ci-dessus. En général, plusieurs T-surfaces peuvent être produites à partir d'un contour C donné; dans ce cas, nous proposons de choisir la T-surface ayant l'aire la plus petite comme solution (voir Fig 7.7).

7.2.3 Détermination de $F_i(\square) \cap S(t_0)$

Soit $F_i(\square)$ une face quelconque de $\square$ avec ses 4 sommets (k, l, m, n), soit $(H_i(\square)$ le plan contenant cette face et soit C_i l'intersection de $(H_i(\square)$ avec C. Comme le montre la Figure 7.8 (voir [18]), on peut considérer 3 cas topologiquement différents :

1. $S(t_0)$ traverse deux côtés opposés de $F_i(\square)$.
2. $S(t_0)$ traverse les deux côtés adjacents de $F_i(\square)$.
3. $S(t_0)$ traverse les 4 côtés de $F_i(\square)$.

Dans les deux premiers cas, C_i est constitué d'un unique segment de droite dont la construction est triviale mais, lorsque les quatre côtés sont traversés, il y a deux segments de droite dont la construction peut se faire de deux manières différentes. Dans ce dernier cas, comme nous allons le voir dans ce qui suit, la connaissance des variations de $\varphi(x, y, z)$ au voisinage de $\square$ est nécessaire pour supprimer toute ambiguité .

Supposons que l'on ait déterminé le point $\vec{p_0}$ où $S(t_0)$ coupe le côté kl comme indiqué sur la Figure 7.9 et donnons-nous un pas $h > 0$ petit par rapport à la taille de $\square_i$ considéré, on peut alors construire la suite $\{\vec{p_0}, \vec{p_1}, \ldots, \vec{p_n}\}$ de points telle que:

$$\begin{cases} p_i\vec{p_{i+1}} = h \cdot \frac{\vec{\theta_i}}{\|\vec{\theta_i}\|} \\ \\ avec \begin{cases} \vec{\theta_i} = \varepsilon \cdot \begin{bmatrix} -\frac{\partial\varphi(\vec{p_i})}{\partial y} \\ \frac{\partial\varphi(\vec{p_i})}{\partial x} \end{bmatrix} \\ \\ \varepsilon = \pm 1 \text{ tel que } : \vec{\theta_i} \cdot \vec{oy} > 0 \end{cases} \end{cases}$$

Dans cette expression, $\frac{\partial(\varphi(\vec{p_i}))}{\partial x}$ et $\frac{\partial(\varphi(\vec{p_i}))}{\partial y}$ désignent les dérivées partielles de $\varphi()$ au point $\vec{p_n}$ calculées à l'aide d'une méthode d'interpolation locale basée sur les valeurs de $\varphi()$ connues aux noeuds de la grille régulière.

Soit $\vec{p_n}$ le premier point de la suite ainsi définie à franchir la droite (kn) ou la droite (lm). En remarquant que la suite $\{\vec{p_0}, \vec{p_1}, \ldots, \vec{p_n}\}$ se situe approximativement sur la surface $S(t_0)$, nous

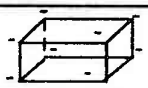
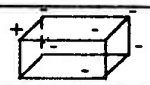
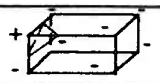
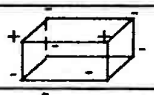
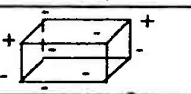
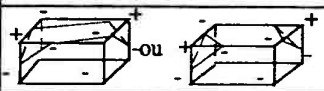
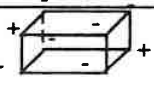
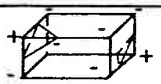
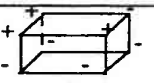
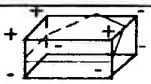
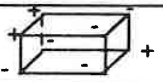
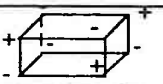
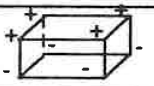
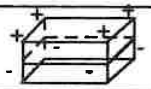
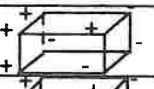
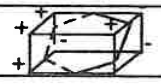
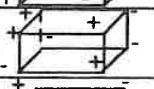
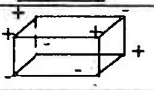
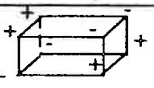
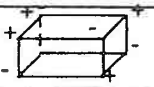
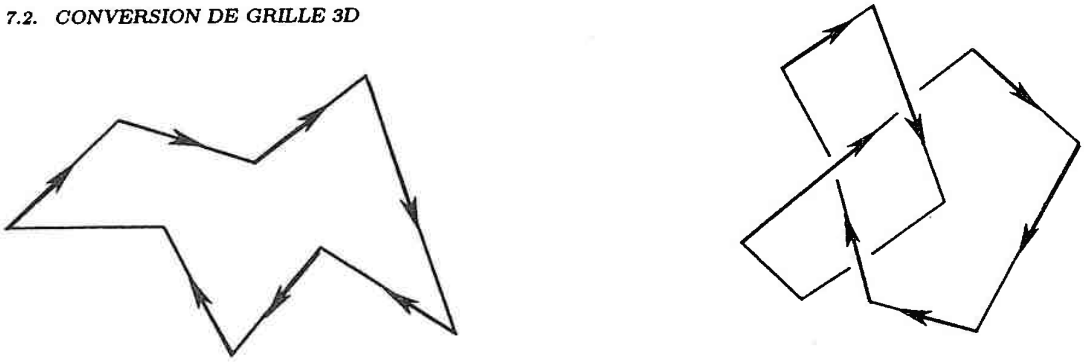
nombre de sommets +	configuration topologique	contour possible (c)	nombre de cas
0 (8)		C = 0	1 (1)
1 (7)			8 (8)
2 (6)			12 (12)
			12 (12)
			4 (4)
3 (5)			24 (24)
			24 (24)
			8 (8)
4			6 (6)
			8 (8)
			16 (16)
			24 (24)
			6 (6)
			2 (2)

Figure 7.4 *Génération de contours C en fonction des valeurs aux sommets d'un cube élémentaire* □. *Ces valeurs sont notées "+" lorsqu'elles sont supérieures à la valeur de coupure* t_0 *et "−" lorsqu'elles sont inférieures.*



c est un polygone simple en 3D

C ne peut pas être un contour tortu montre sur la figure

Figure 7.5 *C est toujours un contour simple et ne peut pas être un contour comme celui dans (b).*

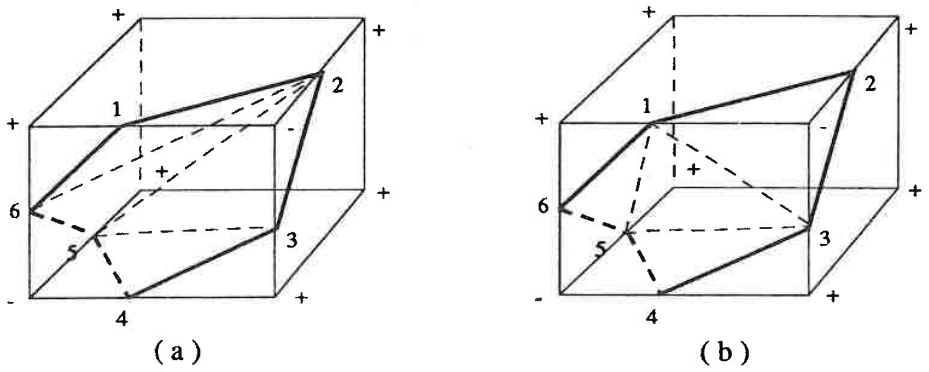


Figure 7.6 *(a) Une triangulation correcte de C; (b) Une triangulation incorrecte de C car les sommets 1 et 3 sont situés dans la même face de* □.

sommes maintenant en mesure de décider si l'on est dans le cas 1 ou dans le cas 2 représentés sur la figure7.9; en effet, il suffit de regarder si $\vec{p_n}$ se trouve à gauche de la droite (kn) ou à droite de de la droite (lm):

$$\begin{cases} \vec{p_n} \text{ est à gauche de } (kn) \Rightarrow \quad cas\ 1 \\ \vec{p_n} \text{ est à droite de } (lm) \Rightarrow \ cas\ 2 \end{cases}$$

On conçoit aisément que la décision ainsi prise aura d'autant plus de chance d'être exacte que la méthode d'interpolation locale utilisée pour déterminer $\varphi(\vec{p_n})$ fait intervenir plus de noeuds du réseau R.

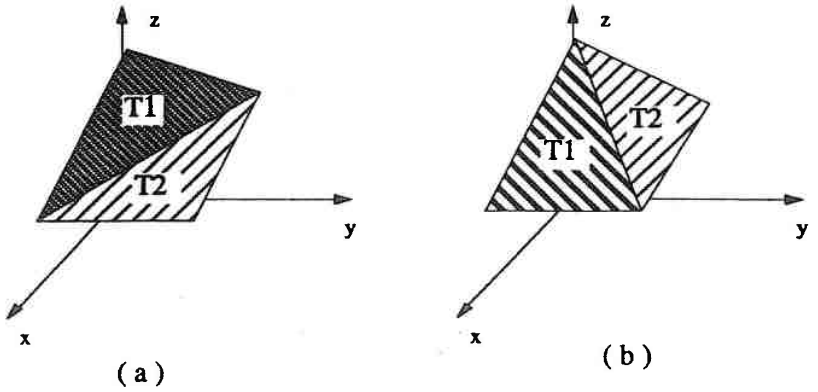


Figure 7.7 *L'aire de la T-surface obtenue dans (b) est plus petite que celle dans (a), donc la T-surface dans (b) est considérée comme meilleure.*

7.3 Lissage de la T-surface $S(t_0)$ par DSI

Dans certains cas, la T-surface $S(t_0)$ obtenue par la méthode présentée précédement peut avoir des rugosités locales (pouvant être très importantes). Pour les applications pratiques, ces rugosités présentent des inconvenients. Afin d'éviter celles-ci, on peut soit :

- découper la grille R plus finement.
- régulariser la fonction $t = \varphi(x, y, z)$.

Si on ne peut appliquer une des méthodes précédentes, on se propose d'utiliser une méthode de lissage appllée "Discrete Smooth Interpolation" (DSI voir [16]). Dans ce qui suit, nous présentons brièvement cette méthode et montrons quelques exemples de T-surfaces traitées par DSI.

7.3.1 Introduisant la méthode DSI

Supposons que les sommets de la T-surface $S(t_0)$ ont été numérotés de 1 à N et soit Ω l'ensemble de ces N sommets. Dans ce qui suit, nous allons identifier le sommet numéro "k" avec le "sommet k" :

$$\Omega = \{1, 2, \ldots, N\}$$

Pour un sommet quelconque "k" appartenant à Ω, nous définissons le voisinage $N(k)$ comme le sous-ensemble de Ω :

$$\alpha \in N(k) \Longleftrightarrow (\alpha, k) \text{ est le côté d'un triangle de } S(t_0)$$

le vecteur joignant l'origine de l'espace 3D et un sommet donné k est noté $\vec{\varphi_k}$ et φ est la collection de tous ces sommets :

$$\varphi = \{\vec{\varphi_1}, \vec{\varphi_2}, \ldots, \vec{\varphi_N}\}$$

la méthode DSI est basée sur un critère de rugosité locale $R(\varphi|k)$ définie à chaque noeud $k \in \Omega$:

$$R(\varphi|k) = || \sum_{\alpha \in N(k)} v^{\alpha}(k) \cdot \vec{\varphi_{\alpha}} ||^2$$

les coefficients $\{v^{\alpha}(k)\}$ sont des pondérateurs donnés. Pour l'étude présentée dans cette section, nous avons choisi ces coefficients de la façon suivante :

$$\left[\begin{array}{lll} v^{\alpha}(k) & = & \left\{ \begin{array}{lll} -|\Lambda(k)| & \text{si} & \alpha = k \\ 1 & \text{si} & \alpha \in \Lambda(k) \end{array} \right. \\ \\ \textit{avec} & : & \left\{ \begin{array}{lll} \Lambda(k) & = & N(k) - \{k\} \\ |\Lambda(k)| & = & \text{le nombre d'éléments de } \Lambda(k) \end{array} \right. \end{array} \right.$$

Le critère de rugosité locale $R(\varphi|k)$ est utilisé pour construire une rugosité globale $R(\varphi)$ tel que :

$$R(\varphi) = \sum_{k \in \Omega} R(\varphi|k)$$

En pratique, la T-surface $S(t_0)$ doit respecter un ensemble de contraintes données et ce qui introduit une contrainte sur l'ensemble φ des solutions admissibles; la méthode DSI suppose que chacune de ces contraintes peut être exprimée de la manière suivante :

$$C_i(\varphi) = 0$$

Le but de la méthode DSI est de rechercher l'ensemble φ des sommets qui minimise $R^*(\varphi)$ tel que :

$$R^*(\varphi) = \sum_{k \in \Omega} R(\varphi|k) + \sum_i \varpi_i^2 \cdot |C_i(\varphi)|^2$$

les coefficients ϖ_i^2 sont appelés "facteurs de certitude", et ils sont donnés pour moduler l'importance relative de chaque contrainte.

7.3.2 Lissage d'une T-surface

Soit $S^0(t_0)$ la surface initiale produite par notre algorithme et soit φ^0 la collection de ses sommets associés $\vec{\varphi_k^0}$:

$$\varphi^0 = \{\vec{\varphi_1^0}, \vec{\varphi_2^0}, \ldots, \vec{\varphi_N^0}\}$$

nous nous proposons de transformer cette surface en une T-surface lisse $S(t_0)$ qui est proche de la $S^0(t_0)$; autrement dit, chaque noeud φ_i de $S(t_0)$ doit satisfaire les contraintes suivantes :

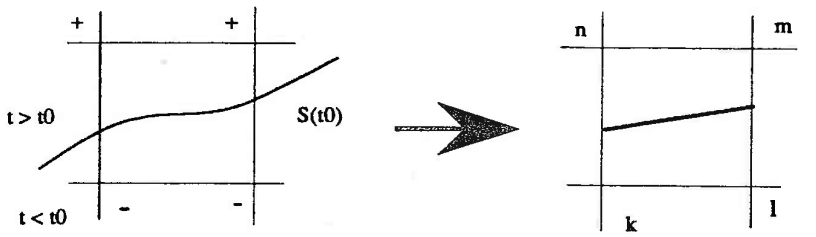
$$\vec{\varphi_i} \simeq \vec{\varphi_i}^0$$

ceci nous suggère de choisir $R^*(\varphi)$ tel que :

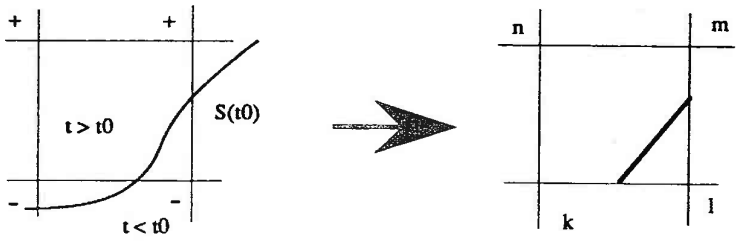
$$R^*(\varphi) = \sum_{k \in \Omega} R(\varphi|k) + \sum_{i \in \Omega} \varpi_i^2 \cdot ||\vec{\varphi_i} - \vec{\varphi_i^0}||^2$$

La minimisation de $R^*(\varphi)$ permet de calculer la position des sommets de la T-surface lisse $S(t_0)$.

On trouve sur la Figure 7.10 un exemple de lissage de surface $S(t_0)$ obtenu à l'aide de DSI.



S(t0) coupe deux cotes opposes



S(t0) deux cotes adjacents

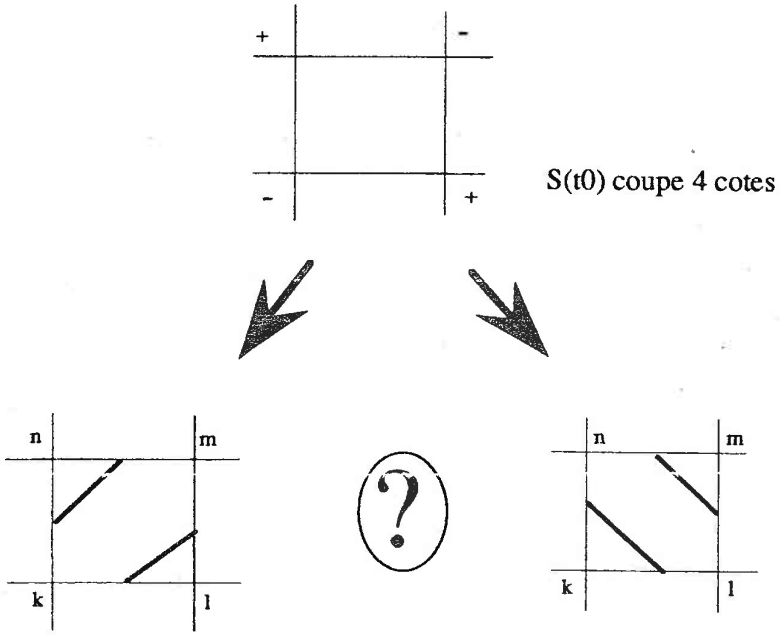


Figure 7.8 *Intersection d'une face $F_i(\square)$ avec la surface isovaleur $S(t_0)$ et génération du segment de droite correspondant appartenant à C.*

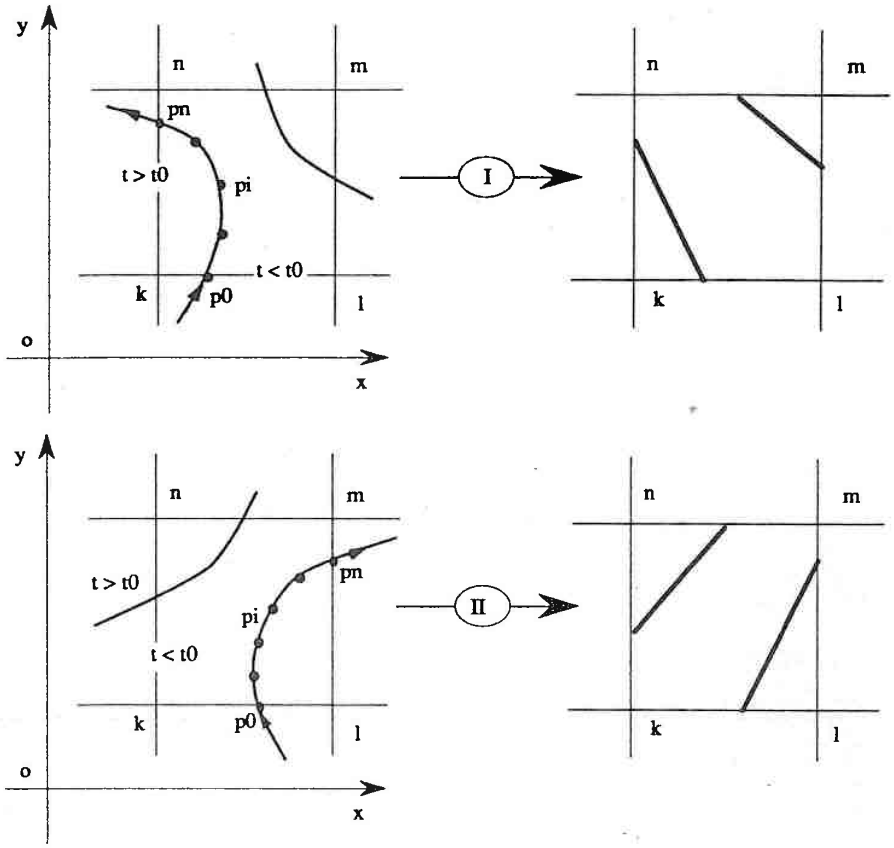
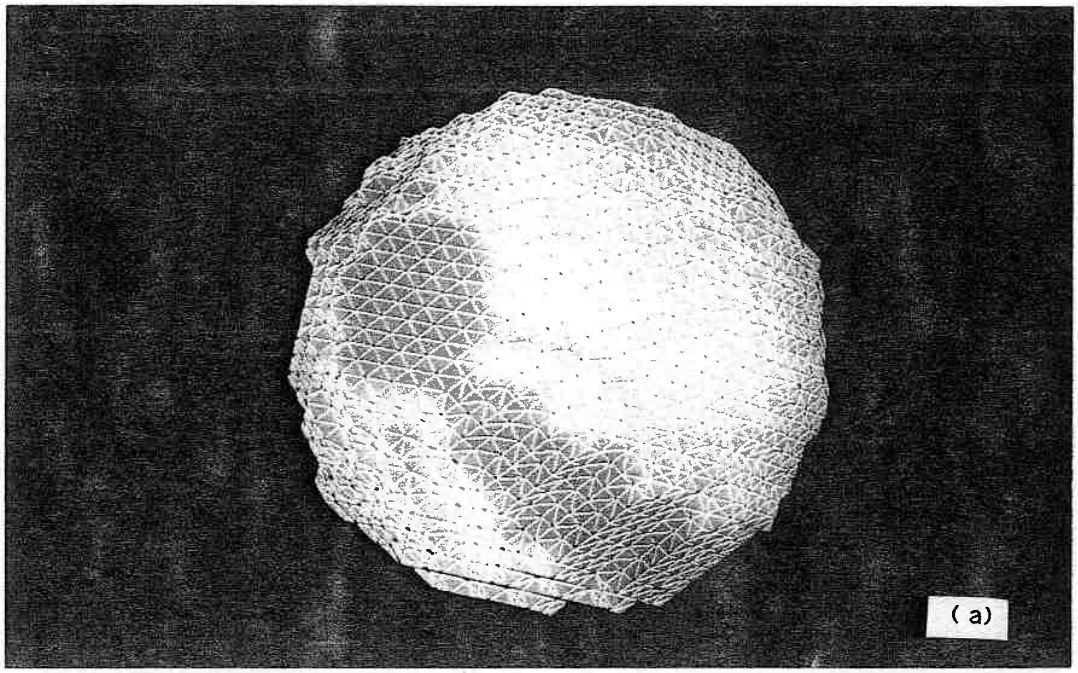


Figure 7.9 *(a) $\vec{p_n}$ est à gauche de (kn); (b) $\vec{p_n}$ est à droite de (lm).*



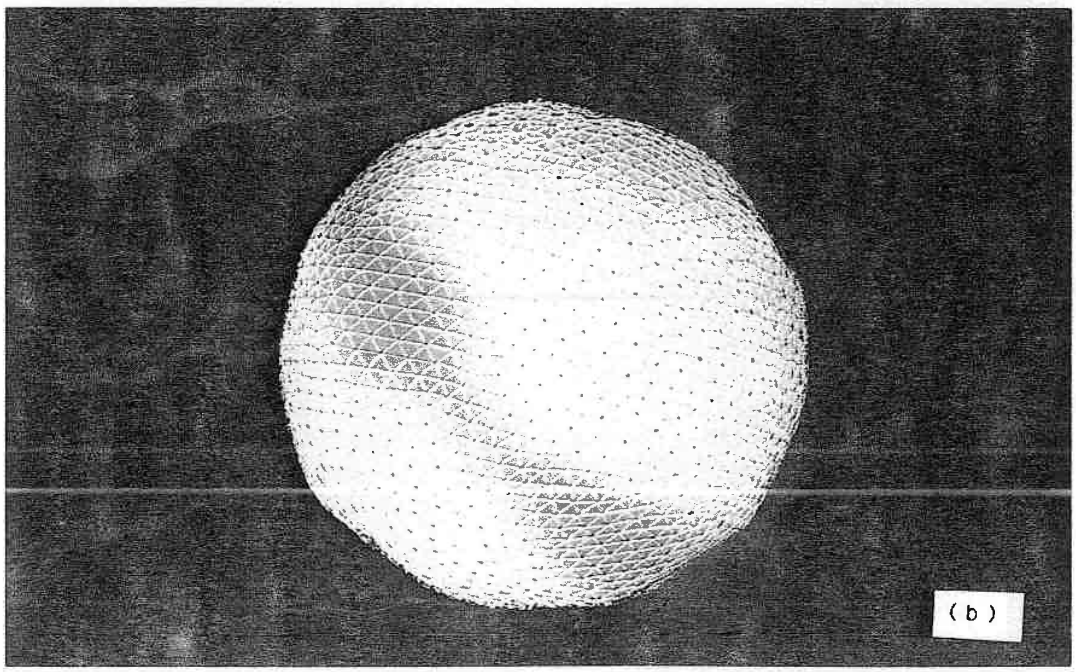
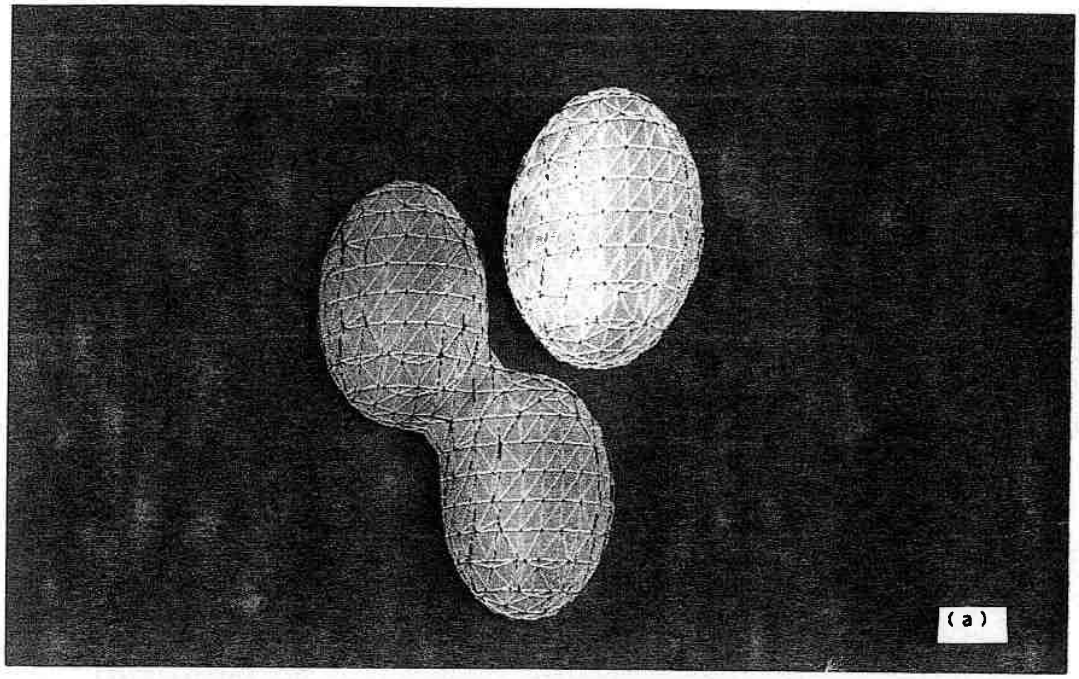


Figure 7.10 *(a) Exemple de surface isovaleur obtenue à l'aide de notre méthode. (b) la même surface après lissage à l'aide de la méthode DSI.*



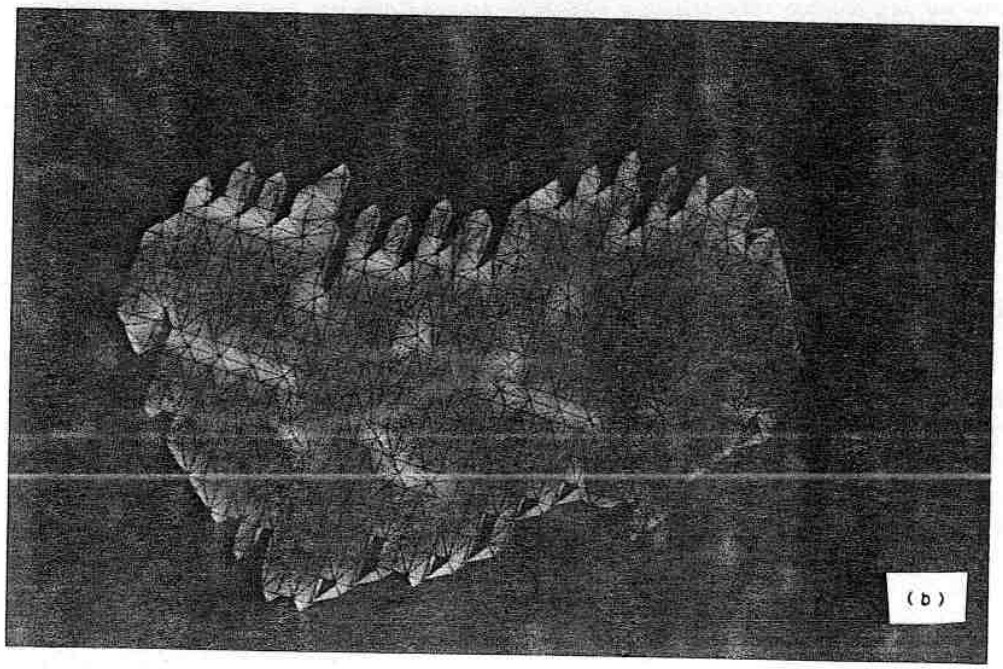


Figure 7.11 *(a) Surface isovaleur correspondant à une fonction mathématique. (b) Surface isovaleur correspondant à la peau d'un gisement.*

Chapitre

8

Reconstruction de forme à partir des coupes sériées

8.1 Introduction

Dans beaucoup d'applications, un objet ou un ensemble d'objets sont connus par un ensemble de coupes sériées. Par exemple, en médecine, les coupes sériées sont constituées par des images planes obtenues en déplacant un appareil d'échographie ou de rayons-X dans une direction donnée. Ici la structure à reconstruire apparait sous la forme de contours correspondant à l'intersection de la peau de l'objet avec les plans d'acquisition des données.

Parmi les méthodes proposées dans la littérature, on distingue deux approches différentes :

- Méthode de Kappel et Pauchon(voir [28]) qui réduit le problème en construisant une série de T-surface, chacune est entre deux sections adjacentes.
- Méthode de Boissonnat (voir [2]) qui construit directement le volume de l'objet au lieu de construire sa peau.

Dans la première approche, la surface entre deux sections adjacentes est constituée par des facettes triangulaires, chacune d'elle étant définie par deux points d'un contour (dans une section) et un point dans l'autre contour (l'autre section adjacente). La construction de cette surface est équivalente à la recherche d'un chemin dans un graphe. Si l'on associe un poids à toutes les arêtes du graphe, un algorithme classique de la recherche du plus court chemin peut donner une construction optimale. Plusieurs méthodes pour définir le poids sont proposées :

- basée sur le volume entouré par la surface.
- basée sur l'aire de la surface.
- basée sur la longueur de ses arêtes joignant les deux contours.

- etc ...

Grâce à l'existence de ces critères, cette méthode devient très utile dans un nombre considérable d'applications. Néanmoins, cette méthode ne peut pas fonctionner lorsque la différence de deux contours devient grande. Un cas extrême est un contour convexe dans un plan, un contour spiral dans un autre plan. Le résultat va être une T-surface qui se croise avec elle-même.

La méthode de Boissonnat est une approche toute nouvelle. Elle peut traiter les cas où l'objet contient de multiple contours et le nombre de contours varie d'une section à l'autre. Différente de la première approche, cette méthode ne construit pas directement la peau de l'objet mais son volume. Il est naturel qu'une fois le volume de l'objet construit, sa peau (une T-surface) peut être obtenue facilement en cherchant la frontière du volume de l'objet. Cette méthode utilise la triangulation de Delaunay (voir [23]), elle permet de traiter les objets multiples et les objets avec des trous, sans jamais produire une T-surface qui se recoupe par elle-même.

Dans ce chapitre, nous présentons brièvement la méthode de Pauchon (voir [28]) avec le critère basé sur la longueur des arêtes entre les contours et ensuite l'approche de Boissonnat.

8.2 Algorithme de Pauchon

8.2.1 Représentation du problème et résolution générale

Soit $L_1 = \{p_1, \ldots, p_m\}$ la liste ordonnée des points d'un contour simple situé dans un plan; de même soit $L_2 = \{q_1, \ldots, q_n\}$ la liste ordonnée des points d'un contour simple situé dans un plan parallèle au premier (cf Figure 8.1).

Dans un premier temps, on suppose que la triangulation cherchée doit contenir les arêtes p_1q_1 et p_mq_n. Pour obtenir une représentation pratique de l'ensemble des triangulations possibles entre L_1 et L_2, on définit un graphe $G = \{C_{ij}, N\}$ associé à L_1 et L_2 (cf Figure 8.1). Les noeuds de ce graphe noté C_{ij} représentent un segment p_iq_j joignant le point p_i de L_1 et q_j de L_2, un arc à (C_{ij}, C_{ik}) de G qui joint C_{ij} à C_{ik} représente le triangle $T(p_i, q_j, q_k)$.

La triangulation Γ cherchée doit respecter les règles suivantes :

- Tout point d'un contour doit être relié au moins à un point de l'autre contour, autrement dit, une triangulation est représentée par un chemin dans G qui part de C_{11} et va en C_{mn} en passant au moins une fois sur chaque ligne et chaque colonne.
- Si le segment p_iq_j appartient à Γ, alors l'un des deux segments $p_{i+1}q_j$ ou p_iq_{j+1} doit appartenir aussi à Γ. L'interprétation de cette règle sur G est :

  tout noeud C_{ij} de G doit être relié par une arête soit au noeud C_{ij+1}, soit au noeud C_{i+1j}.
- Les arêtes de Γ ne doivent pas se croiser. Si un chemin dans G représentant une triangulation de L_1 et L_2 passe par C_{ij} et C_{i+1j}, alors il ne peut pas passer par C_{ij+1}, d'où l'orientation donnée aux arcs de G.

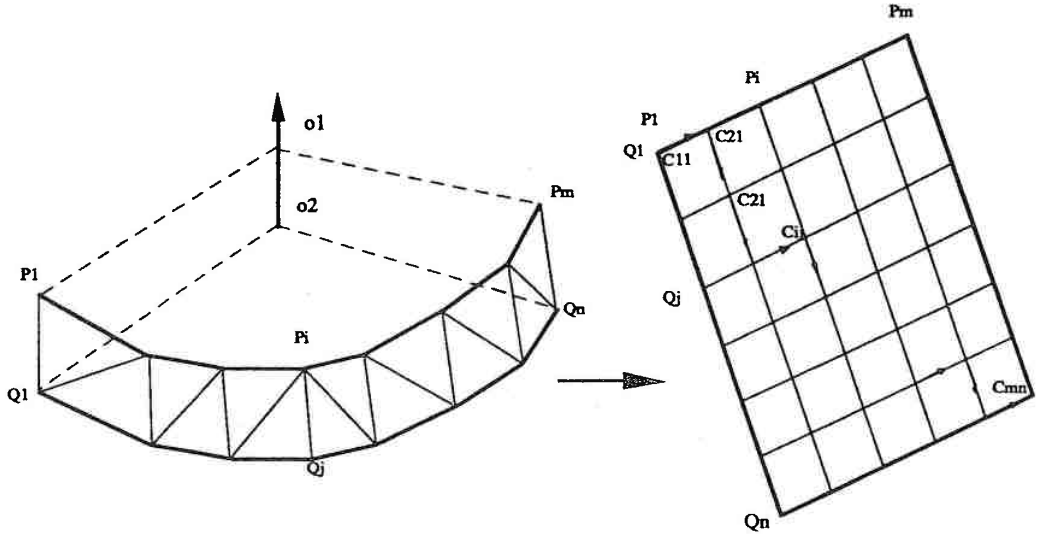


Figure 8.1 *La triangulation se réduit à la recherche d'un plus court chemin dans un graphe.*

Le graphe G permet donc de donner une interprétation très simple du problème initial en terme de théorie des graphes : toute triangulation admissible entre L_1 et L_2 est représentée par un chemin dans G joignant C_{11} et C_{mn}, on peut aussi montrer que le nombre total de triangulation est $\mathcal{T}(m,n) = \frac{((m+n-2)!}{(m-1)!(n-1)!}$. L'idée est donc d'associer un coût aux arcs de G et de chercher le plus court chemin au sens de ce coût entre C_{11} et C_{mn}. Pour que la triangulation correspondant à ce plus court chemin soit celle qui représente le mieux la portion de surface entre les contours L_1 et L_2, il faut que le coût soit bien choisi.

Pauchon a constaté qu'en minimisant la somme des longueurs des arêtes qui relient un contour à l'autre, on obtient des triangulations très satisfaisantes que le contour soient connexe ou non. Le coût associé à chaque arête du type (C_{ij}, C_{ij+1}) ou (C_{ij}, C_{i+1j}) a donc été choisi égal à la longueur du segment $p_i q_{j+1}$ ou $p_{i+1} q_j$.

8.2.2 Cas où les deux coupes ont le même nombre de composantes

Soient $\{C_{11}, C_{12}, \ldots, C_{1n}\}$ les contours de la coupe 1, $\{C_{21}, C_{22}, \ldots, C_{2n}\}$ ceux de la coupe 2 et G_{ji} l'équibarycentre des points du contour $C_{ji} (j = 1, 2; i \in [1, n])$. Pour i allant de 1 à n, on associe le contour C_{1i} au contour C_{2k} de la coupe 2 dont le centre de gravité est le plus proche de celui de C_{1i}. On triangule C_{1i} et C_{2k} par la méthode décrite ci-dessus.

Le seul cas où cette méthode donnerait des résultats absurdes est celui où G_{1i} et G_{1k} auraient un même plus proche voisin dans $\{G_{2i} | i \in [1.n]\}$. Ce cas est peu probable dans les applications tant que la distance entre deux plans de numérisation successives n'est pas très grande. Or cela est toujours vrai si l'on veut avoir une numérisation suffisemment précise de l'objet (cf La Figure 8.2).

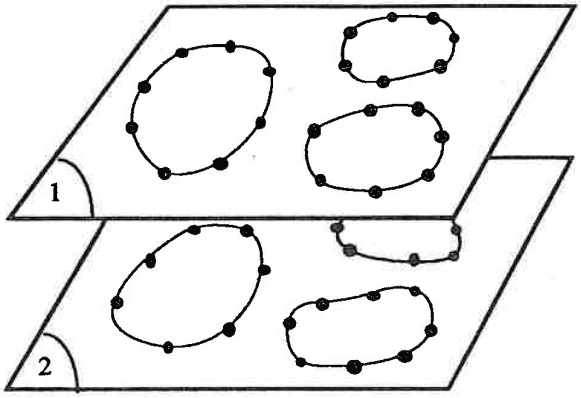


Figure 8.2 *Les deux coupes contiennent le même nombre de contours.*

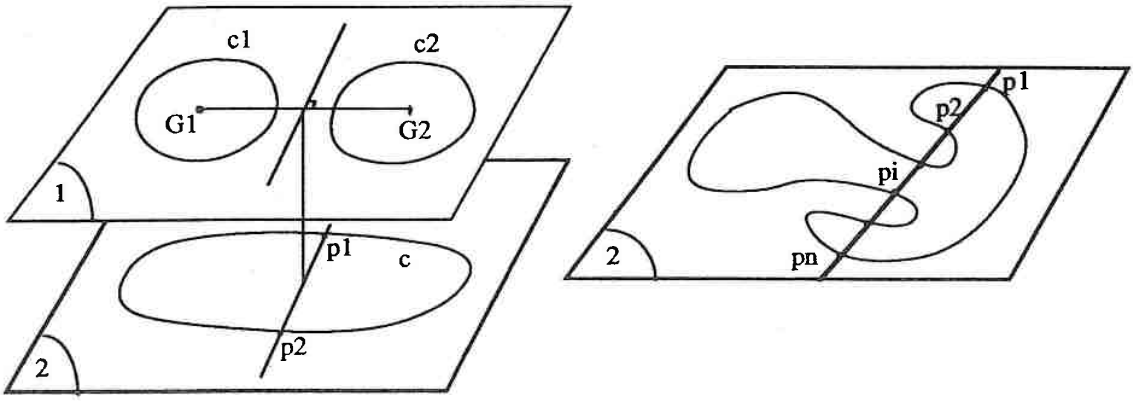


Figure 8.3 *Les deux coupes contiennent un nombre différent de contours.*

8.2.3 Cas où l'une des coupes a une composante connexe et l'autre a n composantes connexes

Cette situation est représentée dans la Figure 8.3.a où une coupe est constituée d'un seul contour et la suivante de deux. Dans ce cas relativement simple, une idée naturelle est d'essayer de diviser le contour C de la coupe 2 en deux sous-contours S_c1 et S_c2 et de se ramener ainsi au cas que l'on vient de voir. Pour diviser le contour C, une méthode simple est la suivante :

Soient P le plan médian de G_{11} et G_{12} et T l'intersection de P avec le plan de la coupe 2. T intersecte C en deux points P_1 et P_2: il est alors facile d'en déduire S_c1 et S_c2.

Il peut cependant se produire que T coupe C en plusieurs points comme indiqué sur la Figure 8.3.b. Dans ce cas la décomposition de C en deux sous-contours reste faisable, mais elle est plus délicate à mettre en oeuvre, surtout dans la généralisation à un nombre n de

contours ($n > 1$) : en effet, cette généralisation conduit à chercher, d'une manière analogue, une décomposition de C en n sous-contours qui doivent correspondre aux n contours de la coupe suivante. La médiatrice T utilisée dans l'exemple ci-dessus ($n = 2$), doit être remplacée par le diagramme de Voronoï associé aux équibarycentres (G_i) $n_i = 1$ des n contours. Ce diagramme définit une partition du plan en n régions. En projetant cette partition sur le plan de la coupe n'ayant qu'une composante connexe, on obtient la superposition du diagramme et du contour. Moyennant certaines manipulations analogues à celles représentées sur la Figure 8.3.b, dans le cas où $n = 2$, on peut en déduire une décomposition du contour C en n sous-contours.

8.3 Algorithme de Boissonnat

8.3.1 Position du problème

Supposons que l'on ait un ou plusieurs objets dans l'espace 3D, qui sont intersectés par un nombre fini de plans spécifiques, pas forcément parallèles. Les seules informations connues de ces objets sont les intersections de leurs peaux avec les plans. Chaque intersection peut être représentée par un nombre fini de courbes fermées et orientées qui séparent l'intérieur et l'extérieur des objets. Ces courbes sont approximées par des lignes polygonales fermées appellées contours. Soient C l'ensemble des ces contours et M l'ensemble des sommets de C. La reconstitution de l'objet se fait en construisant le volume, dont le bord est obtenu tranche par tranche à partir de la triangulation de Delaunay associée à deux sections adjacentes. Dans ce cas particulier la triangulation de Delaunay peut être calculée de façon efficace en ne faisant intervenir que des opérations 2D.

Soient P_1, P_2 deux sections adjacentes et C_1, C_2 deux contours situés respectivement dans P_1 et P_2 (C_1 et C_2 peuvent être constitués de plusieurs composantes connexes). Dans ce qui suit, C_1 et C_2 sont considérés comme étant contenus dans la triangulation de Delaunay de leurs sommets.

8.3.2 Construction du volume convexe

Soient M_i ($i = 1, 2$) le sous-ensemble des points M situés dans le plan P_i, DT_i (resp V_i) la triangulation de Delaunay (resp diagramme de Voronoï) de M_i dans le plan P_i et DT la triangulation de Delaunay de $M = M_1 + M_2$ en 3D.

Proposition 1

L'intersection de DT avec le plan P_i ($i = 1, 2$) est identique à DT_i.

Une importante conséquence de cette proposition est que le contour C_i va être contenu dans DT si et seulement si C_i est contenu dans DT_i. Cette condition est satisfaite lorsqu'il existe un ensemble de cercles dans P_i ($i = 1, 2$) tel que chaque cercle passe par les sommets de deux côtés consécutifs sans contenir aucun autre sommet de C_i (cf Figure 8.4). Si cette condition n'est pas vérifiée, il suffit d'ajouter de nouveaux points comme l'affirme la proposition suivante.

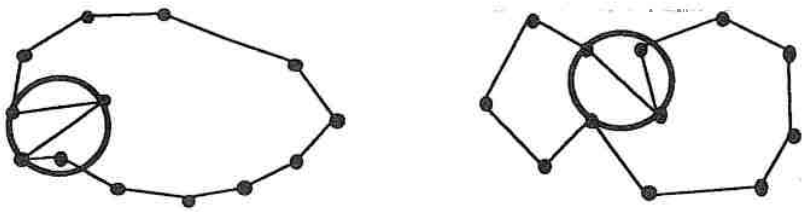


Figure 8.4

Proposition 2

Chaque polygone simple est géométriquement identique à un polygone contenu dans la triangulation de Delaunay de ses sommets et obtenu en ajoutant au plus un nombre fini de points sur certains côtés du polygone initial.

Grâce à ces deux propositions, maintenant on peut supposer que les contours de C_i $(i = 1, 2)$ sont contenus dans la triangulation DT_i. La méthode de la triangulation de Delaunay a été présentée auparavant (voir chapitre 3.1), dans la suite, on détaille la construction de DT une fois DT_1 et DT_2 connues.

Pour ne pas alourdir le problème, on suppose que les deux plans P_1 et P_2 sont parallèles. Les tétraèdres de DT sont de deux types (cf Fig8.5) :

- Le tétraèdre T_i qui a 3 sommets dans $P_i (i = 1, 2)$ et par conséquent dû à la proposition 1 ces 3 sommets sont également les sommets d'un triangle t de DT_i; le quatrième sommet de T est le point de $M_j (i \neq j)$ qui est le plus proche du centre du cercle circonscrit à t .
- Le tétraèdre T_{12} qui n'a pas de face contenue dans P_1 et P_2 mais posséde un côté contenu dans P_1 (resp. et P_1) et qui fait partie de DT_1 (resp. DT_2).

La proposition suivante réduit le problème du calcul de ces tétraèdres à celui de la recherche d'intersections des segments dans un plan.

Proposition 3

Un côté e_1 de DT_1 et un côté e_2 de DT_2 appartiennent à un tétraèdre de DT **si et seulement si** leurs côtés duals dans le diagramme de Voronoï s'intersectent lorsque l'on projette perpendiculairement P_1 sur P_2.

Considérons le graphe $G(M_1, M_2)$ qui est l'union de deux diagrammes de Voronoï obtenus après projection de P_1 sur P_2 suivant une direction orthogonale à ces plans. Un noeud de ce graphe est un sommet V_i de l'un des deux diagrammes de Voronoï V_1 et V_2 ou l'intersection entre un côté de V_1 et un côté de V_2. Toutes les informations concernant la structure de la triangulation

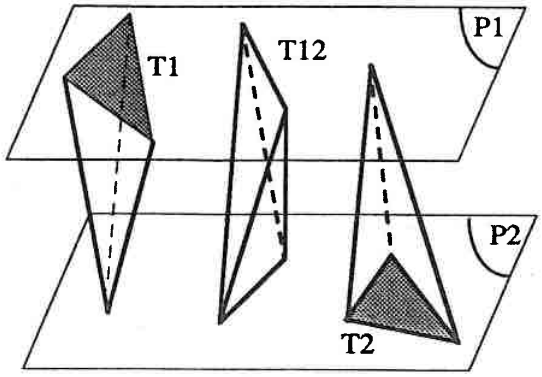


Figure 8.5 *Trois types de tétraèdres.*

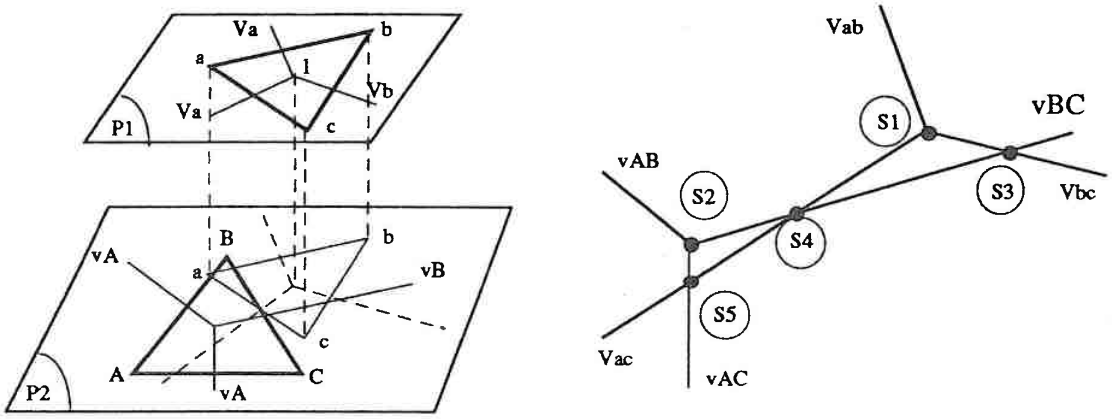


Figure 8.6 *Génération des tétraèdres à partir de deux contours qui se réduisent à deux triangles.*

de Delaunay 3D de $M_1 \cup M_2$ peuvent être déduites de ce graphe. A chaque noeud de ce graphe correspond un tétraèdre tel que :

- Si le noeud est un sommet de V_1 (ou V_2), il est alors le centre d'un cercle circonscrit sur un triangle noté $T(abc)$ de DT_1 (resp DT_2). Lorsque l'on projete P_1 sur P_2, ce noeud est projeté à l'intérieur d'une cellule de V_2 (resp V_1) associée à un point de M_2 (resp M_1) noté B. Ce noeud correspond à un tétraèdre $T(abcB)$ de type T_1 (resp T_2). Considérons l'exemple simple de la Figure 8.6. Dans les deux plans P_1 et P_2, C_1 et C_2 se réduisent à deux triangles $t_1 = T(abc)$ et $t_2 = T(ABC)$ et le centre du cercle circonscrit t_i est I_i. DT_i contient évidement t_i. Notons V_{ab}, V_{bc}, V_{ca}, V_{AB}, V_{BC}, V_{CA} les côtés du diagramme de Voronoï dual aux côtés ab, bc, ca, AB , BC, CA respectivement. Dans cet exemple, on obtient un tétraèdre $S_1 = (abcB)$ de type T_1, et un tétraèdre $S_2 = (aABC)$ de type T_2.

- Si le noeud est l'intersection de deux côtés v_1 dual de e_1 de DT_1 et v_2 dual de e_2 de DT_2, il correspond à un tétraèdre de type T_{12} contenant e_1 et e_2. Dans l'exemple de la Figure 8.6, on obtient les 3 tétraèdres suivants :
  - V_{bc} intersecte V_{BC} produit le tétraèdre $S_3 = (bcBC)$.
  - V_{ca} intersecte V_{BC} produit le tétraèdre $S_4 = (acBC)$.
  - V_{ac} intersecte V_{AC} produit le tétraèdre $S_5 = (acAC)$.

La relation d'adjacence entre ces tétraèdres est donnée par le graphe $G(M_1, M_2)$. Par exemple, S_4 posséde 4 voisins S_1, S_2, S_3 et S_5, S1 a deux voisins S_3 et S_4 et donc deux faces sur le bord de la triangulation DT (qui est l'enveloppe convexe de M).

Remarques (Grâce à la proposition 3)

- DT est indépendant de la distance des deux plans. La méthode de calcul de DT par la proposition 3 peut alors éviter beaucoup de problèmes numériques.
- Le nombre de tétraèdres de DT, dans le pire cas, est de l'ordre $O(n * n)$.

8.3.3 Adaptation du volume convexe au contour

position du problème

Soit O un objet (un ensemble d'objets va aussi être appellé un objet), la seule information connue de cet objet est un ensemble de L sections qui sont les intersections de l'objet avec L plans. Supposons que dans chaque section, l'objet est constitué par des régions polygonales simples, certaines pouvant être situées à l'intérieur des autres. L'objectif est de trouver un objet O' dont les intersections avec les L plans de coupe sont exactement les sections de coupes (ceci évite l'objet ayant une épaisseur nulle). Le problème de la construction de O' se réduit à la construction de $L-1$ formes partielles, chacune joignant deux sections de coupe C_i et C_{i+1} situées dans deux plans adjacents. La méthode consiste à calculer la triangulation de Delaunay des points M_i de C_1 et M_{i+1} de C_2 qui est un solide convexe. Pour modéliser la portion O_{12} de l'objet O entre P_1 et P_2, il est nécessaire d'éliminer certains tétraèdres . On applique cette procédure itérativement pour toutes les paires de plans adjacents; à la fin, on obtient un ensemble de tétraèdres O' qui doit satisfaire les conditions suivantes :

1. O' est un solide. Un solide est défini comme un ensemble de tétraèdres (ou l'union d'ensemble disjoints) connectés face à face et sa frontière est un polyèdre simple.
2. O' intersecte les plans de coupe suivant les sections de coupe données.

Marquer les arêtes dans les sections de coupe

Chaque section de coupe est supposée avoir une face dessus et une face dessous, la face dessus du plan P_i est vue dans la face dessous du plan adjacent P_{i-1}. Par convention, les normales $\vec{n_i}$ à ces plans sont orientées dans la direction dessous dessus.

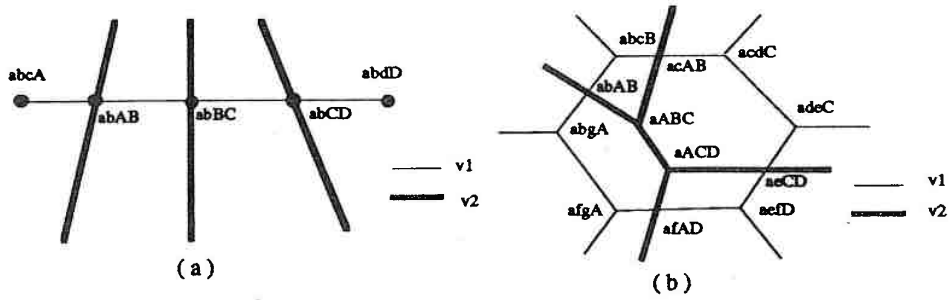


Figure 8.7 *Recherche des tétraèdres internes.*

Pour séparer l'intérieur et l'extérieur de l'objet les contours sont orientés dans le sens des aiguilles d'une montre autour des normales des plans correspondants. On peut distinguer facilement si une arête M_iM_j se trouve à l'intérieur, à l'extérieur ou sur les contours de la façon suivante :

- M_iM_j est à l'intérieur, si M_iM_{i-1}, M_iM_j et M_iM_{i+1} sont dans le sens inverse des aiguilles d'une montre.
- M_iM_j est à l'extérieur, si M_iM_{i-1}, M_iM_j et M_iM_{i+1} sont dans le sens des aiguilles d'une montre.
- M_iM_j est sur les contours, si $M_{i+1} = M_j$.

Recherche des tétraèdres internes

Rappelons que toutes les informations concernant DT sont contenues dans le graphe $G(M_1, M_2)$ défini précédement (noté G). Ce graphe est obtenu en superposant les deux diagrammes de Voronoï DV_1 de M_1 et DV_2 de M_2. Chaque noeud de G représente un tétraèdre de DT et deux noeuds sont adjacents si et seulement si ils représentent un tétraèdre ayant une face commune. G posséde les 3 propriétés suivantes :

- L'ensemble de tous les tétraèdres de DT partageant une arête e dans P_i ($i = 1, 2$) est représenté par tous les noeuds de G appartenant à l'arête de v_i dual à e (cf Fig 8.7.a).
- L'ensemble de tous les tétraèdres de DT partageant un sommet commun m de P_i est représenté par tous les noeuds $n(m)$ de G appartenant à la frontière et tombant à l'intérieur de la cellule de v_i autour de m (cf Fig 8.7.b).
- L'ensemble de tous les tétraèdres de DT partageant une arête m_1m_2 avec $m_1 \in P_1$ et $m_2 \in P_2$ est représenté par les noeuds de G appartenant à un cycle élémentaire de G noté $G(m_1, m_2)$.

Si un tétraèdre de type T_1, T_2 ou T_{12} contient une arête dans le plan P_1 ou dans P_2 qui est à l'extérieur des contours donnés, ce tétraèdre n'appartient alors pas au modèle S_{12}, sinon,

la condition 2 ci-dessus ne serait pas vérifiée. Dans le graphe G, ceci implique que l'on doit supprimer tous les noeuds représentant les tétraèdres contenant les arêtes qui sont à l'extérieur des contours et les arêtes qui sont adjacentes à ces noeuds dans G (règle 1). En effectuant ce procédé, G devient G_1. Une fois cette règle appliquée, il reste à satisfaire la condition 1.

On doit d'abord éviter la situation où une portion de la triangulation D_1 est connectée (même localement) seulement avec une portion de D_2 à une k-dimension (k = 0,1). Ce genre de connection va produire des liens étranges qui engendrent une tranche non solide. Heureusement on peut vérifier ce comportement sur chaque arête interne ou sur la frontière d'un contour donné. Dans la triangulation DT, cette arête e (du plan P_2) est contenue dans une série de tétraèdres se joignant face-à-face commençant par un tétraèdre de type T_2, ensuite un ou plusieurs tétraèdres de type T_{12}. Si e est à l'intérieur d'un contour P_2, alors cette série va se terminer dans un seul tétraèdre de type T_2; sinon, elle se termine dans un tétraèdre intermédiaire de type T_{12} ayant une face sur la frontière de DT. Pour obtenir une représentation de solidité de O_{12}, on doit être sûr que les tétraèdres qui sont gardés (attachés à cette arête e dans S_{12}) sont encore connectés face-à-face à au moins un des deux tétraèdres possibles de type T_2, sinon, la région locale du contact de cette portion du modèle avec le plan P_2 va être unidimensionnelle, il n'y a donc pas de solide. De cette manière, on arrive à un critère décrivant les tétraèdres intermédiaires qui doivent être éliminés. Pour un tétraèdre S de type T_{12} qui est gardé après l'exclusion des tétraèdres ayant une arête à l'extérieur de C_1 ou C_2 (règle 1), on effectue le test suivant :

Supposons que S ait deux arêtes $e \in P_1$ et $f \in P2$, ces arêtes étant à l'intérieur ou sur le bord. On vérifie si S est connecté (à travers les tétraèdres gardés) face-à-face à au moins un des deux tétraèdres possibles de type T_1 et connecté aussi à au moins un des deux tétraèdres possibles de type T_2. Si ceci est vrai, alors S appartient à O_{12}, sinon S doit être rejeté (règle 2).

La solidité ci-dessus peut aussi être vérifiée dans le graphe G_1. Soit n un noeud de type T_{12} de G_1 situé à l'intersection d'une arête v_1 de V_1 et d'une arête v_2 de V_2; soit S le tétraèdre correspondant. S appartient à O_{12} si et seulement si n est connecté avec un noeud de type T_1 à travers une série d'arêtes appartenant à v_1 au moins, et connecté avec un noeud de type T_2 à travers une série d'arêtes appartenant à v_2 au moins. Lorsqu'un noeud a été éliminé pendant le processus, il ne peut plus être un élément du chemin vérifiant les conditions 1 et 2 des noeuds quels qu'ils soient. Cette propriété peut être vue dans G_1 de la manière suivante :

On oriente les arêtes de G_1 qui font partie de V_1 et V_2 à partir de n vers les noeuds de type T_1 ou T_2 (cf Fig 8.8). Pour traiter les arêtes infinies de G_1, on suppose que chaque arête infinie se termine avec un noeud infini, ce noeud ne correspond pas à un tétraèdre de O_{12} qui est donc éliminé. L'arête infinie est donc orientée de la même manière que les arêtes finies.

Un noeud n de type T_{12} appartient à un chemin valide si et seulement s'il existe au moins deux arêtes de G_1 provenant de n, l'une faisant partie de V_1 et l'autre faisant partie de V_2. Lorsqu'une arête de G_1 reçoit une orientation contradictoire, elle ne peut pas être utilisée dans le "solid path" et doit donc être éliminée.

Il reste à décider quels tétraèdres de type T_1 et de T_2 doiventêtre gardés. Considérons un ensemble de tétraèdres ayant le même type T_i ($i = 1, 2$) maintenus après l'application de la règle 1 et intersectés par le plan P_j ($i \neq j$) au même point. Ces tétraèdres sont donc connectés

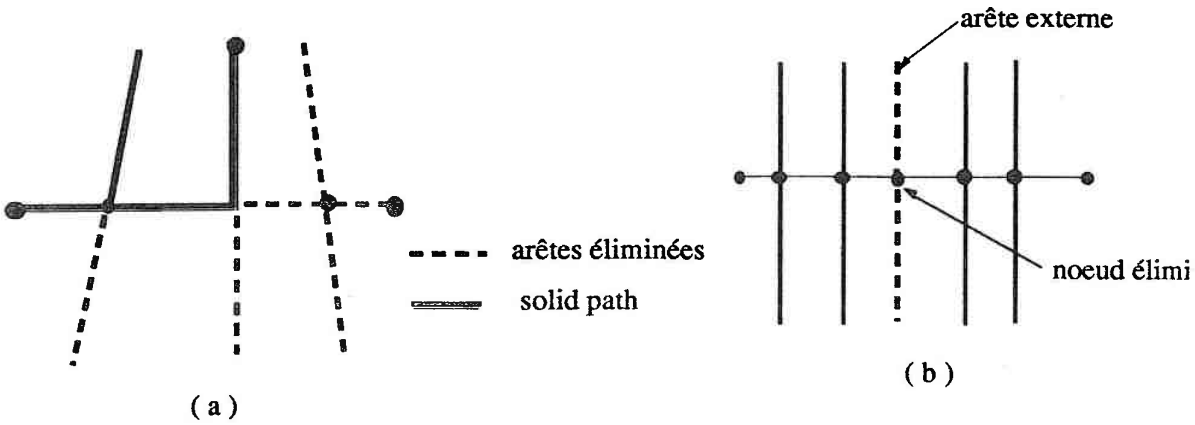


Figure 8.8

face-à-face. Si cet ensemble a un élément qui partage une face avec un tétraèdre gardé de type T_{12}, alors tous les tétraèdres de cet ensemble doivent être gardés (règle 3). Dans le graphe G_1, cette règle est facilement appliquée après une contraction du graphe. Si deux noeuds de même type T_i sont adjacents, on les considère comme un seul noeud de type T_i. Un noeud est éliminé si et seulement si il n'a pas d'arêtes adjacentes.

Construction de la forme totale

Pour la construction de la forme entière de l'objet, on applique la méthode ci-dessus à $L-1$ tranches. L'objet reconstruit va, dans plupart des cas, satisfaire les conditions 1 et 2 (cf page ??). Or la condition 1 peut être violée si une face d'un plan de coupe P_i se trouve isolée dans deux tranches adjacents $P_{i-1}P_i$ et P_iP_{i+1} (cf Fig 8.9.a). Dans ce cas, l'objet reconstruit ne va intersecter P_i que sur des sous-ensembles de la section de coupe donnée. La condition 1 peut aussi être violée si on trouve une arête d'un plan de coupe P_i ayant une singularité dans les deux tranches adjacentes contenant l'arête (voir Fig 8.9.b).

Ces difficultés arrivent lorsque la distance entre deux plans adjacents est trop grande. Pour éviter ce genre de circonstances et la création d'objets "non-solide", on doit effectuer des coupes intermédiaires. Si cela n'est pas réalisable, il est alors nécessaire d'interpoler des nouvelles sections.

La duplication de la section C_i du plan P_i produit une nouvelle section C_i', on place C_i' proche de P_i. La tranche entre C_i et C_i' peut être calculée par un moyen trivial et ceci va garantir qu'aucune face de la section de coupe n'est perdue et qu'aucune singularité ne peut se produire.

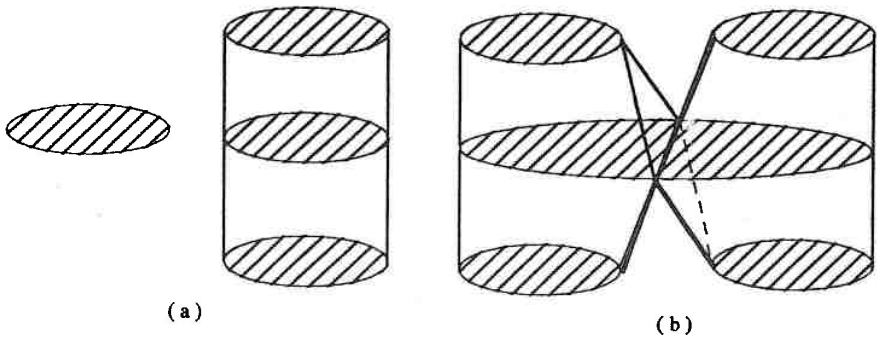


Figure 8.9

Conclusion

Nous avons expliqué en introduction les probèmes que posait l'application informatique aux sciences de la terre, pour lesquels le projet *GOCAD* se développe. Ce mémoire fait partie de ce projet.

Nous avons présenté ensuite des algorithmes pour résoudre certains de ces problèmes :

- Premièrement, nous construisons un ensemble d'outils de base pour la modélisation et la manipulation de T-surfaces complexes; ces outils peuvent également être utils pour d'autres applications de *GOCAD*.
- Deuxièmement, nous décrivons des algorithmes détaillés pour la modélisation de T-solides (T-surfaces fermées) et un algorithme pour le découpage de T-surfaces ainsi que des outils pour les opérations chirurgicales sur les T-surfaces effectuées de façon interactive.
- Troisièmement, nous présentons trois méthodes pour la modélisation de T-surface. La deuxième méthode qui convertit une grille 3D en une T-surface est une nouvelle approche, elle est très utile en pratique. Par exemple, beaucoup d'applications minières fournissent des données définies sur les grilles 3D (surtout les grilles de valeur 0 et 1, 0 signifie "à l'extétrieur" de la matière et 1 signifie "à l'intétrieur" de la matière), dans ces cas, cette méthode peut construire des T-surfaces automatiquement. En utilisant les outils développés dans *GOCAD*, toutes les modélisations de ces T-surfaces peuvent être faites de façon simple (ex: simulation de faille).

Tous ces algorithmes ont été implémentés et testés sur divers exemples, nous constatons que :

- La mise en oeuvre d'un algorithme géométrique n'est pas une chose facile à cause de la présence des cas particuliers, ceci étant souvents ignorés dans l'algorithme théorique. Un algorithme simple et efficace peut être très lourd à implémenter, or, bien souvent nous jugeons un algorithme en ne regardant que sa complexité théorique. Nous soulignons ici **qu'être correct est plus important qu'être rapide.**
- La technique de conception en programmation est très importante. Une bonne technique peut faciliter énormement le calcul et économiser considérablement la mémoire.
- L'erreur numérique en informatique est un phénomène important si l'on veut garantir la stabilité du calcul. Par exemple, dans le cas où les objets manipulés ont des tailles très

différentes, les mesures effectuées sur les petits objets peuvent devenir invisibles devant celles effectuées sur les grands.

Les surfaces traitées dans ce mémoire sont constituées par des facettes planes. Dans le futur, nous allons étendre les algorithmes aux surfaces composées de facettes curvilignes en connection avec le tracé de rayon (en cours de développement dans *GOCAD*) et avec l'application de *CFAO*.

Bibliographie

[1] D.Ayala and P.Brunet. ***Object representation by means of nonminimal division quartree and octree.*** In: ACM Transaction on Graphics. Vol 4. No 1. Jaunary, 1985.

[2] J.D. Boissonnat. ***Shape reconstruction from planar cross-sections.*** Rapport de recherche de l'INRIA, Janvier, 1986.

[3] I.Carlbom. ***An algorithm for geometric set operations using cellular subdivision techniques.*** In: IEEE Transaction on pattern Analysis. 1987.

[4] T.Carlier et N.Szafran. *STRUCTURE DE DONNEES POUR LA REPRESENTATION ET LA MANIPULATION DES POLYHEDRES.* Rapport de recherche en Infomatique et mathématique. Unité de recherche INPG-associée avec CNRS. Gronoble.

[5] B.Chazelle and J.Incerpi. ***Trangulation and Shape-complexity.*** In: ACM Transaction on Graphics. Vol 3. No 2. April, 1984.

[6] N.Chémanoff. ***Généralisation de forme pour la planification de l'exploitation souterraine des gisements complexes.*** Thèse, (1988), Ecole Nationale supérieure des mines de paris.

[7] A.Fourier and Y.Montuno. ***Trangulating simple polygons and equvalent problems.*** In: ACM Transaction on Graphics. Vol 3. No 2. April, 1984.

[8] D.Hearn — M.Pautine. ***Computer Graphics. 1986.***

[9] D.T.Lee and B.Schatcher. ***Two algorithms for constructiving Delaunay triangulation.*** In: Int. J.Compt. Informs. Sci. Vol 9. No 3. pp 219-242. June. 1980.

[10] D.T.Lee and F.prepara. ***Computational Geometry- A survey.*** In: IEEE Transaction on computers. Vol C-33. No 12. 1984.

[11] P.Lienhardt. ***Extension of the notion of map and subdivisions of a three-dimensional space.*** (1988), Université Louis Pasteur, Département d'Informatique, Strasbourg.

[12] P.Lienhardt. ***Subdivision de surfaces, Carte et s-v-t cartes.*** (1988), Université lois pasteur, Département d'informatique.

[13] J.L.MALLET. ***Discrete Smooth Interpolation.*** ACM Transactions on Graphics, vol.8, num.2, April 1989, p.121-144.

[14] J.L.MALLET. ***Three dimensionnnal graphics display of disconnected bodies.*** In: Geological mathematics.

[15] J.L.MALLET. ***Geometric Modelling and Geostatistics,*** In: Geostatistics, Proceedings of the Third International Geostatistics Congress, Avignon, vol.2, p.737-748, (1988), Kluwer Academic Publishers.

[16] J.L.MALLET. ***Discrete Smooth Interpolation applied to Geometric Modelling.*** GOCAD annual report, january 1989 - L.I.A.D. Ecole Nationale Supérieure de Géologie de Nancy.

[17] J.L.MALLET. ***Présentation d'un ensemble de méthodes de la cartographie automatique.*** Science de la terre. num 4. Octobre. 1974.

[18] M.E.MORTENSON. ***Geometric Modelling.*** (1985), John Wiley.

[19] M.Mantyla. ***Boolean Operations of 2-manifolds throught vertex neighbourhood classification.*** In: ACM Transaction on Graphics. Vol 5. No 1. Jaunary, 1986.

[20] P.Martin et D.Martin. ***Les algorithmes de calcul de l'intersection de solides définis par leur bord.*** Revue Internationale de CFAO et d'Infographie, vol.4, num.2. 88

[21] L.Quan. ***Contribution de la vision monoculaire à la perception tridimensionnelle.*** Thèse, (1989), INPL-CRIN.

[22] F.P.Preparata and M.I.Shamos. ***Computational geometry. 1985.***

[23] J.Veenstra and N.Ahuja. ***Line drawing of Octree-Represented Objets.*** In: ACM Transaction on Graphics. Vol 7. No 1. page 61-75. Jaunary, 1988.

[24] D-H.WU. ***Etude sur la représentation de surfaces complexes: Application à la reconstruction de surfaces échan- tillonnées.*** Thèse, (1988), Ecole Nationale Supérieure des Télécommunications.

[25] F.Yamaguchi and T.Tokieda. ***A unified algorithm for Boolean shape operations.*** In: IEEE Computer graphics and applications. June,. 1984.

[26] Y.CHIPOT. ***Smoothing a surface with DSI.*** Revue Internationale de CFAO et d'Infographie, vol.4, num.2.

[27] R.Bar-Yehudan and R.Grinward. ***Triangulating Polygons with Holes.*** Proceed of the 2nd Canadian Conference in Computatinnal Geometry.August, 1990

[28] H.EDELSBRUNNER. ***Algorithms in Combinatorial Geometry.*** EACTS. Monographs on Theoretical Computer Science. 1987

[29] GERALD E.FARIN. ***Geometryic Modeling: Algorithms and new trends.*** Society for Industrical and Apllied Mathematics. 1987

[30] P.BURGER and D.GRILLIES. ***Interactive Computer Graphics.*** Imperial College of Science, Technology and Medicine. 1989

[31] A.WATT. ***Three-Dimensional Computer Graphics.*** University of Sheffield. 1989

[32] M E.MORTENSON. ***Geometric Modeling.*** 1985.

[33] R.SEDGEWICK. ***Algorithms.*** in the Addison-Wesly series in Computer Science. 1988.

[34] W.E.LORENSEN and H.E.CLINE. ***Marching cubes: a high resolution 3D surface construction algorithm.*** in: Computer Graphics, Vol 21, Num 4, July 1987.

[35] B.Chazelle. ***Intersection of Convex objects in 2 or 3 dimensions.*** In: ACM Transaction on Graphics. Vol 34. No 1. Jaunary, 1987.

[36] J.ESCAMILIA, V.FAVIER et P.JEAN. ***Répresentation de connaissances dynamiques dans sherpa.*** in: Rapport de recheche, INRIA, Num 1208, Avril 1990.

[37] J.D BOISSONNAT. ***A dynamic construction of hirher order voronoi diagrams and its randomized analysis.*** in: Rapport de recheche, INRIA, Num 1207, Avril 1990.