Introduction à la modélisation et à l'algorithmique géométrique

O.STAB - olivier.stab@ensmp.fr

Octobre 2007

Table des matières

Table des figures

Introduction générale

Bien que ce document s'adresse à des élèves ingénieurs ou de 3^{ieme} cycle, un effort a été réalisé pour qu'il reste accessible à toute personnes ayant des bases en mathématiques et informatique. La connaissance de quelques notions en topologie, sur les algorithmes de tri et le concept de récursivité est souhaitable sans être indispensable.
Le thème abordé ici est le traitement de la géométrie sur ordinateur. Il ne s'agit pas d'un traité mais plutôt d'une introduction aux fondements de la modélisation et de l'algorithmique géométrique. Certains thèmes importants comme : la modélisation des courbes et des surfaces, l'interpolation polynomiale... ne sont pas abordés.
Le document se compose de 2 chapitres correspondant à des disciplines souvent abordées dans des ouvrages séparés. Le premier traite des modèles géométriques solides avec les représentations par la frontière, les approches constructives et l'énumération spatiale. Le second aborde l'algorithmique géométrique, c'est à dire les méthodes et les structures informatiques permettant de traiter efficacement des problèmes géométriques de calcul de proximité, d'enveloppe convexes, de triangulation... L'ensemble est agrémenté de références pour des approfondissements et d'exemples et d'exercices permettant d'améliorer la compréhension des concepts.

Chapitre 1

Modélisation géométrique

1.1 Introduction

Ce chapitre introduit la problématique et les fondements de la modélisation géométrique solide à travers les représentations les plus classiques. Il tente de répondre à quelques questions élémentaires :

- Quels objets cherche t-on a modéliser ?
- Comment sont-ils representés ?
- Quels sont les critères de comparaison entre les différentes représentations ?

"Un modèle géométrique solide est une représentation mathématique (non ambigue et complète) de la forme d'un objet physique afin de la traiter sur ordinateur" page 372 de [19].

1.1.1 Les propriétés des objets

Il faut avant tout définir les caractéristiques des objets que l'on cherche à représenter. Ici les solides rigides qui respectent donc :

Le déterminisme de frontière : la frontière des l'objets définissent sans ambiguité l'intérieur de l'extérieur. Cette propriété exclut par exemple la bouteille de Klein.

La description finie : pour pouvoir être décrite de façon exaustive.

La rigidité : la forme des objets est invariante, indépendante de la position et de l'orientation.

L'homogenéité : les objets doivent être homogènes. Ils doivent être exclusivement volumique (en 3D), surfacique (en 2D) ou linéiques (en 1D).

Des opérations rigides internes : Toutes combinaisons de rotation, de déplacement, d'opérations booléennes entre des solides doivent donner un solide valide.

1.1.2 Les principaux modèles

De nombreux modèles formels ont été décrits mais il s'apparentent tous à l'un des 3 approches ci-dessous :

La représentation frontière : (ou Brep[1]) : l'objet est décrit par sa frontière qui sépare l'intérieur de l'extérieur.

L'approche constructive : La représentation par construction solide souvent notée C.S.G[2] est une approche très générale où une géométrie est décrite par un processus de construction. Un exemple est l'arbre binaire de construction où les feuilles sont des primitives (boite, cylindre, cones...) et les noeuds des opérateurs (union, différence, intersection...)

La décomposition cellulaire : c'est un maillage conforme ou non de l'objet à représenter. Un cas particulier très usité dans certains domaines (imagerie médicale...) est l'énumération spatiale. L'espace est découpé en cellules orthomorphes aux axes. Elles sont localisées par rapport à l'objet : elles sont à l'intérieur ou à l'extérieur.

1.1.3 Les propriétés des modèles

Une représentation peut être comparée à une autre suivant plusieurs critères :

Le domaine : l'ensemble des objets qu'elle peut modéliser.

La complétude : sa capacité à répondre à des questions géométriques, par exemple : calcul de l'aire, la position d'un point...

L'unicité : important si l'on cherche à déterminer l'égaliter entre des objets.

La facilité de manipulation : pour créer et modifier le modèle.

Les performances techniques : la précision, la concision de la structure, et la rapidité des algorithmes...

1.1.4 Un exemple

Nous allons, pour chacun des modèles, présenter les concepts, la représentation mathématique et informatique ainsi que les algorithmes élémentaires d'interrogation :

La localisation d'un point : de coordonnées (x, y, z) à savoir si il se trouve DANS, HORS ou alors SUR la frontière de l'objet représenté.

L'intersection avec une droite : c'est à dire les coordonnées des points d'intersection avec la frontière de l'objet.

Prenons un exemple simple qui nous permettra au passage de faire quelques rappels de géométrie élémentaire. Supposons que l'on cherche à modéliser les cylindres. Les caractéristiques sont assez précises pour qu'un modèle valide soit réduit aux 2 réels définissant ses dimensions : le rayon R et la hauteur H du cylindre.

[1] Brep pour Boundary REPresentation
[2] C.S.G pour Constructive Solid Geometry

La localisation d'un point

La position (DANS, HORS ou SUR) d'un point de coordonnées (x, y, z) par rapport à un cylindre de rayon R et la hauteur H est résolu simplement en utilisant les représentations algébriques implicites des surfaces limites. Si l'on suppose que le modèle de cylindre est centré et son axe suivant z on obtient les équations suivantes :

$$\begin{array}{l} p_1 = x^2 + y^2 - R^2 \\ p_2 = z + H/2 \quad c \\ p_3 = z - H/2 \end{array} \tag{1.1}$$

p_1 donnent la distance du point (x, y, z) à la surface cylindrique, de même p_2, p_3 donnent la distance aux 2 plans. Les représentations des surfaces sont donc orientées avec la normale vers l'extérieur (ie. distance positive à l'extérieur). La localisation d'un point s'écrit donc simplement à l'aide de tests d'inégalités :

```
si (p1 < 0) et (p2 < 0) et (p3 < 0) alors position = DANS
sinon
  si ((p1 == 0) et (p2 <= 0) et (p3 <= 0)) ou
     ((p2 == 0) et (p1 <= 0)) ou
     ((p3 == 0) et (p1 <= 0))       alors position = SUR
                                    sinon position = HORS
```

L'intersection avec une droite

La représentation algébrique des surfaces est aussi utilisée pour calculer l'intersection d'une droite et d'un cylindre. Ici la droite sera représentée sous forme paramétrique. L'intersection d'un plan et d'une droite est donné par :

L'équation du plan :

$$Ax + By + Cz + D = 0 \tag{1.2}$$

La représentation paramétrique de la droite :

$$\begin{array}{l} x = a_1.t + a_2 \\ y = b_1.t + b_2 \\ z = c_1.t + c_2 \end{array} \tag{1.3}$$

La coordonnée t du point d'intersection quand il existe :

$$t = \frac{Aa_1 + Bb_1 + Cc_1}{Aa_2 + Bb_2 + Cc_2 + D} \tag{1.4}$$

Dans le cas d'un cylindre l'intersection avec les 2 plans donne :

$$\begin{array}{l} t_1 = \frac{c_1}{c_2 + H/2} \\ t_2 = \frac{c_1}{c_2 - H/2} \end{array} \tag{1.5}$$

Pour l'intersection avec la surface cylindrique on procède de la même façon et l'on obtient une équation du second degré en t. Il suffit ensuite de vérifier si les 4 solutions (2 seulement dans les 2 cas particuliers) appartiennent à la frontière du volume.

Conclusion

La manipulation du modèle offre, dans le cas d'un simple cylindre, assez peu d'intérêt. En revanche notez que le doublet (R, H) est une représentation valide des cylindres qui permet de répondre efficacement aux questions élémentaires d'appartenance et d'intersection.

1.2 La représentation frontière

La représentation frontière est souvent notée Brep (pour Boundary REPresentation). Comme son nom l'indique elle représente un objet par sa frontière c'est à dire par la limite entre l'intérieur et l'extérieur de l'objet. En 3D il s'agit donc d'un ensemble de faces elles même limitées par des arêtes (indispensable si l'objet n'est pas convexe) dont les extrémités sont des sommets. La géométrie des éléments de frontière peut être quelconque, en revanche il est nécessaire de pouvoir orienter les faces afin de distinguer, en chacun de leurs points, l'intérieur de l'extérieur. On aura deviné que la représentation frontière repose sur des propriétés topologiques importantes que nous allons rappeler dans une première partie. Dans la deuxième, nous présenterons la relation d'Euler qui lie les cardinaux des ensembles de faces, d'arêtes et de sommets. Enfin, après avoir présenté les structures informatiques (DECL) nous aborderons les algorithmes élémentaires d'interrogation et de manipulation.

1.2.1 Rappels de topologie

Ce paragraphe a pour objet de formaliser certaines notions comme l'homogénéïté, la connexité, le trou... Il n'est pas nécessaire à la première lecture mais néanmoins indispensable à une parfaite compréhension. Une présentation mathématique plus rigoureuse et surtout plus complète qui aborde les notions de complexe simplicial, de chemin et de bord se trouve dans [30].
L'homogénéïté est un concept qui mérite d'être précisé, pour ce rappelons la définition de l'adhérence.

Définition 1 L'adhérence : *un point x d'un espace topologique E est dit adhérent au sous ensemble A de E si l'intersection de A avec n'importe quel voisinage de x n'est pas vide.*

Définition 2 *Un ensemble E est* **régulier** *si il est égal à l'adhérence de son intérieur.*

Cette définition permet de formaliser la propriété d'homogénéïté : un objet homogène est régulier et réciproquement. La connexité simple est aussi une notion indispensable à la compréhension.

Définition 3 Connexité simple : *On dit qu'un espace topologique E est simplement connexe par rapport au point de base x si tout chemin fermé de E commençant et finissant en x peut être contracté en x.*

Prenons un exemple dans le plan : un polygone comme celui de la figure 1.3 n'est pas simplement connexe. Sa frontière est définie par des contours que l'on appele : **contour externe et contours internes**, (un seul contour interne dans l'exemple de la figure). Un tel polygone n'est pas simplement connexe car un chemin "entourant" un contour interne ne peut être contracté en un point sans sortir de l'espace (ici le polygone) ou sans être découpé.

Prenons maintenant un exemple dans l'espace : sur une surface torique il existe 2 familles de courbes fermées qui ne peuvent pas être contractées en un point. Elles indiquent l'existence d'un trou (voir la figure 1.2). Une surface torique n'est donc pas simplement connexe. En revanche on admettra que le plan lui-même est simplement connexe ce qui permet d'énoncer le théorème suivant :

Théorème 1 Le théorème des courbes de Jordan *stipule qu'une courbe simple fermée dans le plan le divise en 2 régions, un "intérieur" et un "exterieur".*

Il est clair que le théorème de Jordan peut être étendu aux surfaces simplement connexe mais ne s'applique pas aux autres (voir la figure 1.2). Dans le même ordre d'idée une autre définition peut s'avérer utile : c'est le genre d'une surface.

Définition 4 Le genre d'une surface *est le nombre maximum de découpage que l'on peut effectuer sans que la surface ne soit séparée en plusieurs morceaux (voir figure 1.2).*

Le terme de "séparation" mérite d'être précisé : elle fait appel au concept de la connexité par arc.

Définition 5 Connexité par arc : *On dit qu'un espace topologique E est connexe par arc si, pour toute paire p et q de points de E, il existe un chemin joignant p et q.*

Définition 6 Homéomorphisme : *Soient E et F deux espaces topologiques et soit f une application bijective de E sur F. Alors, si f et f^{-1} sont continues, ont dit que f est un homéomorphisme de E sur F, et on dit que E et F sont homéomorphes.*

Définition 7 Une variété de dimension r : *est un espace dont chaque point a un voisinage homéomorphe à une boule de dimension r.*

Dans un espace de dimension d, une variété de dimension $k(k < d)$ sera appelé une $k - face$.

De nombreuses représentations Brep se limitent au cas où la frontière est une variété de dimension 2. Dans la littérature anglo-saxone on parle de **frontière 2-manifold**.

1.2.2 Relation d'Euler

En 1752 Léonard Euler énonce une relation qui relie le nombre de sommets, d'arêtes et de faces d'un polyèdre convexe. À l'age de 20 ans Cauchy la démontre par récursion. C'est Henri Poincaré qui la généralise à tout n-polytope convexe :

Théorème 2 (La Relation d'Euler-Poincaré) *Les nombres* $n_k(P)(0 \leq k \leq d-1)$ *de k-faces d'un d-polytope P satisfont la relation :*

$$\sum_{k=0}^{d-1}((-1)^k n_k(P)) = 1 - (-1)^d \tag{1.6}$$

Démonstration 1 *On trouvera la démonstration dans [6] p 150.*

Soit P un polytope, le nombre de k-faces $n_k(P)$ est le nombre de variétés de dimension k. Notons $n_0(P) = s$ le nombre de sommets, $n_1(P) = a$ le nombre d'arêtes, $n_2(P) = f$ le nombre de faces et $n_3(P) = v$ le nombre de volumes. La relation d'Euler pour $d = 2, d = 3$ et $d = 4$ donne respectivement :

$$\begin{array}{c} d = 2 : s - a = 0 \\ d = 3 : s - a + f = 2 \\ d = 4 : s - a + f - v = 0 \end{array} \tag{1.7}$$

La figure 1.1 montre un exemple en 3D et 4D.

La relation d'Euler est également très utile pour calculer les cardinaux des éléments d'une partition. Pour une partition d'un espace de dimension XD il faut utiliser la relation de dimension $d = X+1$. Par exemple, pour une partition du plan $(X = 2)$ le nombre de polygones est donné par le nombre de faces de la relation pour $d = 3 : s - a + f = 2$. Pour une partition de l'espace $(X = 3)$ le nombre de polyèdres est donné par le nombre de volumes de la relation pour $d = 4 : s - a + f - v = 0$. La figure 1.1 l'illustre pour le cas de l'hypercube et du pentatope.

Dans le cas général, pour un polyèdre quelconque dont la frontière ou ses faces ne sont pas simplement connexes, la relation liant les cardinaux devient :

$$f - (a + a_n) + s = 2(cc - t) \tag{1.8}$$

ou a_n est le nombre d'anneaux (cad le nombre de contours interne dans les faces), cc le nombre de composantes connexes par arc et t le nombre de trous.

La surface d'un solide avec un trou n'est pas simplement connexe : il existe 2 chemins fermés indépendants non réductibles en un point (exemple du tore). La notion de trou et de genre d'une surface sont très proches (voir figure 1.2).

1.2.3 Représentation informatique

La représentation frontière a la structure d'un graphe.

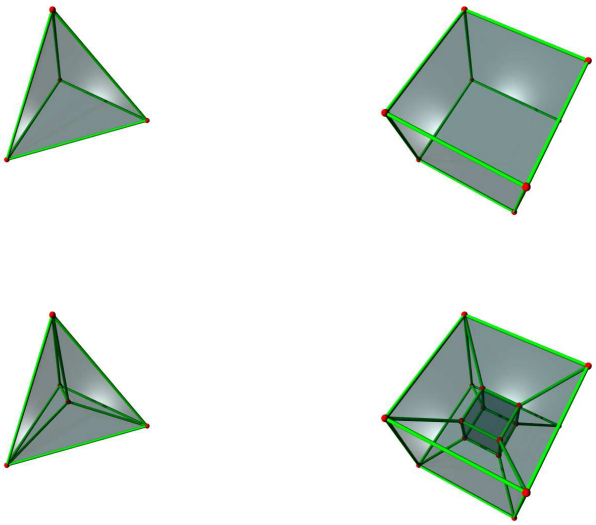


FIG. 1.1 – Relation d'Euler sur quelques polytopes
La relation d'Euler se vérifie aisément sur des polyèdres canoniques : $4-6+4=2$ pour le tétraèdre, $8-12+6=2$ pour le cube, mais aussi pour des partitions de l'espace. Attention dans ce cas il faut aussi compter le volume infini contenant le pentatope $(5-10+10-5=0)$ ou l'hypercube $(16-32+24-8=0)$.

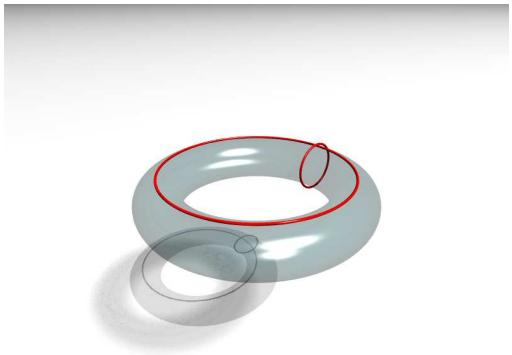


FIG. 1.2 – Genre des surfaces et trou
La surface d'un tore n'est pas simplement connexe : il existe 2 cycles indépendants non réductibles en un point. Le tore est une surface dont le genre est 2. Le tore présente 1 seul trou. Les surfaces comme le ruban de Moebius ou la bouteille de Klein ne représentent pas la frontière d'un volume mais le genre de leur surface est aussi de 2.

En 2D une frontière est constitué par 2 ensembles : un ensemble de sommets S et un ensemble d'arêtes A, ainsi que le graphe des arêtes :
G(A,S) = $a_i = (s_j, s_k)$, l'arête i joint le sommet s_j au sommet s_k

ou son graphe dual (le graphe d'adjacence des arêtes) :
G(A,S) = $s_i = (a_j, a_k)$, l'arête a_j arrive au sommet i, l'arête a_k en repart

Par convention les arêtes sont orientées pour que la matière soit toujours à gauche ou à droite. La description d'un domaine comportant des cavités se fait à l'aide de plusieurs cycles d'arêtes : un cycle correspondant au **contour extérieur**, des cycles correspondant aux **contours intérieurs**.

Le modèle doit respecter des **contraintes d'intégrité** :

- les arêtes ne peuvent s'intersecter qu'en leurs sommets communs,
- un sommet appartient à 2 arêtes,
- il existe une orientation des arêtes telle que tout sommet est origine d'une arête et extremité d'une autre.

En 3D une frontière peut être représentée par 3 ensembles : un ensemble de sommets S, un ensemble d'arêtes A et un ensemble de faces F.

Pour chaque face on donne le graphe des arêtes :
G(A,S) = $a_i = (s_j, s_k)$, l'arête i joint le sommet s_j au sommet s_k

ou son graphe dual (le graphe d'adjacence des faces) :
G(A,F) = $a_i = (f_j, f_k)$, la face f_j est à "droite" de l'arête i, la face f_k est à "gauche" de l'arête i

ou un graphe mixte qui réunit les 2 précédents et contient donc différents type de noeuds et différents type d'arcs. Une représentation informatique est donnée figure 1.4.

Par convention la normale aux faces désigne l'extérieur du volume.

Comme en 2D, le modèle doit respecter des **contraintes d'intégrité** :

- les faces ne peuvent s'intersecter qu'en leurs sommets ou arêtes communs,
- une arête appartient à 2 faces (ou plus précisément 2 contours),
- il existe une orientation des faces telle que toutes les arêtes sont parcourues dans un sens et dans l'autre,
- tous les sommets adjacents à un sommet donné forment dans l'espace un polygone simple.

A chaque arête peut être associée une géométrie ou au contraire les courbes sont toutes représentées par des segments de droite (discrétisation). De même en 3D, on peut associer des géométries aux faces ou travailler sur des facettes planes (généralement des triangles).

```
typedef unsigned long        t_compteur;
typedef unsigned long        t_sommet;
typedef double               t_coord;

t_coord COORD[nbp][2]; /* coordonnees des points dans le plan */

/* Pour un polygone simple de nbc cotes */
/* Avec simple chainage : */
t_sommet SUIV[nbc];   /* sommet suivant dans le polygone      */

/* Avec double chainage : */
t_sommet SUIV[nbc];  /* sommet suivant dans le polygone       */
t_sommet PREC[nbc];  /* sommet precedent dans le polygone     */


/* Un domaine avec nbt trous : */
t_sommet DEBUT[nbt+1]; /* premier sommet d'un contour          */
t_sommet SUIV[nbp];  /* sommet suivant dans un polygone       */
```

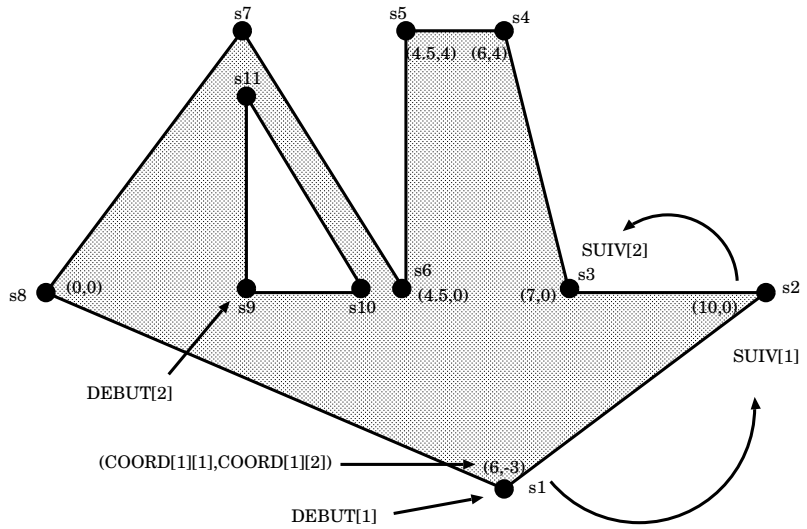


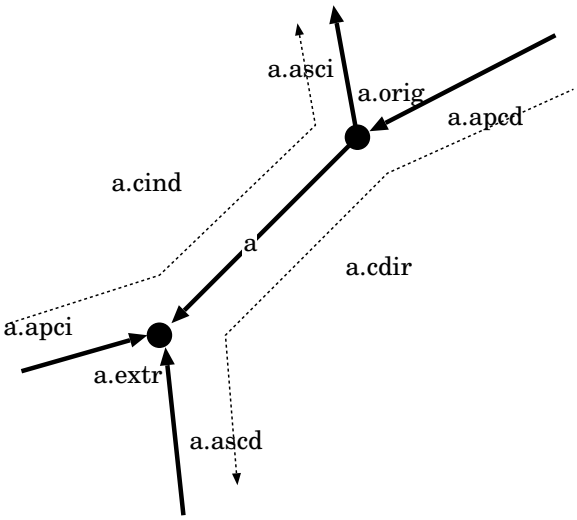
FIG. 1.3 – Structure de données pour un domaine polygonal

DEBUT[i] : donne le premier noeud du ième contour. Par convention le premier contour est le contour extérieur les suivant les contours intérieurs ou "trou".

SUIV[j] : donne le noeud qui suis le noeud j en parcourant le contour dans le sens trigonométrique si le contour est externe, dans le sens inverse si le contour est interne.

COORD[j [1]] : première coordonnée du noeud j.

```
typedef     double      decl_coord_t;
typedef     long        decl_nature_t;
/* valeurs pour le type  decl_nature_t  */
# define DECL_FACE      0
# define DECL_TROU      1
typedef     struct POLYEDRE {
    struct CONTOUR      *face;   /* premiere face du polyedre     */
    }                       decl_polyedre_t, *decl_polyedre_p;
typedef     struct CONTOUR {
    decl_nature_t        natu;   /* DECL_FACE ou DECL_TROU        */
    struct POLYEDRE     *poly;   /* polyedre contenant la face    */
    struct SOMMET       *somm;   /* si le contour est sans arete  */
    struct ARETE        *aret;   /* premiere arete du contour     */
    struct CONTOUR      *trou;   /* contour interne (trou) suivant*/
    }                       decl_contour_t, *decl_contour_p;
typedef     struct ARETE {
    struct SOMMET       *orig, *extr;
    struct ARETE        *ascd, /* arete suivante contour direct   */
                        *asci, /* arete suivante contour indirect */
                        *apcd, /* arete precedente contour direct */
                        *apci; /* arete "     "    "  "  indirect */
    decl_contour_p       cdir,cind; /* contour direct,indirect    */
    }                       decl_arete_t, *decl_arete_p;
typedef     struct SOMMET
    decl_arete_p        *aret; /* une arete au sommet             */
    decl_coord_t          x,y,z;
    }                       decl_sommet_t, *decl_sommet_p;
```



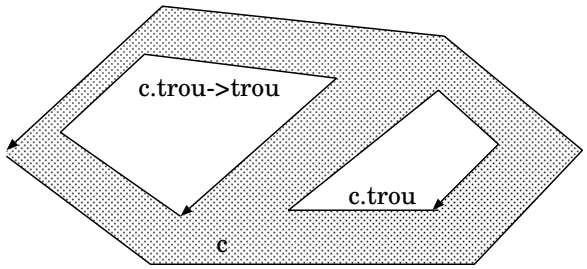


FIG. 1.4 – Structure de données d'une DECL

DECL signifie "Double Edged Connected List". La structure est constituée des objets : SOMMET, ARETE, CONTOUR et POLYEDRE (qui correpond à une composante connexe). Chaque arête appartient à 2 listes : le contour direct et le contour indirect. Une face est constitué d'un contour extérieur (de nature DECL_FACE) et de sa liste de contours internes (de nature DECL_TROU et chainée sur le champ *trou).

1.2.4 Les algorithmes élémentaires

La localisation d'un point :

Le problème est de localiser un point x par rapport à un domaine Ω : à savoir si x est à l'intérieur de Ω, ou pas. Pour une représentation frontière le principe de l'algorithme est simple. Il suffit de prendre une demi-droite partant du point x (par exemple une parallèle aux abscisses) et de compter le nombre d'intersections i avec les faces de la frontière. Si i est impair alors x est dans Ω sinon il est hors de Ω. L'algorithme 2D est donné figure 1.5. Comme pour beaucoup d'algorithmes géométriques, sa mise au point n'est pas triviale à cause des cas particuliers (ici la tangence).

L'algorithme 3D repose sur le même principe qu'en 2D : la parité du nombre d'intersections avec les faces. Il est décrit dans [15] et utilise la procédure de calcul d'intersection d'une demi-droite avec le polyèdre, décrite dans le paragraphe suivant.

intersection avec une droite :

De nombreux algorithmes nécessitent le calcul de l'intersection d'une droite avec le modèle, entre autre les algorithmes de calcul des parties cachées.

Si l'on ne cherche pas à connaître la position du point d'intersection et si la droite est parallèle à un des axes alors, l'algorithme le plus simple est le suivant :

```
Pour toutes les faces F du polyedre
  Projeter F sur le plan perpendiculaire a la droite : F'
  Dans ce plan la droite est un point : P'
  Si F' contient le point P' alors INTERSECTION
```

L'algorithme est identique dans le cas d'une demi-droite ou d'un segment si l'on prend la précaution de supprimer au préalable les faces dont le plan se situe avant ou après le segment (ou la demi-droite). Dans le cas du segment il suffit de tester si les deux extrémités sont du même coté de la face, c'est à dire si l'equation algébrique implicite du plan appliquée aux coordonnées des deux points, donne le même signe.

Dans le cas où l'on souhaite la liste des coordonnées des points d'intersections, l'algorithme est légèrement différent :

```
LPI = {} /* liste vide*/
Pour toutes les faces F du polyedre
  Calculer P le point d'intersection avec le plan de F
  Projeter P et F dans un des 3 plans de base : P', F'
  Si F' contient P' alors LPI = {P'}+LPI
```

Exercice 1 *Donnez l'expression des coordonnées du point d'intersection d'un plan avec une droite.*

Exercice 2 *Décrivez les étapes de l'algorithme qui détermine si un point est intérieur ou extérieur à un polyèdre, en utilisant l'intersection avec une demi-droite.*

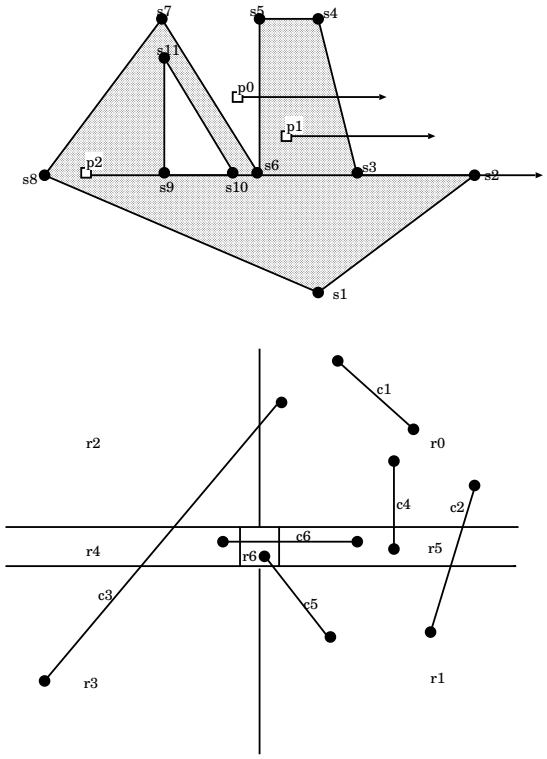


FIG. 1.5 – Point dans polygone

L'algorithme classe chaque segment du polygone par rapport à la demi-droite. Il y a six classes : pas d'intersection (c1), intersection simple (c2), singularité...Le plan est découpé en sept régions (r0, r1... r6). Il suffit de localiser les extremités des segments pour identifier les cas triviaux (voir tableau). Par exemple si les deux extrémités sont dans la région r0 il ne peut y avoir d'intersection (c1). De même si les 2 points sont dans la région r1, r2 ou r3... Les singularités (c3,c4) apparaissent quand le couple (ri,rj) n'est plus suffisant pour décider de l'intersection ou de la non-intersection. Par exemple les singularités simples (c3) apparaissent quand on a les couples (r0,r3),(r2, r1) ou leurs symétriques. Elles peuvent être traitées à l'aide de tests géométriques locaux alors que l'on est obligé de parcourir le polygone pour traiter les singularités compliquées (c4).

	r0	r1	r2	r3	r4	r5	r6
r0	c1	c2	c1	c3	c1	c4	c5
r1	c2	c1	c3	c1	c1	c4	c5
r2	c1	c3	c1	c1	c1	c4	c5
r3	c3	c1	c1	c1	c1	c4	c5
r4	c1	c1	c1	c1	c1	c6	c5
r5	c4	c4	c4	c4	c6	c4	c5
r6	c5	c5	c5	c5	c5	c5	c5

c1 :	Pas d'intersection
c2 :	Intersection simple
c3 :	Singularité simple
c4 :	Singularité compliquée
c5 :	Sur une extremité du segment
c6 :	Sur le segment (à l'interieur)

Les algorithmes ci-dessus sont donnés pour le cas 3D, mais ils fonctionnent aussi pour le cas 2D (dans le plan).

Le calcul de l'aire ou du volume :

Dans les cas simples l'aire d'un polygone de n arêtes est donnée par la formule (le polygone n'est pas forcément simplement connexe mais les arêtes doivent être orientées) :

$$A = \sum_{i=0}^{n} \int_{x_i}^{x_{i+1}} Y_i(x)dx \tag{1.9}$$

Si les arêtes sont des segments de droite on obtient :

$$A = \sum_{i=0}^{n} \frac{1}{2}(y_{i+1} + yi)(x_{i+1} - x_i) \tag{1.10}$$

Dans le cas général il faut utiliser la représentation paramétrique de la géométrie frontière et faire un changement de variable pour l'intégration :

$$\int_{x_i}^{x_{i+1}} Y_i(x)dx = \int_0^1 Y_i(t(u))t'(u)du \tag{1.11}$$

En 3D le principe est le même, mais il est utile , au préalable, de trianguler les faces de la frontière pour faciliter l'expression de l'intervale d'intégration. En revanche la triangulation peut être quelconque et même présenter des triangles intersectants.

$$\int\int Z_i(x,y)dxdy = \int_{u=0}^{1} \int_{v=0}^{1-u} Z_i(x(u,v), y(u,v))Det(J)dudv \tag{1.12}$$

où $Det(J)$ est le déterminant de la Jacobienne :

$$J = \begin{pmatrix} \frac{dx}{du} & \frac{dy}{du} \\ \frac{dx}{dv} & \frac{dy}{dv} \end{pmatrix} \tag{1.13}$$

Si les triangles sont linéaires on peut montrer que :

$$V = \sum_{i=0}^{n} \frac{z_{i0} + z_{i1} + z_{i2}}{2}(x_{i1} - x_{i0})(y_{i2} - y_{i0}) - (x_{i2} - x_{i0})(y_{i1} - y_{i0}) \tag{1.14}$$

1.2.5 La Manipulation de la Brep

On considère 2 niveaux de manipulation :
- Le niveau élémentaire qui est généralement transparent pour l'utilisateur mais pas pour le programmeur (ce sont les méthodes).
- Le niveau de l'interface utilisateur.

Ces 2 niveaux sont abordés dans les 2 sections qui suivent.

Les opérateurs qu'ils soient d'un niveau ou de l'autre doivent répondre à 2 exigences :

- Ils doivent garantir la validité de la structure (c.a.d. interdire la construction d'une structure non-valide).
- Ils doivent être efficaces et facile d'utilisation.

Les opérateurs d'Euler

Les opérateurs d'Euler sont des opérateurs élémentaires qui respectent la relation du même nom. Ils permettent de construire n'importe quel polytope. Ils sont principalement utilisés en 3D pour les polyèdres (en 2D la structure est si simple qu'elle ne nécessite pas ces opérateurs). Nous nous limiterons donc dans cette section au cas des polyèdres.

Il existe une infinité d'opérateurs en 3D mais ils peuvent être réduits au nombre de 5. D'autres sont néanmoins utilisés pour des raisons d'ordre pratique ou algorithmique.

MVSF	Make Vertex Shell and Face $(s+1)-(a+a_n)+(f+1)=2((cc+1)-t)$
MEV	Make Edge and Vertex $(s+1)-(a+1+a_n)+f=2(cc-t)$
MEF	Make Edge and Face $s-(a+1+a_n)+(f+1)=2(cc-t)$
MEKR	Make Edge Kill Ring $s-(a+1+a_n-1)+f=2(cc-t)$
KFMRH	Kill Face Make Ring and Hole $s-(a+a_n+1)+(f-1)=2(cc-(t+1))$

Un cube peut être construit et de différentes façon à l'aide de ces opérateurs. Un exemple est donné figure 1.6. Si l'on souhaite maintenant modifier ce cube en le perçant il faut construire des anneaux (avec KEMR) puis utiliser KFMRH.

Théorème 3 Le théorème d'inversion *établit que tout polyèdre se déduit de l'objet nul par application d'une suite finie d'opérateurs d'Euler de base. Inversement, en partant d'un polyèdre quelconque, on peut aboutir à l'objet nul par application répétée des inverses des opérateurs de base.*

Le théorème d'inversion présente un intérêt théorique mais aussi pratique puisqu'il permet entre autre de stocker les polyèdres sous une forme standard même si elle n'est pas unique.

Plusieurs algorithmes d'inversion ont été proposés. Le plus connu est probablement l'algorithme de Mantyla [18]. L'idée est de parcourir toutes les arêtes dans un ordre quelconque et de les réduire par l'opérateur approprié (KEF,KEV, KEMR ou MFKRH et KEF). On aboutit alors à un polyèdre sans arête et contenant donc une unique face par composante connexe. Chaque face est formée de plusieurs contours tous réduits à un sommet. On joint dans chaque face, le sommet de chaque anneau à celui du contour extérieur par un MEKR puis on

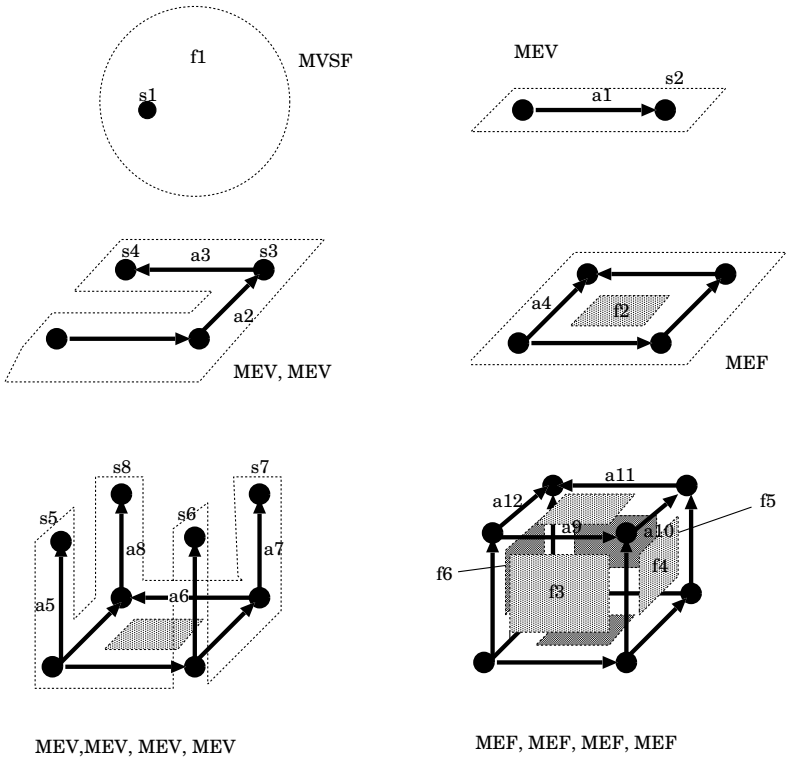


FIG. 1.6 – Construction d'un cube avec les opérateurs d'Euler

	faces	aretes	anneau	sommets	comp. conn.	trou
	0	0	0	0	0	0
MSVF	1	0	0	1	1	0
MEV	1	1	0	2	1	0
MEV	1	2	0	3	1	0
MEV	1	3	0	4	1	0
MEF	2	4	0	4	1	0
MEV	2	5	0	5	1	0
MEV	2	6	0	6	1	0
MEV	2	7	0	7	1	0
MEV	2	8	0	8	1	0
MEF	3	9	0	8	1	0
MEF	4	10	0	8	1	0
MEF	5	11	0	8	1	0
MEF	6	12	0	8	1	0

détruit le premier sommet et l'arête ainsi créée par un KEV. Chaque composante connexe n'est plus formée alors que d'une face, d'un contour et d'un sommet.

Les opérations booléennes

Le problème est le suivant : étant donné 2 domaines A et B (constitués d'une ou plusieurs composantes connexes) et représentés par R_A et R_B comment obtenir une représentation R_C du domaine résultant de l'union $C = A \cup B$, de l'intersection $C = A \cap B$ ou la différence $C = A - B$.

Définition 8 *On appele* **opérations booléennes régularisées** *les opérations booléennes classiques* $\cup, \cap, -$ *modifiées pour ne produire que des objets réguliers. Elles sont généralement notées :* $\cup^*, \cap^*, -^*$.

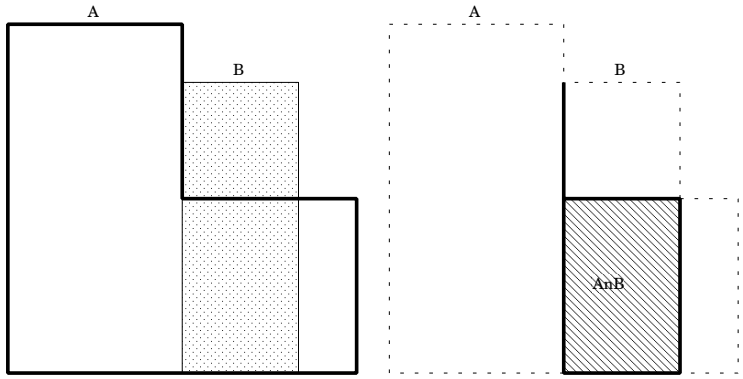


FIG. 1.7 – Opérations booléennes non régularisées

Le principe des opérations booléennes sur une représentation frontière est le suivant. La frontière du domaine résultant est formée des parties des frontières des opérandes. Les parties de la frontière de A et celle de B sont à garder ou à rejeter suivant leurs positions relatives et le type d'opération. Il y a 8 classes suivant leurs positions $(dans, hors, sur)$:

$$\begin{array}{llll} Fr(A)dansB & Fr(A)horsB & Fr(A)sur+B & Fr(A)sur-B \\ Fr(B)dansA & Fr(B)horsA & Fr(B)sur+A & Fr(B)sur-A \end{array}$$

où Fr désigne la frontière de A et le signe associé "sur" est le signe du produit scalaire entre les normales des faces de la frontière de A et les normales de la frontière de B.

$$\begin{array}{llll} Fr(A \cup^* B) & = (Fr(A)horsB) & \cup(Fr(A)sur+B) & \cup(Fr(B)horsA) \\ Fr(A \cap^* B) & = (Fr(A)dansB) & \cup(Fr(A)sur+B) & \cup(Fr(B)dansA) \\ Fr(A -^* B) & = (Fr(A)horsB) & \cup(Fr(A)sur-B) & \cup(Fr(B)dansA) \end{array}$$

Définition 9 *On appele* **contour d'intersection** *noté* C_i *du triplet* $A, B, \diamondsuit$ *la limite sur la frontière de* A *(ou celle de* B*) entre les parties à garder et celles*

à rejeter. C_i est constitué de plusieurs cycles d'arêtes résultant de l'intersection de la frontière de A avec celle de B.

Les grandes étapes de l'algorithme des d'opérations booléennes sont les suivantes :

- Calcul de A_i : l'ensemble des arêtes données par l'intersection entre toutes les faces du polyèdre P_A et les faces du polyèdre P_B.
- Calcul de C_i : choix des cycles d'arêtes dans A_i. Insertion de C_i dans P_A et P_B.
- Suppression des faces, des arêtes et des sommets de P_A et P_B qui n'appartiennent pas à la frontière du domaine résultant.
- Reconstruction du polyèdre par collage des frontières sur le C_i.
- Union des faces coplanaires et des arêtes colinéaires.

On trouvera dans [8] une description précise des étapes de cet algorithme.

1.2.6 Les propriétés du modèle Brep

Les propriétés du modèle Brep sont les suivantes :

Le domaine : L'ensemble des objets représentés peut être très large, cela dépend principalement de la géométrie des arêtes et de la géométrie des faces pour le 3D.

La complétude : Les méthodes pour le calcul de l'aire (le volume) et la localisation d'un point sont simples et efficaces.

L'unicité : En 2D il existe une forme standard unique et indépendante de la position et de l'orientation pour des polygones simples (shamos). Pour des géométries plus complexes et pour le 3D, l'unicité n'est généralement pas vérifiée.

La facilité de manipulation : Elle dépend des objets à modéliser, généralement correcte en 2D, la manipulation devient difficile en 3D et nécessite des outils vérifiant la validité de la structure.

Les performances techniques : La précision et la concision dépendent de la géométrie des arêtes. La plupart des algorithmes sont rapides et efficaces en 2D.

1.2.7 Les conclusions

La représentation frontière est probablement la représentation qui est la plus utilisée, celle sur laquelle on a le plus travaillé. Néanmoins, de nouveaux problèmes surgissent quand on cherche à traiter le cas des polyèdres à faces quelconques. On distingue :

- Les problèmes de type géométriques liées à la géométrie des faces. De nombreux algorithmes deviennent alors très difficiles à mettre au point ou peu performants (on pourra consulter à ce propos [10]).

– Les problèmes de type topologiques liées à l'adjacences entre les faces et donc la manipulation de domaines non-manifold. On pourra trouver des extensions du modèle présenté précédemment dans [8] page 61, dans [28] p 190 et dans [32]. Une autre approche très prometteuse est celle des **cartes généralisées ou G-map** dont on trouvera une description dans [17].

1.3 L'arbre de construction solide ou C.S.G

L'approche constructive est probablement la plus "riche" dans la mesure où toute méthode pouvant produire un historique s'y apparente. Pour des raisons pédagogiques nous allons ici nous limiter aux arbres booléens de construction ou C.S.G. pour Constructive Solid Geometry.

1.3.1 La représentation informatique

Le C.S.G est un arbre binaire dont les feuilles sont des "primitives géométriques" et les noeuds les opérateurs booléens régularisées ($\cup^*, \cap^*, -^*$). Les primitives peuvent être très diverses. En 3D on peut considerer : les volumes élémentaires (boites, cylindres, cones...), les volumes obtenus par d'autres opérations (extrusion ou balayage...), les volumes décrits par leur représentation frontière... Les transformations géométriques (translation, rotation, homothétie) peuvent être stockées dans l'arbre de construction : au niveau des feuilles elles ne s'appliquent qu'à la primitive, au niveau d'un noeud elles s'appliquent aux sous-arbre droit et gauche. Un exemple de C.S.G. est donné figure 1.8.

Remarquons que la structure de l'arbre est identique en 2D et en 3D. Un exemple de déclaration en langage C est donné figure 1.9.

1.3.2 Les algorithmes élémentaires

La plupart des algorithmes élémentaires ont la structure d'un algorithme de parcours de l'arbre. Arrivé aux feuilles le traitement spécifique est appliqué, puis quand 2 sous-arbres sont traités leurs résultats sont "combinés". Un schéma directeur est donné figure 1.10.

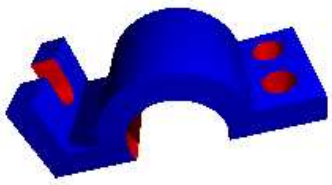
Exercice 3 *Modifiez la structure de données (figure 1.9) et l'algorithme de parcours (figure 1.10) pour traiter les transformations au niveau des noeuds.*

La localisation d'un point :

L'algorithme repose sur le principe de la distributivité des opérations. Par exemple, si l'on note p un point de l'espace :

$$p \in (A \cup B) \Longleftrightarrow (p \in A) ou (p \in B) \tag{1.15}$$

L'algorithme teste d'abord si le point appartient aux primitives (les feuilles) puis, en chaque noeud, combine les résultats des 2 sous arbres suivant le type de



```
...
difference {
   union {                  /* --- la partie bleue --- */
         union {
            box { <0,0,0> <107,48,11> }
            box { <0,0,0> < 11,48,25> translate <96,0,11> }
            }
         cylinder { <0,0,0> <0,48,0> 30 translate <53.5,0,5.5> }

         texture { pigment { red 0.0 green 0.0 blue 1.0 } }
         }
   union {                 /* --- la partie rouge --- */
         union {
             cylinder { <0,-1,0> < 0,49,0> 19 translate <53.5,0,5.5> }
             box      { <0,-1,0> <60,49,-30>  translate <23.5,0,0> }
             }
         union {
             union {           /* les 2 trous verticaux */
                 cylinder { <0,0,-1> <0,0,12> 6.5 translate <0,13,0> }
                 cylinder { <0,0,-1> <0,0,12> 6.5 translate <0,35,0> }
                 translate <13,0,0>
                 }
             union {           /* le trou au fond */
                 cylinder { <0,0,0> <13, 0, 0>  6.5 translate <0,24.0,0> }
                 box      { <0,0,0> <13,13,13>      translate <0,17.5,0> }
                 translate <95,0,24>
                 }
             }
         texture { pigment { red 1.0 green 0.0 blue 0.0 }   }
         }
}
```

FIG. 1.8 – Définition de la géométrie avec un arbre CSG
Exemple de pièce mécanique décrite dans la syntaxe de POV-RAY : fichier de description et image générée.

```
typedef ARBRE{
              p_feuille        f;
              p_noeud         op;
              struct ARBRE   *fd;
              struct ARBRE   *fg;
            } t_arbre, *p_arbre;

/* valeurs pour le type des feuilles */
#define CSG_BOITE     0
#define CSG_CYLINDRE  1
#define CSG_TORE      2 /* .... */
typedef FEUILLE{
              t_type      type;     /* type de la primitive */
              t_scalaire  data[3];  /* parametres `` `` ``  */
              t_point     origine;  /* position  `` `` ``   */
              t_vecteur   repere[3];/* orientation  `` ``   */
            } t_feuille, *p_feuille;

/* valeurs pour le type des noeuds */
#define CSG_INTER     0
#define CSG_DIFFE     1
#define CSG_UNION     2
typedef NOEUD{
              t_type      type; /* type de l'operateur */
            } t_noeud, *p_noeud;

/* macro d'acces a la structure */
#define FILS_DROIT(arbre)     ((arbre)->fd)
#define FILS_GAUCHE(arbre)    ((arbre)->fg)
#define FEUILLE_ARBRE(arbre)  ((arbre)->f)
#define NOEUD_ARBRE(arbre)    ((arbre)->op)
#define TYPE_FEUILLE(feuille) ((feuille)->type)
#define TYPE_NOEUD(noeud)     ((noeud)->type)
```

FIG. 1.9 – Déclaration des structures de données pour les arbres CSG

Déclaration de la structure des "objets" et des méthodes d'accès aux données sous forme de macro. Cette structure est minimum, elle doit être modifié pour traiter :

- le 2D et le 3D indifférement,
- différents types de positionnement (relatif, absolu),
- différents types de primitives (volumes canoniques ou obtenus par opération d'extrusion...),
- ...

```
TRAITER_ARBRE(A,A_RES)
/*********************************************/
/* A : arbre binaire                         */
/* A_RES : resultat du traitement de l'arbre */
/*********************************************/
{ if( A == NIL )return(ERREUR);
  AD = FILS_DROIT(A);
  AG = FILS_GAUCHE(A);
  if((AD == NIL)&&(AG == NIL)){
      /* cas d'une feuille */
      A_F = FEUILLE_ARBRE(A);
      return(TRAITER_FEUILLE(A_F,A_RES));
  }
  if( TRAITER_ARBRE(AG,&AG_RES) != OK)
    return(ERREUR);
  if( TRAITER_ARBRE(AD,&AD_RES) != OK)
    return(ERREUR);
  A_OP = NOEUD_ARBRE(A);
  return(TRAITER_NOEUD(A_OP,AG_RES,AD_RES,A_RES));
}
TRAITER_FEUILLE(F,A_RES)
/*********************************************/
{ switch(TYPE_FEUILLE(F)){
  case CSG_BOITE   : return(TRAITER_BOITE(F,A_RES));
  case CSG_CYLINDRE: return(TRAITER_CYLINDRE(F,A_RES));
  case CSG_TORE    : return(TRAITER_TORE(F,A_RES));
...}
TRAITER_NOEUD(OP,AG_RES,AD_RES,A_RES)
/*********************************************/
{ switch(TYPE_NOEUD(OP)){
  case CSG_UNION        :
    return(TRAITER_UNION(AG_RES,AD_RES,A_RES);
  case CSG_INTER :
    return(TRAITER_INTERSECTION(AG_RES,AD_RES,A_RES);
  case CSG_DIFFE   :
    return(TRAITER_DIFFERENCE(AG_RES,AD_RES,A_RES);
...}
```

FIG. 1.10 – Algorithme de parcours d'un arbre (CSG)

TRAITER_ARBRE est un algorithme classique de parcours d'arbre binaire. Les informations propres au CSG apparaissent au niveau des feuilles et des noeuds. Ce schéma sert pour de nombreux traitements : la localisation d'un point dans le CSG, le calcul de la Brep associée...

l'opérateur (les tables de composition sont données sur la figure 1.11). L'algorithme s'écrit simplement sur le schéma de la figure 1.10 où il suffit de remplacer TRAITER_ par POINT_DANS_. Cependant, comme le montre la figure 1.11, il y a ambiguïté quand le point (noté p) se trouve classé sur les deux frontières (de A et de B). Il est important de noter que **l'ambiguïté apparaît parce que l'on utilise des opérations booléennes régularisées**. Résoudre le problème n'est pas trivial : utiliser les normales aux surfaces n'est pas suffisant dans le cas général.

Dans la pratique c'est le point testé qui est ramené dans le repère local de la primitive et ce afin de faciliter les calculs. Les transformations stockées dans l'arbre CSG sont inversées puis appliquées au point. Par exemple, si l'on note T une transformation et p un points de l'espace :

$$p \in (A \cup T(B)) \Longleftrightarrow (p \in A) ou (T^{-1}(p) \in B) \tag{1.16}$$

Exercice 4 *Ecrire l'algorithme qui permet de localiser un point par rapport à un cylindre de rayon r et de hauteur h.*

Intersection avec une droite

Comme précédemment, l'algorithme repose sur le principe de la distributivité des opérations. Par exemple, si l'on note D une droite :

$$D \cap (A \cup B) \Longleftrightarrow (D \cap A) \cup (D \cap B) \tag{1.17}$$

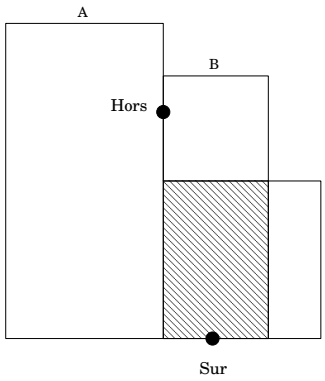
Le problème est maintenant trivial puisque 1D. L'algorithme suit le même schéma que le précédent : il calcule d'abord l'intersection de la droite avec les primitives (les feuilles) puis, en chaque noeud, combine les résultats des 2 sous arbres suivant le type de l'opérateur. En chaque noeud une opération booléenne régularisée est effectuée entre des segments de droite. Remarquons que les opérateurs régularisées introduisent là encore une ambiguïté qu'il faut résoudre par des tests géométriques.

Comme précédemment c'est la droite qui est ramené dans le repère local de la primitive. Les transformations stockées dans l'arbre CSG seront inversées puis appliquées à la droite. Par exemple, si l'on note T une transformation et D la droite de l'espace :

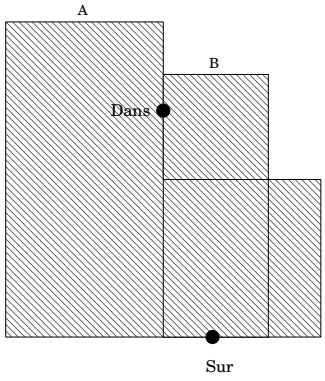
$$D \cap (A \cup T(B)) \Longleftrightarrow (D \cap A) \cup T(T^{-1}(D) \cap B) \tag{1.18}$$

Exercice 5 *Ecrire l'algorithme qui calcule l'intersection d'une droite avec un cylindre de rayon r et de hauteur h.*

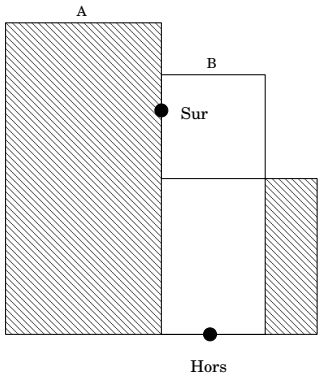
Exercice 6 *Ecrire l'algorithme qui réalise une opération booléenne régularisée entre deux segments de droite.*



$A \cap B$	Dans	Sur	Hors
Dans	Dans	Sur	Hors
Sur	Sur	Sur/Hors	Hors
Hors	Hors	Hors	Hors



$A \cup B$	Dans	Sur	Hors
Dans	Dans	Dans	Dans
Sur	Dans	Sur/Dans	Sur
Hors	Dans	Sur	Hors



$A - B$	Dans	Sur	Hors
Dans	Hors	Sur	Dans
Sur	Hors	Sur/Hors	Sur
Hors	Hors	Hors	Hors

FIG. 1.11 – Localisation d'un point par rapport à un CSG

Si un point est localisé par rapport à 2 domaines A et B, alors les tables ci-dessus donnent la position du point par rapport au domaine résultant de l'intersection, l'union, ou la différence. Notons qu'il y a une ambiguité dans le cas Sur/Sur et il faut connaitre la position relative d'une frontière par rapport à l'autre (stocker la normale à la frontière n'est pas suffisant pour résoudre le problème dans le cas général).

Le calcul de l'aire ou du volume :

Le calcul de l'aire ou du volume est évident dans les cas particuliers où les primitives ne s'intersectent pas pleinement (ou s'incluent totalement), le calcul est impossible à réaliser dans le cas général sur la structure elle même.

1.3.3 La manipulation du C.S.G

Les opérations élémentaires

Les transformations géométriques élémentaires s'appliquent simplement aux feuilles de l'arbre : en modifiant la position des primitives (rotation, translation...) et leurs dimensions (changement d'échelle...).

Au niveau de la structure des données on peut modifier l'arbre, modifier les primitives ou leurs positions.

Les opérations booléennes

Les opérateurs booléens sont les opérateurs naturels pour manipuler le CSG. Ils ne nécessitent aucun algorithme à proprement parler. Ce sont ici des opérations élémentaires !

1.3.4 Les propriétés du modèle C.S.G

Les propriétés du modèle C.S.G sont les suivantes :

Le domaine : L'ensemble des objets représentés peut être très large, cela dépend principalement de la géométrie des primitives.

La complétude : Les méthodes pour la localisation d'un point sont simples et efficaces. En revanche le modèle n'est pas adapté aux calculs d'aire ou de volume.

L'unicité : L'unicité n'est pas vérifiée : il existe géneralement, même pour des géométries simples, plusieurs décompositions en primitives différentes. De plus, pour un même ensemble de primitives il existe plusieurs façons de les agencer. Le nombre d'arbres est proportionnel à celui de Catalan. On en trouve une démonstration dans [20]. Le nombre d'arbres géométriquement équivalents est généralement très inférieur.

La facilité de manipulation : Le C.S.G est un modèle très ergonomique et proposé dans l'interface utilisateur de tous les grands modeleurs 3D.

Les performances techniques : La précision et la concision dépendent de l'adéquation des primitives à la géométrie que l'on cherche à représenter.

1.3.5 Les conclusions

Le C.S.G est le modèle de prédilection de la C.A.O mécanique mais il peut aussi être utilisé dans d'autres domaines d'application. C'est le modèle le plus ergonomique, il est généralement utilisé avec un modèle Brep calculé automatiquement.

1.4 L'énumération spatiale

L'espace est découpé en cellules dont les faces sont perpendiculaires aux axes. Elles sont localisées par rapport à l'objet : elles sont à l'intérieur (pleines) ou à l'extérieur (vide) ou encore dans certains cas à cheval (partiel). Quelque soit le type de découpage, l'espace d'étude (représenté par le modèle) est limité, généralement à un parallélépipède rectangle. Un système de coordonnées local, propre à cet espace est utilisé : il est généralement discrêt (il est adressé par des entiers naturels).

On peut distinguer 3 types de modèles d'énumération spatiale :

La grille régulière : chaque cellule est identique, par exemple les pixels d'un écran raster, les voxels d'une image 3D.

"Le quadtree", "l'octree" : c'est une extension du cas précédent puisque l'on admet des niveaux de découpage différents. Les cellules sont identiques à un rapport près (en 2^d où d est la dimension de l'espace).

Le polytree ou PM tree [3] : c'est une extension du cas précédent puisque l'on admet que certaines cellules soient redécoupées suivant des schémas simples qui ne sont pas des plan perpendiculaires aux axes.

1.4.1 La représentation informatique

La grille régulière peut être représentée par un tableau de booléens de dimension 2 ou 3 suivant la dimension de l'espace.

Le quadtree est un arbre quaternaire dont chaque feuille contient l'information plein ou vide. Un exemple est donné figure 1.13. L'octree est l'extension du même schéma au 3D (arbre octal). Dans un espace à dimension d les noeuds de l'arbre ont 2^d fils.

Exercice 7 *Complétez le schéma du quadtree de l'exemple de la figure 1.13.*

Le polytree est un arbre quaternaire ou octal dont chaque feuille contient l'information plein ou vide ou mixte. Dans ce dernier cas la feuille doit contenir le modèle élémentaire de frontière. Par exemple en 3D une cellule mixte contient :
- soit un sommet et 1 seul (et toutes les arêtes incidentes) ;
- soit une portion d'arête (et les 2 faces incidentes) ;
- soit une portion de face.

C'est un modèle mixte entre octree et Brep.

De nombreuses structures informatiques permettent de représenter les modèles d'énumération spatiale. Notons seulement que le quadtree (l'octree...) peut être représenté sous la forme d'une liste de cellules pleines : pour chaque cellule le chemin qui permet d'y accéder est stocké. Le chemin est la suite de chiffres

[3] PM tree : PM pour Polygonal Map

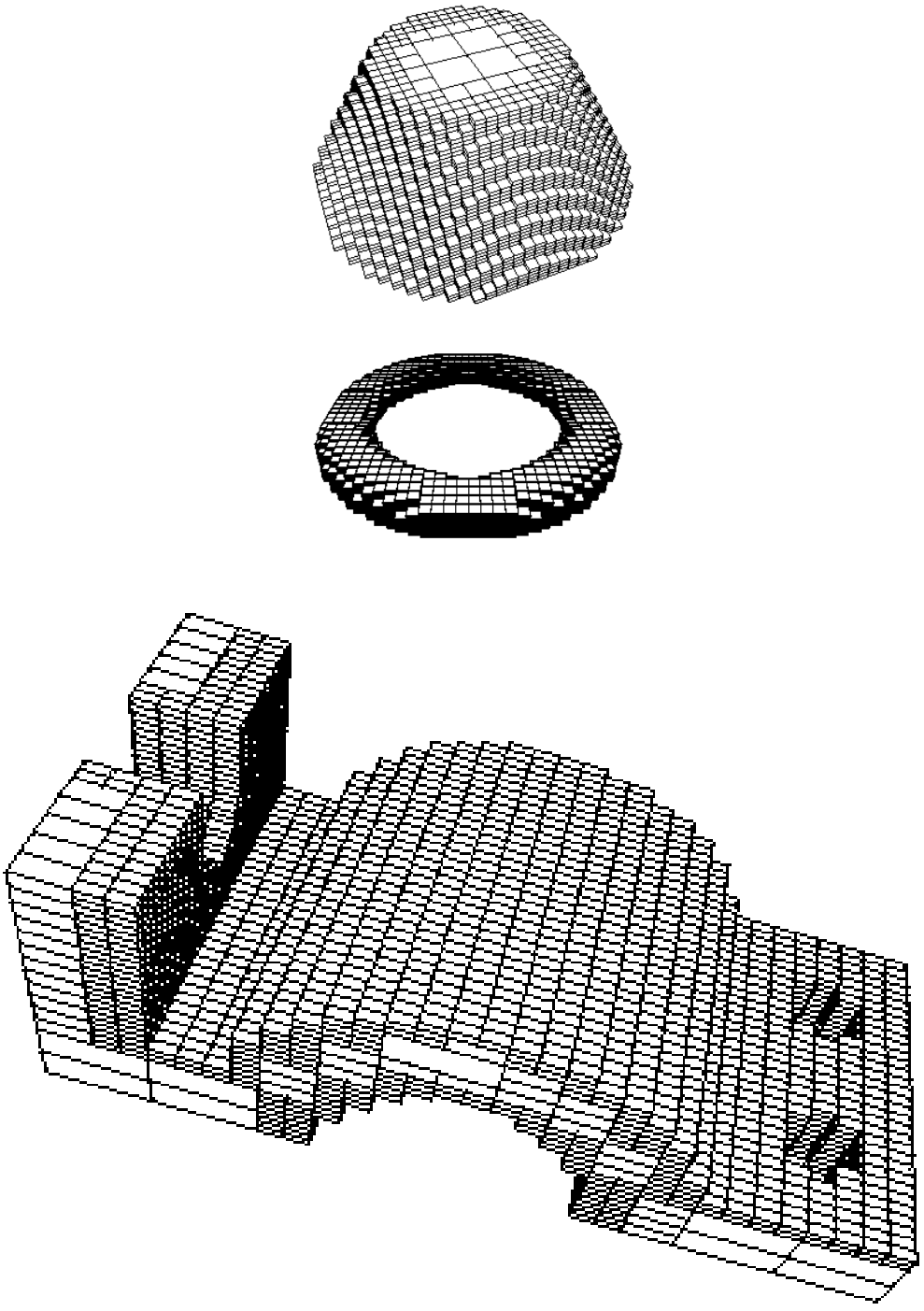


FIG. 1.12 – Exemples d'octrees de géométries simples

(compris entre [0 :3] pour un quadtree, entre [0 :7] pour un octree) chacun correspondant à un embranchement à un niveau de l'arbre (à un fils en un noeud de l'arbre). Quand une cellule pleine n'est pas une feuille des 'X' sont ajoutés. Cette structure n'est pas optimale pour l'accès mais bien adaptée pour le stockage car relativement concise. Le codage de l'exemple ci-dessous correspond à la géométrie (et à la numérotation des fils) de la figure 1.13.

`0XX,101,102,103,111,113,121,122,123,131,132...`

Exercice 8 *Complétez le codage de l'exemple ci-dessus correspondant à la géométrie de la figure 1.13.*

Exercice 9 *Sur l'exemple de la figure 1.13, la numérotation des fils privilégie les x. Montrer que pour la géométrie présentée sur la même figure, la numérotation privilégiant les y donne la même décomposition.*

Exercice 10 *Montrez que la taille d'un fichier (binaire) de stockage peut être égal à $n * p * d$ bits si n est le nombre de cellules pleines, d la dimension de l'espace et p la profondeur de l'arbre.*

1.4.2 Les algorithmes élémentaires

La localisation d'un point :

Le calcul est trivial sur une grille régulière et de complexité optimum ($O(1)$). En effet, il suffit de déterminer la case de la grille contenant le point. Une transformation simple permet de passer des coordonnées du point ((x, y) en 2D) dans l'espace réel (une fenêtre $(x_{min}, y_{min}, x_{max}, y_{max})$) aux coordonnées (i, j) dans l'espace discrêt (le tableau de nm cases : $[0 : n-1][0 : m-1]$) par la formule :

$$\begin{aligned} i &= n\frac{(x-x_{min})}{(x_{max}-x_{min}} \\ j &= m\frac{(y-y_{min})}{(y_{max}-y_{min}} \end{aligned} \tag{1.19}$$

Le calcul sur le quadtree ou l'octree nécessite un parcours de l'arbre avec un test élémentaire à chaque noeud (voir figure 1.14). Sa complexité est en $O(p)$ pire cas ou p est la profondeur de l'arbre. De même que pour la grille, le passage d'un point (de coordonnées (x, y) en 2D) de l'espace réel à un quadtree de profondeur p se fait par une transformation simple : il suffit d'utiliser le codage binaire (sur p bits de (y, x) dans notre exemple) pour déterminer le chemin menant à la cellule contenant le point.
Démontrons le simplement à l'aide d'un algorithme recursif : si l'on suppose un arbre de profondeur p, au premier niveau les coordonnées x_i du point P seront comparées avec les coordonnées du centre de la cellule C_p : 2^{p-1}. Soit C_{p-1} la cellule contenant P, plaçons nous dans son repère, c'est à dire soustrayons son origine si nécessaire : si $(x_i >= 2^{p-1}$ alors $(x_i = x_i - 2^{p-1})$. Au niveau inférieur les nouvelles coordonnées de P seront comparées avec les coordonnées du centre de la cellule C_{p-1} : 2^{p-2}... Et ainsi de suite jusqu'à la feuille...

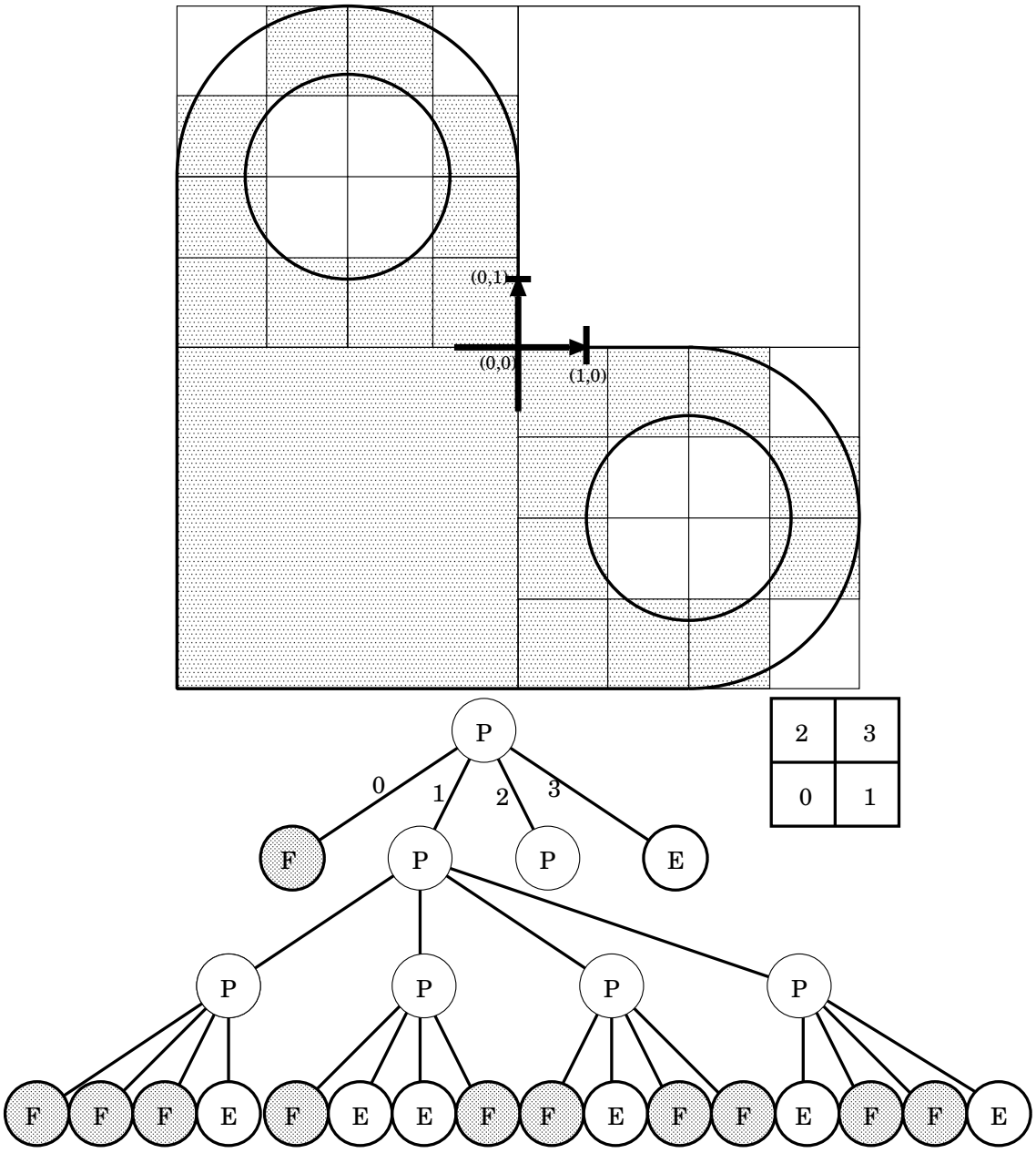


FIG. 1.13 – Exemple de quadtree sur une géométrie simple
Sur la figure du haut le grisé représente la géométrie du quadtree. Sur la figure du dessous l'arbre quaternaire est laissé incomplet pour des raisons évidente d'encombrement.

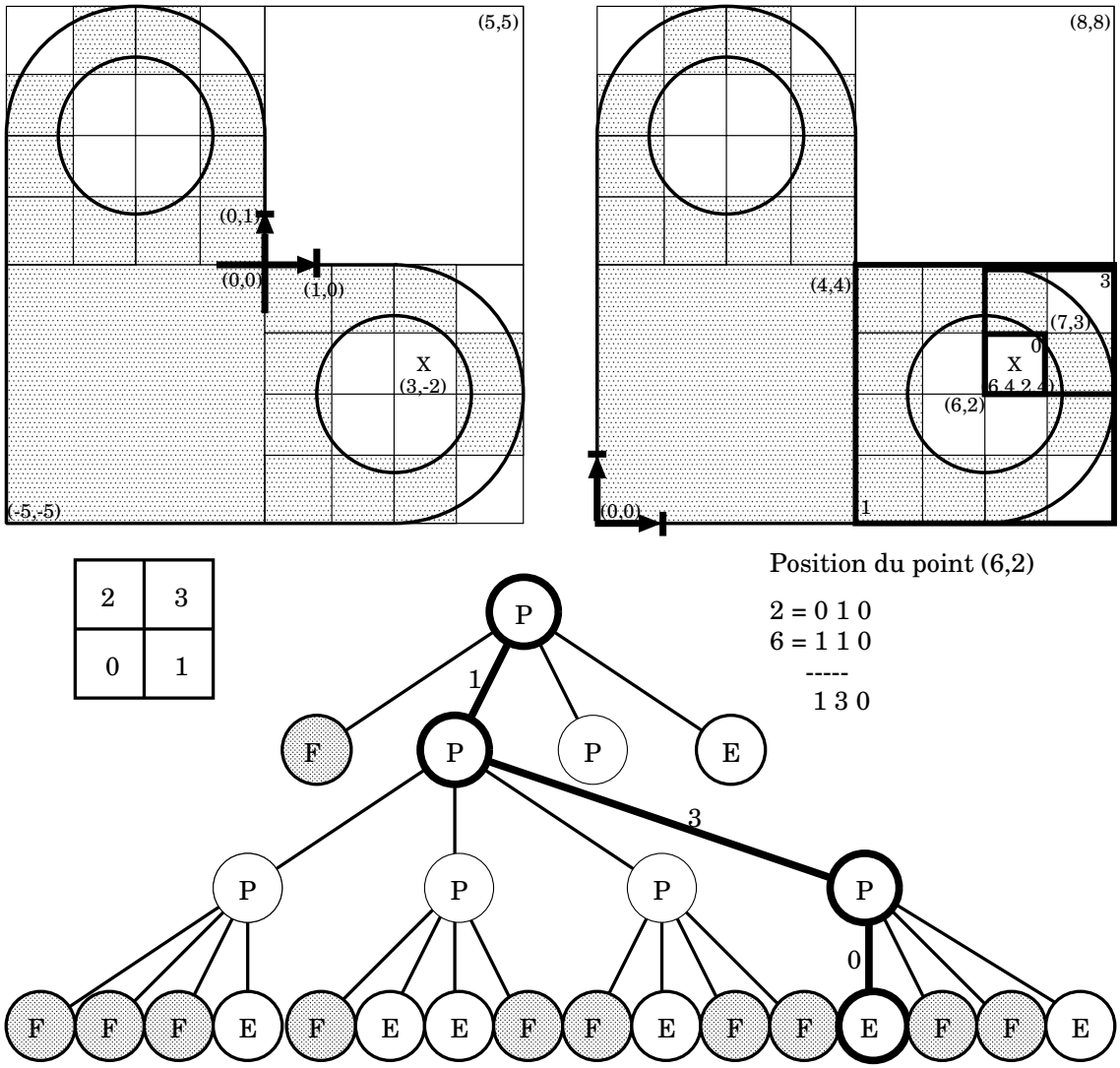


FIG. 1.14 – Localisation d'un point dans un quadtree

Le calcul sur le quadtree (ou l'octree) nécessite un parcours de l'arbre avec, à chaque noeud, un test pour savoir de quel coté du centre se trouve le point à localiser. La structure même du quadtree (ou de l'octree) impose une relation simple entre tous les centres des quadrants. Cette relation est même élémentaire dans l'espace de référence $E_r = [0, 2^p][0, 2^p]$ où p est la profondeur de l'arbre. Pour tester un point il suffit donc de le ramener dans l'espace de référence et de le coder sur p bits comme l'illustre l'exemple.

Exercice 11 *Peut-on identifier des points sur la frontière d'un objet représenté par une grille ou un quadtree ?*
En déduire la topologie des cellules des modèles d'énumération spatiale ?

Le calcul sur le polytree est identique au précédent à l'exception des feuilles mixtes ou le calcul est plus ou moins trivial.

Intersection avec une droite :

L'algorithme parcourt les cellules contenant la droite, et si ces cellules sont pleines, elles appartiennent à l'intersection.

Dans le cas d'une grille 2D le parcours d'une droite s'apparente au tracé d'un segment (chaque cellule de la grille est un pixel sur l'écran) dont l'algorithme est donné pour une droite $y = ax + b$ dans le cas où $0 < a < 1$:

```
WritePixel(i0,j0)
d=a*(i0+1)+b-(j0+1/2)
for(i=i0+1;i<nb;i++){
  if( d<0 )d=d+a
  else d=d+a-1
       j=j+1
  WritePixel(i,j)
}
```

C'est un algorithme connu sous le nom de "midpoint line algorithm". Il fonctionne par balayage (de l'axe x si $0 < a < 1$ de l'axe y sinon) : on part d'un point de la droite (i_0, j_0) et l'on incrémente i. Il peut être réécrit pour devenir très efficace et ne manipuler que des entiers (voir [12] page 78).

Exercice 12 *Réecrire l'algorithme ci-dessus pour qu'il ne manipule plus que des entiers.*

Exercice 13 *Modifiez l'algorithme pour tester l'intersection de la droite avec le modèle.*

Le calcul de l'aire ou du volume :

Le calcul est trivial sur une grille régulière et en $O(I * J)$ en 2D, $O(I * J * K)$ en 3D, où I, J, K sont respectivement le nombre de lignes, de colonnes et de rangées de la grille.

Le calcul sur le quadtree ou l'octree nécessite un parcours des feuilles de l'arbre. Il est donc en $O(n)$ ou n est le nombre de feuilles de l'arbre.

Le calcul sur le polytree est identique au précédent à l'exception des feuilles mixtes ou le calcul est plus ou moins trivial.

1.4.3 La manipulation des modèles

Quelque soit le modèle d'énumération spatial, l'opération booléenne ne peut être réalisée que si l'espace représenté est identique (une même fenêtre en 2D).

Dans le cas de grilles de même résolution (même nombre de lignes et de colonnes en 2D), le calcul d'une opération booléenne est réalisé en parcourant les tableaux et en effectuant les opérations directement sur les cellules correspondantes. Par exemple, si l'on note G_A (G_B) la grille représentant la géométrie A (resp. B), la grille représentant l'union est donné par :

$$G_{A \cup B}(i,j) = G_A(i,j) \cup G_B(i,j) \tag{1.20}$$

Exercice 14 *Écrire l'algorithme d'union de 2 grilles de "résolution" différentes en 2D.*

Pour le quadtree et l'octree l'algorithme est aussi assez simple : il est identique à la grille pour le cas des feuilles de même profondeur mais il faut aussi traiter le cas des noeuds. Par exemple, l'arbre C résultant de l'union de 2 arbres A et B est construit de la façon suivante :

```
 union( Ca , Cb )
/***************/
if( vide(Ca) ) return(Cb);
if( vide(Cb) ) return(Ca);
for(i=0;i<2^d;i++)fils(Cab,i)=union(fils(Ca,i),fils(Cb,i));
return(Cab);
```

Exercice 15 *Optimiser l'algorithme précédent pour éviter d'avoir toutes les feuilles d'un même sous arbre identiques (pleines ou vides).*

Exercice 16 *Montrer que les opérations booléennes n'ont pas besoin d'être régularisées.*

Exercice 17 *Trouver des exemples où les opérations booléennes en 3D ne donnent pas un résultat avec une frontière 2-manifold.*

Le calcul sur le polytree est identique au précédent à l'exception des feuilles mixtes ou le calcul est plus ou moins trivial.

D'autres opérations comme la rotation en général (d'un angle qui n'est pas multiple de 90 degrés), le changement d'échelle (d'un facteur qui n'est pas une puissance de 2) , évidentes sur d'autres modèles peuvent ici poser de sérieux problèmes.

1.4.4 Les propriétés des modèles d'énumération

Les propriétés des modèles d'énumération spatiales sont les suivantes :

Le domaine : L'ensemble des objets représentés peut être très large, cela dépend principalement de la précision autorisée.

La complétude : Les méthodes pour la localisation d'un point sont très simples. Le modèle est bien adapté aux calculs d'aire ou de volume.

L'unicité : L'unicité n'est pas vérifiée à cause de la sensibilité des modèles à l'orientation de l'objet.

La facilité de manipulation : simple pour certaines opérations (opérations booléennes) pose des problème pour d'autres (rotation).

Les performances techniques : Pour les grilles la structure est généralement volumineuse surtout en 3D. La précision est proportionnelle à la taille de la structure ! En revanche les algorithmes sont simples et efficaces, ils peuvent souvent être cablés. Pour les quadtrees et les octrees ces remarques deviennent plus nuancées pour devenir tout à fait discutables pour les polytrees.

1.4.5 Les conclusions

Les grilles sont très utilisées dans le domaine de l'imagerie 2D et 3D (l'imagerie médicale comme pour la RMN, l'imagerie géophysique comme pour la réflexion sismique, la géostatistique...). Les quadtrees et surtout les octrees sont utilisés pour optimiser de nombreux calculs géométriques (recherche des intersections par exemple). Ils sont rarement utilisés comme des modèles ou alors en combinaison avec d'autres.

Les limitations imposées aux objets (rigidité, homogénéité) ne sont pas naturellement respectées par le modèle.

À l'exception du polytree les objets sont représentés dans des espaces discrets. Les définitions des voisinages, de la connexité, du théorème de Jordan s'expriment sur des pavages de l'espace (topologie discrète de Khalimsky) et prennent une toute autre forme. On trouvera ces définitions dans des traités d'analyse d'images ou de morphologie mathématique [9].

1.5 Comparaison et conversion des représentations

Comme nous l'avons vu toutes les représentations ne sont pas équivalentes. La tableau récapitulatif montre entre autre que le CSG est le modèle le plus interactif et que la Brep est le modèle qui offre le plus large domaine de représentation.

	Domaine	Cal. Mass.	Interact.	G. Automat
CSG	large (<Brep)	difficile	facile	Pas d'algo
Brep	large	facile	difficile	difficile
Quadtree	large (<Brep)	facile	très difficile	difficile
PMtree	large (=Brep)	facile	très difficile	difficile

Les représentations solides ayant des qualités complémentaires il est logique que de nombreuses personnes aient travaillé sur les algorithmes de passage de l'un à l'autre.

De ce point de vue aussi les représentations ne sont pas équivalentes. Il existe une hierarchie : certains modèles contenant en fait plus d'informations que d'autres ils ne peuvent être générés à partir de ces derniers. Le CSG en est un bon exemple.

$$CSG > Brep \simeq PMTree \leq Quadtree, grille$$

Certaines transformations peuvent dégrader la précision de la représentation initiale comme par exemple le passage d'une Brep à un Quadtree.

La section suivante donne les principes des principaux algorithmes de transformation.

1.6 La construction d'un modèle d'énumération spatiale

La grille, le Quadtree ou le PMTree peuvent être construits à partir d'une Brep ou directement à partir d'un CSG.

1.6.1 Deux algorithmes généraux

L'algorithme le plus simple à mettre en oeuvre pour les quadtree, octree et PMtree est certainement le suivant.

Notons p_{max} la profondeur maximum de l'arbre et t_{min} la taille d'une cellule pour cette profondeur. La classification d'une cellule (quadrant pour $d = 2$ ou octant pour $d = 3$) par rapport au domaine : DANS, HORS, INTERSECTANT ou INDETERMINÉ conditionne le découpage.

Une cellule n'est pas redécoupée, si elle est classée DANS ou classée HORS. De même une cellule n'est pas redécoupée si elle est classée INTERSECTANT **et** que l'approximation de la frontière est acceptable ou que la taille minimum des cellules est atteinte ($t = t_{min}$), Dans tous les autres cas la cellule est redécoupée en 2^d nouvelles cellules qui sont de nouveau testées.

Une première façon de déterminer la position d'une cellule par rapport au domaine est de tester la position de ses coins (voir par exemple [2]). Ce test est rapide mais ne peut être satisfaisant que si la taille de la cellule est assez petite ($t < t_{max}$) par rapport à la géométrie à tester. Les cellules sont alors classées :

DANS : si tous les coins sont à l'intérieur du domaine et si la taille t de la cellule est telle que $t < t_{max}$.

HORS : si tous les coins sont à l'extérieur du domaine et si la taille t de la cellule est telle que $t < t_{max}$.

INTERSECTANT : si des coins sont à l'extérieur du domaine et d'autres à l'intérieur.

INDETERMINÉ : dans tous les autres cas.

Une deuxième façon de déterminer la position d'une cellule par rapport au domaine est de tester la position de son centre par rapport au domaine dilaté et érodé[4] de la taille de la cellule (voir par exemple [25] ou [23]).

DANS : si le centre de la cellule est à l'intérieur du domaine érodé de t.

HORS : si le centre de la cellule est à l'extérieur du domaine dilaté de t.

INTERSECTANT : si le centre de la cellule est à l'intérieur du domaine dilaté de t et à l'extérieur du domaine érodé de t et si des coins sont à l'extérieur du domaine et d'autres à l'intérieur.

INDETERMINÉ : dans tous les autres cas.

Finalement, si l'indétermination subsiste à la profondeur p_{max} on a recours au opérations booléennes régularisées pour le calcul exacte de l'intersection.

Cette seconde technique a l'avantage de classer rapidement les cellules DANS et HORS alors qu'il fallait dans l'algorithme précédent atteindre la taille t_{max} sans savoir si cela était suffisant (car les problèmes peuvent subsister) ou nécessaire (sous arbres dont toutes les feuilles sont DANS ou HORS et qu'il faut supprimer).

Pour appliquer le premier algorithme à un CSG ou à une Brep il suffit d'utiliser la procédure de classification ad-hoc d'un point par rapport au modèle.

En revanche le deuxième algorithme demande en plus le calcul des opérations booléennes et surtout le calcul du domaine érodé et dilaté qui n'est pas trivial. Cet algorithme peut être envisagé sur un CSG si chaque primitive admet des opérations (même approximatives) d'érosion et de dilatation. En revanche il est difficile de l'appliquer à une Brep dont l'érosion et la dilatation peuvent changer la topologie (voir les réseaux bissecteurs [21]).

1.6.2 Algorithmes spécifiques de construction à partir d'une Brep

Le premier algorithme de la section précédente fonctionne entre autre sur la Brep, mais il est général et n'utilise pas explicitement le modèle. Ici, l'approche repose sur le parcours de la frontière ; elle est donc spécifique aux Brep. Le principe est simple, on détermine dans le modèle spatial les cellules que la frontière traverse. On s'arrange pour que l'ensemble des cellules traversées forment une frontière (au sens discrêt). Le théorème de Jordan "discrêt" permet ensuite de

[4] L'érosion et la dilatation sont defini par un élément structurant (ici la cellule) repéré par son centre et que l'on déplace dans l'espace [11]

définir et de parcourir l'intérieur du domaine.

Dans le cas où l'on cherche à construire une **grille 2D**, les problèmes peuvent se ramener au "tracé" de segments (voir le "midpoint line algorithm") et de remplissage de polygones. Ce sont des traitements classiques pour l'affichage sur écran raster (à balayage de trame) que l'on trouvera traité dans tous les ouvrages abordant le graphisme sur ordinateur (consulter par exemple les chapitres 3 et 19 de [12]).

Dans le cas où l'on cherche à construire un **Quadtree** un **Octree**, ou un **PMtree** il faudra insérer chaque sommet, chaque arête et chaque face dans le modèle. Pour chaque insertion, il faudra calculer l'intersection de l'élément de frontière avec la cellule qui le contient et redécouper cette cellule si nécessaire. Pour de plus amples détails, on pourra consulter [3].

1.7 La construction de la Brep

1.7.1 Construction à partir d'un CSG

Pour passer d'un CSG à une Brep, la méthode la plus simple consiste à calculer la Brep de chaque primitive puis à parcourir l'arbre pour appliquer les opérateurs booléens présents aux noeuds. Notons que le parcours de l'arbre est récursif non terminal (identique au parcours pour la classification d'un point).

1.7.2 Construction à partir d'une grille

Il arrive que les données géométriques initiales soient sous la forme d'une grille 2D ou 3D de cellules informées (chaque cellule dispose d'une valeur). Nous allons voir comment on peut extraire une frontière.

On peut considérer que chaque cellule a une valeur binaire DANS ou HORS. En effet, si les cellules ont une valeur réelle, et que l'on cherche l'isosurface $S(x, y, z) = V_c$, on peut se ramener au cas précédent en seuillant par la valeur de coupure (V_c).

Considérons la grille duale : les cellules deviennent des sommets de la nouvelle grille. Les méthodes d'extraction ont pour principe de parcourir l'ensemble des cellules élémentaires de cette nouvelle grille et, pour chacune d'elle, d'établir en fonction de la valeur de ses coins, la portion de surface qui l'intersecte.

Une cellule de frontière est une cellule dont certains coins sont classés HORS et d'autres sont classés DANS. Les morceaux de surface sont supposés plan dans chaque cellule.

Deux problèmes, d'ordre différents mais tous deux liés à la précision relative de la grille, peuvent apparaitre :

- Le cas d'un objet dont la frontière couperait plusieurs fois le coté d'une même cellule ne peut être considéré. Le détail n'apparait pas au niveau de la grille, il n'apparaitra pas non plus sur la représentation frontière.

– En 2D le cas où les valeurs DANS et HORS sont situées sur les diagonales d'une cellule est un cas ambigu : deux topologies différentes peuvent donner ce résultat. Le choix est donc plus ou moins arbitraire. En 3D le même cas se pose évidemment sur les faces des cellules.

La méthode "marching cube" proposée par [31] associe à chaque cube une signature sur 8 bits (un bit par coin : 1 si le coin est DANS, 0 si il est HORS). Le nombre de configuration est de $256 = 2^8$ mais pour des raison de symétrie leur nombre peut être réduit à 15 cas (voir figure 1.15).

Cependant, l'ambiguïté relevé précedement pour le cas 2D introduit, en 3D, des discontinuités dans la surface : les cas 6, 7, 12 et 13 ont des symétriques qui ne se raccordent pas. Il faut définir de nouveaux cas : 6', 7', 12' et 13' qui se raccordent mais ne sont pas des symetriques. Le nombre de configurations passerait à 19. Une autre technique simple et efficace consiste à découper les cellules en simplexes (triangle en 2D, tétraèdres en 3D). L'ambiguité n'est pas levée mais la diagonale (du quadrangle) impose un choix arbitraire sans équivoque et assure la cohérence.

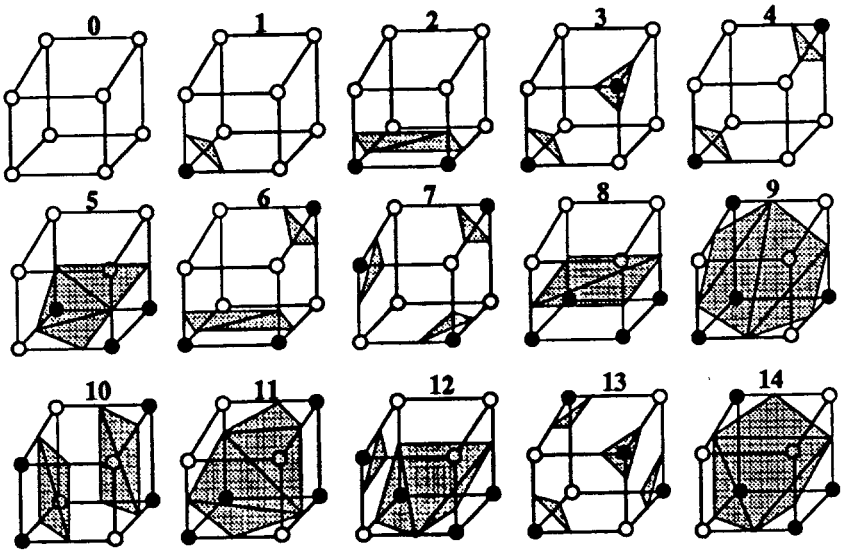


FIG. 1.15 – Les modèles du marching cube
Portions d'isosurface des 15 configurations proposées par Lorensen. Notez que les cas 6, 7, 12 et 13 ont des symétriques qui ne se raccordent pas correctement.

Chapitre 2

Algorithmique géométrique

2.1 Introduction

L'algorithmique géométrique introduit entre autre par Shamos et Preparata en 1978 est une discipline assez jeune et probablement plus connue sous le terme de **Computational Geometry**. Les problèmes abordés sont des problèmes de localisation, de recherche dans l'espace Euclidien, de détection de proximité. Cette discipline étudie les algorithmes en terme de complexité et de fiabilité. Elle a de nombreuses applications dans le domaine de la CAO, de la recherche opérationnelle, de la robotique et du graphisme sur ordinateur.

2.1.1 Exemples d'applications et problèmes types

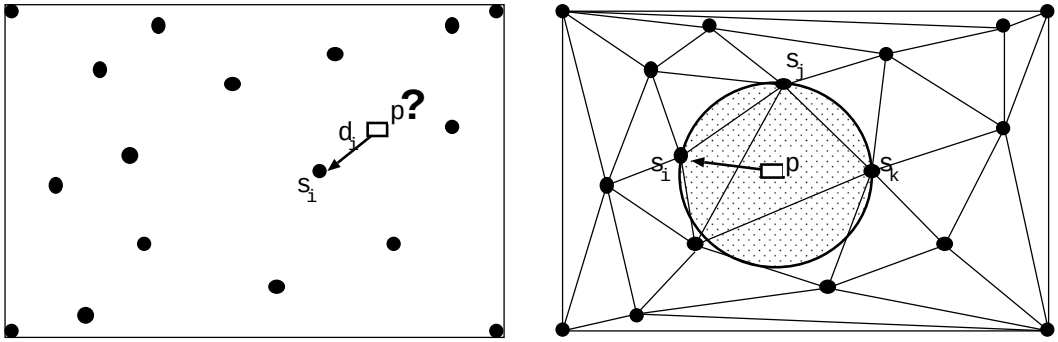


FIG. 2.1 – Le problème de "l'usine polluante"
Où placer une usine polluante p, afin de maximiser sa distance (d_i) avec des villes (les sites s_i) ? Le problème géométrique consiste à trouver le plus grand cercle vide. Il peut être efficacement résolu par la triangulation de Delaunay des sites.

En recherche opérationnelle : Comment relier des stations en minimisant le cable nécessaire ? C'est le problème de l'arbre minimal recouvrant.
Où installer une usine polluante pour l'éloigner des villes ? C'est le problème du plus grand cercle inscrit (voir figure 2.1).
Ces 2 problèmes peuvent être abordés à partir des triangulations de Delaunay.

En modélisation géométrique : elle permet de reconstruire des géométries valides à partir de données incomplètes (nuage de points irréguliers, arêtes imposées...). Les triangulations de Delaunay (adaptatives ou non) sont souvent à la base de ces algorithmes.

Pour les systèmes graphiques interactifs : la désignation à l'écran fait appel à une requete point ou une requete boite. Elle est traitée à l'aide de structures d'accès adaptées comme les quadtree, les kd tree...

En algorithmique géométrique on distingue habituellement 3 catégories de problèmes :

Les problèmes de localisation et de recherche : position d'un point par rapport à un polygone, recherche des points contenus dans une boite...

Le problèmes du calcul de l'enveloppe convexe : L'enveloppe convexe d'un nuage de points dans le plan (ou dans l'espace) est le plus petit polygone convexe (resp. polyèdre en 3D) contenant les points.

Les problèmes de calcul de proximité : qui sont résolus par des structures comme les diagrammes de Voronoi et les triangulations de Delaunay...

2.1.2 Évaluation des algorithmes

Le comportement des algorithmes peut être étudié de plusieurs points de vues qui dépendent des caractéristiques de l'application considérée.

Pour une application interactive : l'algorithme doit être rapide et dans la mesure du possible son temps de réponse doit être indépendant de la quantité des données.

Pour des traitements répétitifs : quand elles ne changent pas, les données sont préparées (ordonnancement, structure d'accès) de façon à optimiser certains traitements que l'on sait fréquents.

Pour manipuler des données changeantes : les structures mises en place pour optimiser les algorithmes doivent être mises à jour rapidement. On parle d'algorithmes dynamiques. Ils sont généralement plus complexes que les algorithmes statiques et ne seront pas abordés dans ce document. Mais l'ouvrage de Boissonnat et Yvinec [6] développe le sujet et pourra être consulté.

2.1.3 Les complexités

Un problème (un algorithme) nécessite des ressources pour être traité (resp. exécuté) : on parle de complexité. La complexité s'exprime généralement en fonction du nombre de données à traiter mais il arrive parfois qu'elle s'exprime aussi en fonction du nombre de solution au problème. Le terme de complexité doit être manipulé avec précautions, en fait il existe une terminologie précise :

La complexité en temps : donne le nombre d'opérations élémentaires nécessaire pour traiter les n données. Une opération est élémentaire si elle est indépendante des données.

La complexité en place : donne la place (mémoire) nécessaire pour traiter les n données.

La complexité pire cas d'un algorithme : donne les ressources nécessaires (temps ou place) pour traiter n données dans le cas où leur "configuration" est la plus défavorable à l'algorithme.

La complexité meilleur cas d'un algorithme : donne les ressources nécessaires (temps ou place) pour traiter n données dans le cas où leur "configuration" est la plus favorable à l'algorithme.

La complexité moyenne d'un algorithme : donne le coût moyen d'un algorithme sur un ensemble très large de données. Son évaluation est analytique dans les cas triviaux, généralement empirique mais aussi statistique.

La complexité d'un algorithme : donne les ressources nécessaires (temps ou place) pour traiter un problème donné avec l'algorithme.

La complexité d'un problème : donne la complexité (pire cas) minimale que peut atteindre un algorithme qui traite le problème.

La complexité s'exprime souvent comme une fonction polynomiale f du nombre de données n, on note : $O(f(n))$. Certains problèmes ont pourtant des complexités non polynomiales (exponentielles) comme le problème classique du voyageur de commerce. Ces problèmes sont classés NP[1].

L'exemple du tri

Prenons l'exemple du tri dans R.

La complexité du problème : $O(nlog_2 n)$, elle est atteinte par le "tri-fusion" entre autres.

La complexité en temps de l'algorithme trivial : $O(n^2)$. L'algorithme trivial recherche n fois le minimum dans le tableau et l'extrait.

La complexité meilleur cas en temps du tri par ventilation : $O(n)$ La ventilation v est une application qui associe directement un rang r à l'élément : $r = v(x), x \in R$. Dans le meilleur cas c'est une bijection, chaque élément x à un rang distinct et ordonné : appliquer v se fait en $O(n)$ et résoud le problème du tri.

[1] NP pour Non-Polynomiaux

La complexité pire cas en temps du tri par ventilation : $O(n^2)$. Généralement plusieurs éléments ont une même image par v c.a.d. un même rang ! Dans le pire cas v ne permet de trier qu' un élément de l'ensemble. L'opération de ventilation (avec une nouvelle fonction) doit donc être répétée... jusqu'à n fois dans le pire cas.

Pour plus de détails on pourra se référer à [13].

2.1.4 Techniques algorithmiques

Les principales techniques algorithmiques sont :

les techniques incrémentales : Soit un ensemble N de n_N données. La phase initiale calcule une solution S (souvent triviale) pour un sous-ensemble $N_s \subset N$ de petit cardinal ($n_{N_s} = m$ avec $m = 2$ ou 3). Puis commence la phase incrémentale : on "active" une nouvelle donnée (dans un ordre fixe : approche déterministe ou dans un ordre aléatoire) et l'on met à jour la solution. Le cardinal des données traitées est incrémenté : $n_{N_s} = m + 1$. L'opération est répétée ($n_N - m fois$) jusqu'à ce que toutes les données soient traitées : $N_s = N$ ($n_{N_s} = n_N$).

les techniques par balayage : Le principe est de "balayer" l'espace et de déterminer à chaque étape quelles données peuvent être concernées par le traitement. On dit que ces données sont "actives". Une structure d'accès aux données "actives" est maintenue afin de limiter encore le nombre de tests. L'algorithme de Bentley et Ottman (calcul d'intersections de segments dans le plan), est un exemple typique de technique de balayage.

l'approche diviser pour regner : (ou "divide and conquer"). Elle s'effectue en 2 phases. La première divise l'ensemble des données N en deux sous-ensembles de même taille $N_d \cup N_g = N$. Chaque sous-ensemble est de nouveau divisé ($N_{dd} \cup N_{dg} = N_d$) et ainsi de suite jusqu'à obtenir des "petits" sous-ensembles qui admettent des solutions triviales. Puis commence la phase de fusion des solutions : la solution de $N_{...dd}$ et celle de $N_{...dg}$ vont permettre de construire efficacement la solution de $N_{...d}$ ($N_{...d} = N_{...dd} \cup N_{...dg}$) et ainsi de suite jusqu'à obtenir la solution de N. C'est souvent l'approche optimale pour les problèmes statiques.

Exercice 18 *Déterminez les approches utilisées par les algorithmes de tri évoqués précédemment.*

2.2 La recherche géométrique

2.2.1 La localisation d'un point vis à vis d'un polygone

Le problème est de déterminer si un point de coordonnées fixées est à l'intérieur où à l'extérieur d'un polygone.

Il existe plusieurs algorithmes spécifiques qui sont adaptés aux caractéristiques des polygones à traiter : polygones simples, polygones étoilés, polygones convexes (voir figure 2.2).

Pour un polygone de n côtés, on peut trouver des algorithmes de localisation dont les complexités (en temps, en mémoire et en temps de préparation) sont les suivantes :

Polygone	Temps	Mémoire	Pré-traitement
Simple	$O(n)$	$O(n)$	-
Étoilé	$O(log_2 n)$	$O(n)$	$O(n)$
Convexe	$O(log_2 n)$	$O(n)$	$O(n)$

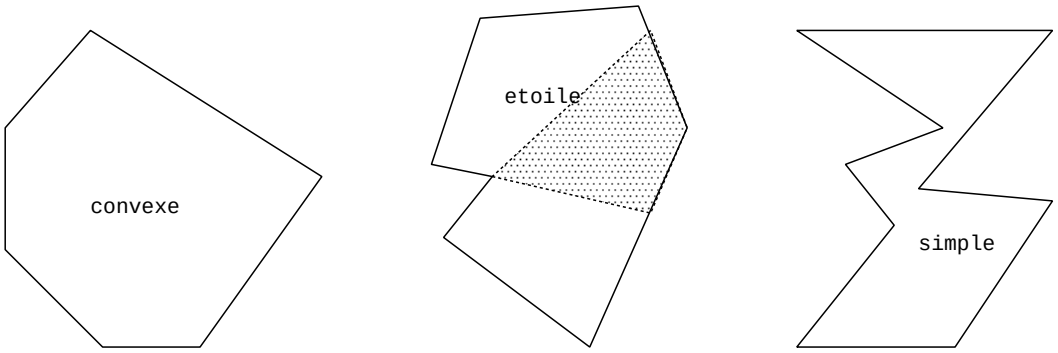


FIG. 2.2 – Quelques types de polygones

Un polygone simple partitionne l'espace en 2 régions (intérieur et extérieur). On appelle noyau d'un polygone l'ensemble des points intérieurs au polygone qui "voient" tous les sommets du polygone. Un polygone est convexe si son noyau est égal à son intérieur. Un polygone est étoilé si son noyau n'est pas vide.

On remarque que la complexité totale du traitement est identique dans les 3 cas (polygone simple, étoilé, convexe) : $O(n)$. En revanche quand un grand nombre de points ($m >> n$) doit être classé par rapport au polygone (traitements répétitifs) la complexité des algorithmes n'est plus équivalentes : $O(m * log_2(n)) + O(n) << O(m * n)$.

L'algorithme de localisation d'un point dans un convexe est présenté figure 2.3.

2.2.2 Les requêtes point, boite

Considérons un ensemble de points particuliers de l'espace (appelés sommets ou sites) $N = \{p \in E^d\}, Card(N) = n$. Le problème est d'obtenir efficacement le point de N de coordonnées fixées $x \in E^d$ (requête point) ou encore les points de N contenus dans un interval $[x, y] \in E^d \mathrm{x} E^d$ (requête boite).

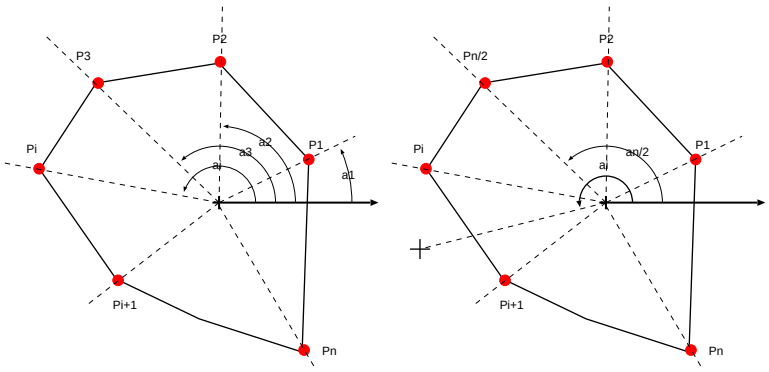


FIG. 2.3 – Point dans convexe

L'algorithme de localisation d'un point P vis à vis d'un polygone à n côtés fonctionne en 2 étapes. La première ordonne les secteurs correspondants à chaque segment et construit une structure d'accès en $O(n)$. La seconde étape détermine le secteur contenant le point (en $O(log_2 n)$ grâce à la structure d'accès) puis teste la position du point par rapport à la droite (portant le segment).

Pour ces recherches répétées, la mise en place de structures adaptées s'impose. Ce sont principalement des arbres. Un prétraitement construit donc la structure et trie les points de N dans cette structure. Le point recherché $x \in E^d$ est ensuite rapidement classé dans une cellule c de cette structure (en $O(log_2(n))$ par exemple pour un KDTree). Les coordonnées des sites dans la cellule c sont ensuite examinés. Ces structures s'écrivent en dimension quelconque d mais seront, pour des raisons pratiques, présentés ici en 2D. Il s'agit :

- des grilles (voir figure 2.4)
- des quadtrees (voir figure 2.5)
- des KDTrees (voir figure 2.5)

Pour les grilles et les quadtree, l'espace est d'abord limité à une fenêtre $((x_{min}, y_{min}, x_{max}, y_{max}))$ puis la précision de la structure est fixée : le nombre de lignes n et de colonnes m pour la grille, la profondeur maximum de l'arbre pour le quadtree (notée p). Il est alors possible de définir des fonctions de transfert permettant des classer les sites dans la structure mais aussi de réaliser les requêtes point en fonction de leurs coordonnées (x, y). Pour la grille la fonction est ventilation donnée par :

$$\begin{aligned} i &= n\frac{(x-x_{min})}{(x_{max}-x_{min}} \\ j &= m\frac{(y-y_{min})}{(y_{max}-y_{min}} \end{aligned} \tag{2.1}$$

Pour le quadtree il s'agit du codage des coordonnées du point sur p bits. Cependant, dans le cas du quadtree, il est nécessaire au préalable de construire l'arbre. L'algorithme est une procédure récursive qui redécoupe une cellule si elle contient plus d'un site (ou un nombre trop important de sites) jusqu'à atteindre, si nécessaire, la profondeur maximum.

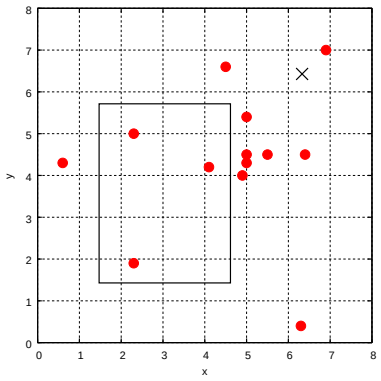


FIG. 2.4 – Requête boite sur une grille

Les intervals correspondants à la boite $[x_{min}, x_{max}], [y_{min}, y_{max}]$ sont calculés dans l'espace de la grille : $[i_{min}, i_{max}], [j_{min}, j_{max}]$. Les cellules correspondantes sont visitées et les sites contenus mis dans la liste. Sur les cellules de bord en revanche, les coordonnées des sites doivent être comparés aux coordonnées réelles de la boite.

Quand la répartition des sites est très hétérogène (de très fortes concentrations par exemple), il arrive que les grilles ou les quadtrees ne donnent pas de résultats satisfaisants. En effet une grille très fine consommerait trop de mémoire mais si elle ne l'est pas assez alors une grande quantité de sites peuvent se retrouver dans une seule cellule et la ventilation ne sert à rien. De même un quadtree deviendrait trop profond et le coût des requêtes pourrait être proportionnel au nombre de site (n).

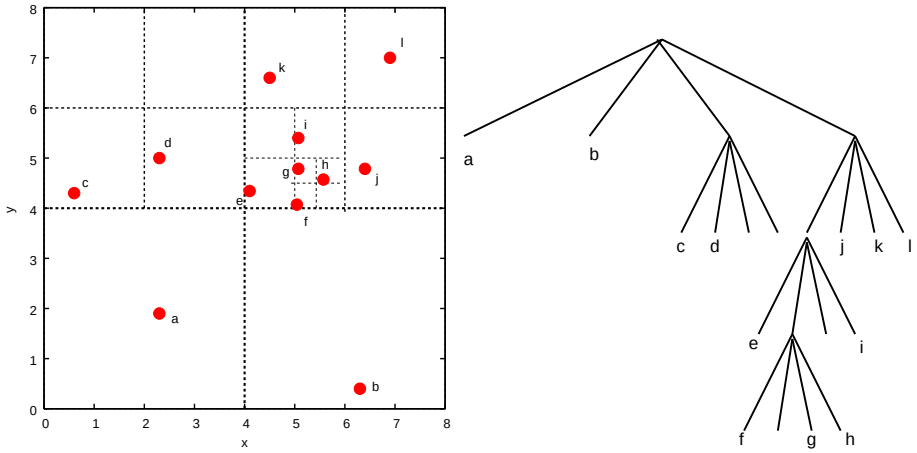
Le KDTree répond à ce problème car les droites séparant ses cellules ne suivent pas des coordonnées entières ou des puissances de 2. Ces droite D seront justement choisies pour séparer équitablement l'ensemble des sites en 2 sous-ensembles S_{D^+} et S_D) afin d'assurer que $Card(S_{D^+}) = Card(S_D) \pm 1$.

Le KDTree est un arbre binaire : à chaque niveau l'espace est séparé en 2 (en 4 pour un quadtree) suivant une seule coordonnées. Au niveau suivant la séparation se fera suivant l'autre coordonnée. L'arbre obtenu est équilibré et de profondeur $log_2 n$, une requête point est cette fois-ci réalisé en temps optimum, c'est à dire en $O(log_2 n)$ opérations.

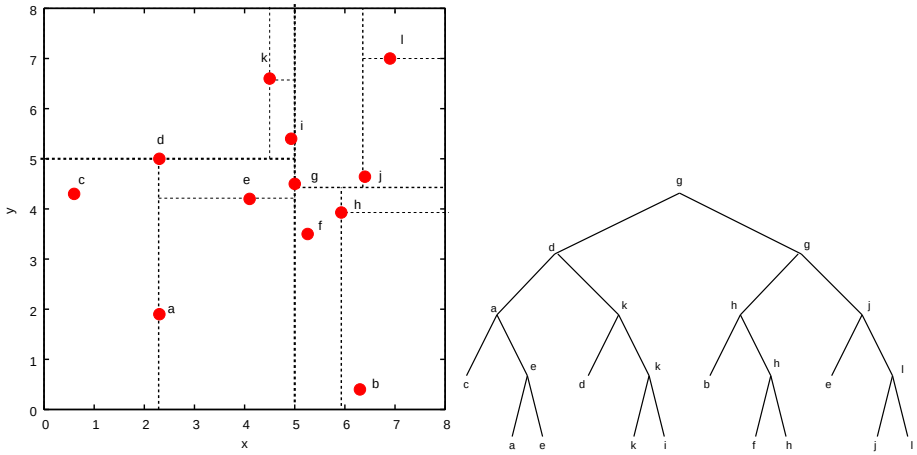
La construction d'un KDTree peut être réalisé en $O(nlog_2 n)$ opérations. Il s'agit alors de l'algorithme optimum.

2.3 L'enveloppe convexe d'un nuage de points

Le problème du calcul de l'enveloppe convexe d'un nuage de points est l'un des premiers problèmes posés en Algorithmique Géométrique. Ce n'est pourtant qu'en 1991 que fut présenté le premier algorithme optimal quel que soit la dimension.



a) Quadtree d'un ensemble de sites



b) KDtree d'un ensemble de sites

FIG. 2.5 – Comparaison du quadtree et du KDtree

Sur un même ensemble de sites, la quadtree et le KDtree donnent des résultats différents. Il faut noter que le KDTree a une profondeur minimum $log_2(12) = 3.58 \simeq 4$ et que le requêtes seront donc réalisées en temps optimum.

On trouvera un développement à cette section dans le chapitre 3 : "convex Hulls : Basic Algorithms" de [27], dans les chapitres 8 "enveloppe convexe incrémentale" et 9 (cas 2D et 3D) de [6] et dans les chapitres 1 (exemple) et 11 (cas 3D) de [5].

2.3.1 La complexité du problème

Théorème 4 *Le problème du calcul de l'enveloppe convexe d'un nuage de point de R^d a une complexité en temps de $O(nlog_2(n) + n^{(d/2)})$ et de $O(n^{(d/2)})$ en place.*

Vous trouverez une démonstration générale et rigoureuse du théorème dans [6].

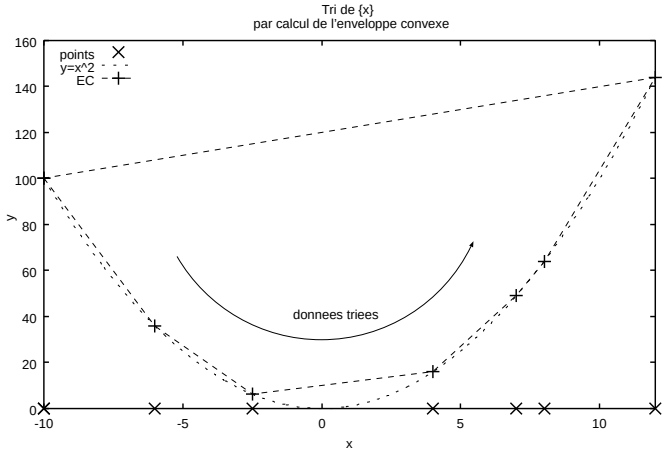


FIG. 2.6 – Tri et Enveloppe convexe

La donnée est un ensemble de points P=$\{x, x \in R\}$. L'idée est de passer en 2D : Q=$\{(x, y), x \in P et y = x^2\}$. Le calcul de l'enveloppe convexe de Q permet d'obtenir le tri de P : il suffit de parcourir le polygone obtenu dans le sens trigonométrique.

Dans le plan on démontre que le problème du tri dans R peut être résolu par le calcul de l'enveloppe convexe dans R^2 (comme l'illustre la figure 2.6). $Pb(tri) \subset Pb(EC)$ d'où $O(Tri) \leq O(EC)$. Réciproquement, l'algorithme de Graham montre que le tri résoud le problème du calcul de l'enveloppe convexe ; $Pb(EC) \subset Pb(Tri)$ d'où $O(EC) \leq O(Tri)$. La complexité des problèmes est identiques dans le plan : $O(tri) = O(EC) = O(nlog_2 n)$ (cqfd).

2.3.2 L'algorithme de Graham (2D)

Soit N un ensemble de points répartis dans R^2, l 'algorithme de Graham est le suivant :

1. Trouver $p \in NR^2$ un point intérieur à l'enveloppe convexe ;

2. Trier l'ensemble des points de N dans l'ordre lexico-graphique : angle polaire et distance à p ;
3. Construire une chaine circulaire doublement chainée : SUIV, PREC, DEBUT (DEBUT est un point extremum : par exemple le point d'ordonnée minimum)
4. Rendre le polygone convexe avec l'algorithme détaillé de la figure 2.7.

Exercice 19 *A quels types de polygones s'applique l'algorithme qui les rend convexes figure 2.7 ?*

L'algorithme de Graham peut être amélioré entre autre pour éviter le calcul des angles :
1. Rechercher les points extrêmes suivant l'axe des x, se placer dans le repère de cet axe et séparer les points au dessus des points au dessous de l'axe.
Puis pour chacun des 2 sous ensembles :
2. Trier l'ensemble des points de N dans l'ordre lexico-graphique : x,y
3. Construire une chaine circulaire doublement chainée.
4. Rendre le polygone convexe avec le même type d'algorithme que précédemment.
Finalement l'union des 2 enveloppes convexes est triviale.

L'algorithme de Graham fonctionne dans le plan mais ne se généralise pas aux dimensions supérieures. Ce n'est pas un algorithme incrémental.

2.3.3 Algorithme incrémental

Exercice 20 *Proposez un algorithme incrémental optimum dans le plan. Inspirez vous de la figure 2.8. Généralisez le au cas 3D.*

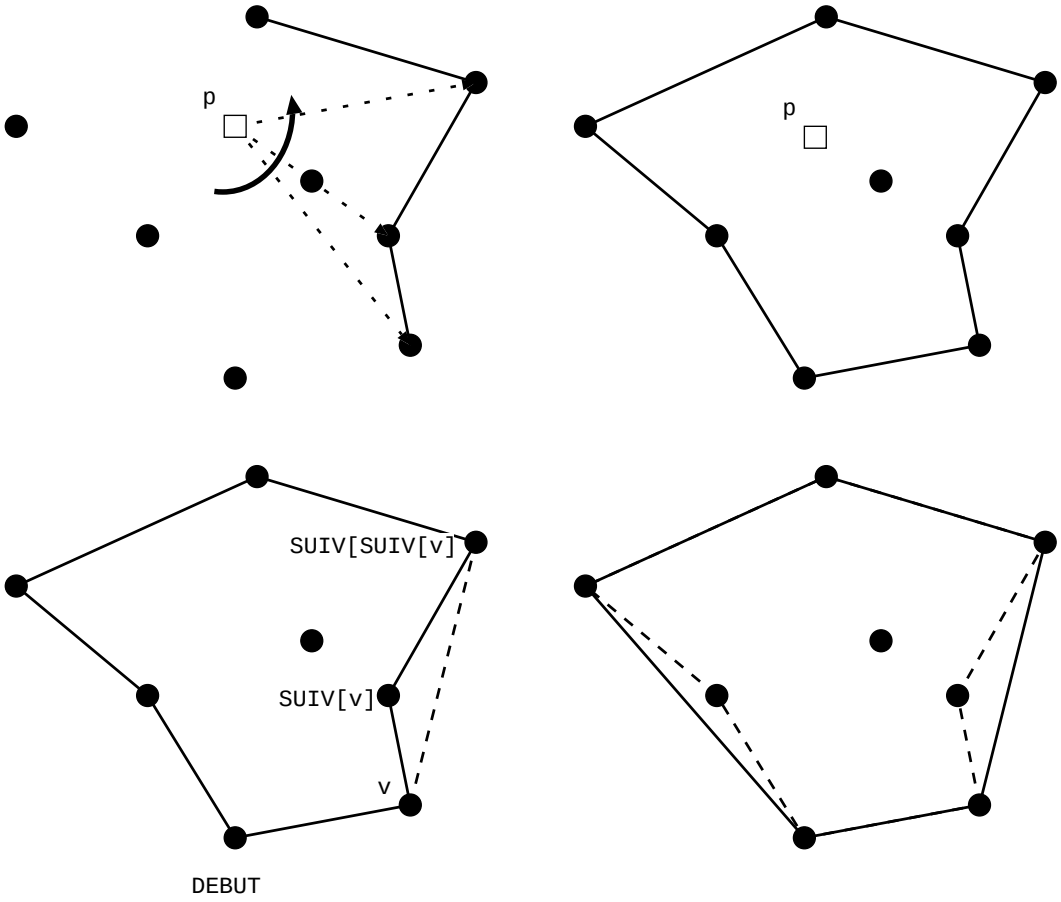
Vous trouverez dans [6] la descritpion d'un algorithme incrémental général (dans un espace de dimension d), et la démonstration de sa complexité :

Théorème 5 *L'algorithme incrémental construit l'enveloppe convexe de n points de R^d en temps $O(nlog_2(n) + n^{((d+1)/2)}$ et occupe $O(n^{(d/2)})$ de mémoire. Il est optimum pour les dimensions paires.*

2.3.4 L'algorithme diviser pour régner (2D)

Soit N un ensemble de points répartis dans R^d, le calcul de l'enveloppe convexe consiste à :
1. Diviser N récursivement en sous-ensembles N_i jusqu'à avoir $Card(N_i) <= (d+1)$ (on peut utiliser les kd-tree par exemple) ;
2. Pour chaque sous ensemble N_i : calculer l'enveloppe convexe ;
3. Puis réunir les enveloppes convexes 2 à 2 (l'algorithme 2D est détaillé figure 2.9).

Vous trouverez l'algorithme 3D dans [6] page 217 à 226.




```
RENDRE_POLYGONE_CONVEXE(DEBUT,PREC,SUIV)
/*******************************************/
/* DEBUT doit etre un point de l'EC        */
/* PrV est le produit vectoriel            */
/*******************************************/
{ v = DEBUT; w = PREC[v]; trouve = FAUX;
  while( (SUIV[v] != DEBUT) || (trouve == FAUX) ){
    if( SUIV[v] == w )trouve = VRAI;
    if( PrV((v,SUIV[v]),(SUIV[v],SUIV[SUIV[v]])) < 0 ){
      /* Enlever SUIV[v] de la liste */
      SUIV[v] = SUIV[SUIV[v]];
      PREC[SUIV[v]] = v
      v = PREC[v];
    } else {
      v = SUIV[v];
    }
  }
}
```

FIG. 2.7 – Algorithme de Graham pour rendre un polygone convexe

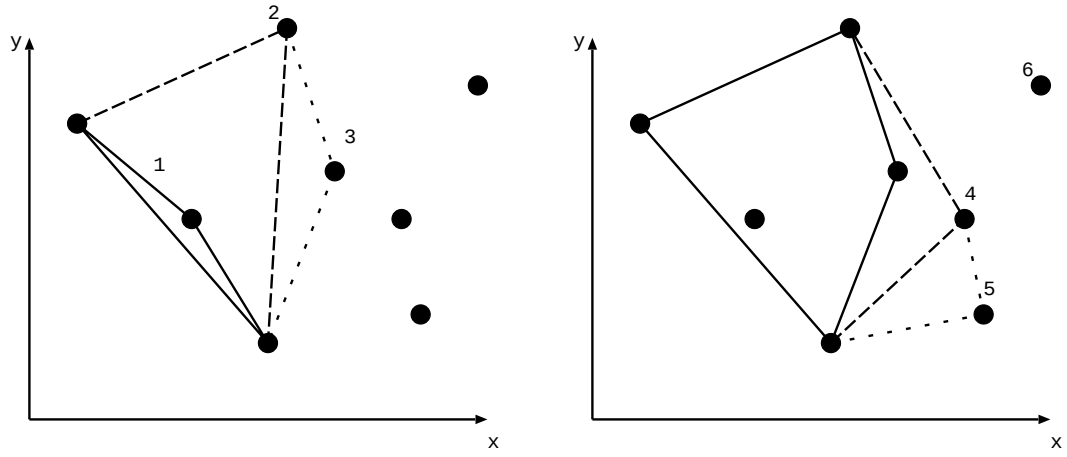


FIG. 2.8 – Algorithme incrémental de calcul d'EC

2.4 Les triangulations

2.4.1 Les propriétés

Dans un espace de dimension d (E^d), une triangulation (ou d-triangulation) est un ensemble de d-simplexes qui sont disjoints ou qui partagent une k-face avec $k \in [0; d[$. On trouvera une définition plus rigoureuse dans [6].

Définition 10 *Un d-simplexe (ou plus simplement simplexe) est l'enveloppe convexe de d+1 points indépendants dans R^d.*

Un 1-simplexe est un segment qui a deux 0-faces : les 2 sommets extrémités. Un 2-simplexe est un triangle qui a trois 1-faces : ses 3 arètes de côtés. Un 3-simplexe est un tétraèdre qui a quatre 2-faces : 4 triangles.

Dans ce chapitre nous traitons principalement les 2-triangulations.

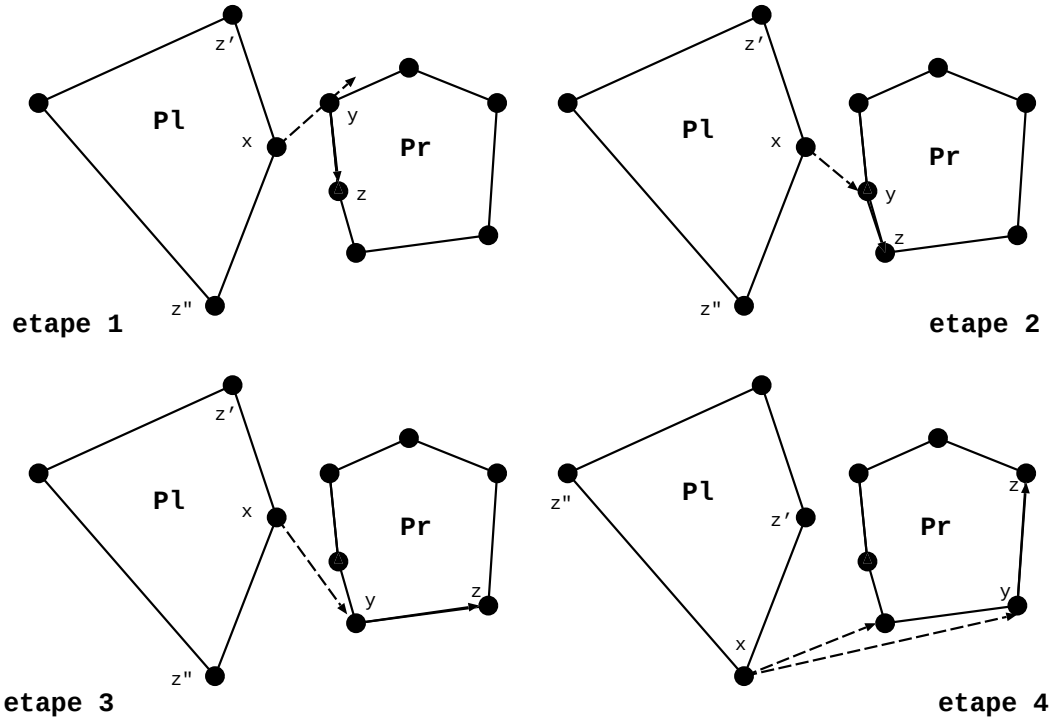
Théorème 6 *Le nombre de triangles (t) d'une 2-triangulation est donné par la formule : t = 2n -2 -a' où t est le nombre de triangles, n le nombre de sommets et a' le nombre d'arètes sur la frontière de la triangulation.*

Exercice 21 *Démontrez le théorème précédent. Idée : utilisez la relation d'Euler reliant les cardinaux des k-faces d'un polytope et la propriété des triangles : a'+ 2a"= 3t où a' est le nombre d'arètes sur la frontière et a" le nombre d'arètes intérieures.*

Cette propriété n'est plus vraie pour une 3-triangulation qui peut avoir jusqu'à n^2 tétraèdres. Un exemple est présenté sur la figure 2.10.

Une 2-triangulation peut, dans R^3, représenter la frontière d'un solide où les faces sont toutes des polygones simples à 3 côtés.

Procedure Fusion_Convexe(T_R, T_L, T **)**
TB = Tangente basse pour T_R et T_L
TH = Tangente haute pour T_R et T_L
$T = T_R \cup T_L$
FinProcedure




```
TANGENTE_BASSE_CONVEXE(DEBUT_R,DEBUT_L,SUIV,PREC)
/*************************************************************/
/* DEBUT_L : le point le plus a droite du convexe gauche     */
/* DEBUT_R : le point le plus a gauche du convexe droit      */
/* PrV     : calcule le produit vectoriel                    */
/*************************************************************/
{ x = DEBUT_L; y = DEBUT_R; trouve = FAUX;
  z = SUIV[y]; z'' = PREC[x];
  while( trouve == FAUX ){ /* z a droite de xy */
    if( PrV((x,y),(y,z)) < 0 ){
      /* on descend z et y sur PD */
      y = z; z = SUIV[z];
    } else {  /* z'' a droite de xy */
      if( PrV((x,y),(y,z'')) < 0 ){
        /* on descend z'' et x sur PG */
        x = z''; z'' = PREC[z''];
      } else { trouve = VRAI; }
  }  return(x,y); }
```

FIG. 2.9 – Fusion d'enveloppe convexe

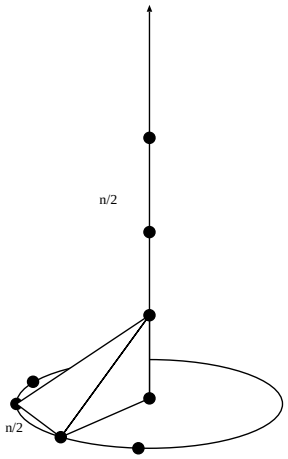


FIG. 2.10 – n^2 tétraèdres s'appuyant sur n points
L'exemple est constitué de $n/2$ points répartit sur un cercle tandis que les $n/2$ autres sont alignés sur l'axe perpendiculaire passant par son centre.

Exercice 22 *Soit une 2-triangulation dans l'espace (R^3) représentant la frontière d'un solide (ie. les sommets et les triangles sont sur la frontière du solide). Combien de triangles contient la triangulation si le solide est homéomorphe à une sphère ? à un tore ?*

Structure de données
La représentation informatique d'une triangulation peut être très simple : 3 tableaux sont suffisants. Ils sont illustrés sur la figure 2.11.

Exercice 23 *Ecrivez la fonction "TriangleVoisin" sur la structure de données de la figure 2.11. Utilisez cette fonction pour écrire l'algorithme de parcours des triangles partageant un sommet. Quelle est la complexité de l'algorithme ?*

Une 2-triangulation peut aussi être considéré comme une partition du plan (voir [1]) : une face "imaginaire" infinie couvrant le complément de la triangulation dans le plan. Afin de ne manipuler que des polygones simples à 3 côtés cette face est découpée en triangles "virtuels" s'appuyant sur un point abstrait (sans géométrie) que l'on peut imaginer à l'infini.

Exercice 24 *Modifiez la structure présentée figure 2.11 pour intégrer cette extension. Ecrivez l'algorithme qui parcours (dans le sens trigonométrique) tous les sommets de la frontière de la triangulation.*

2.4.2 Algorithme incrémental

Exercice 25 *Trouvez un algorithme incrémental permettant de construire une 2-triangulation (dans le plan) en $O(nlog_2n)$. Idée : voir figure 2.12 ou s'inspirer de la solution du même exercice pour le calcul de l'enveloppe convexe.*

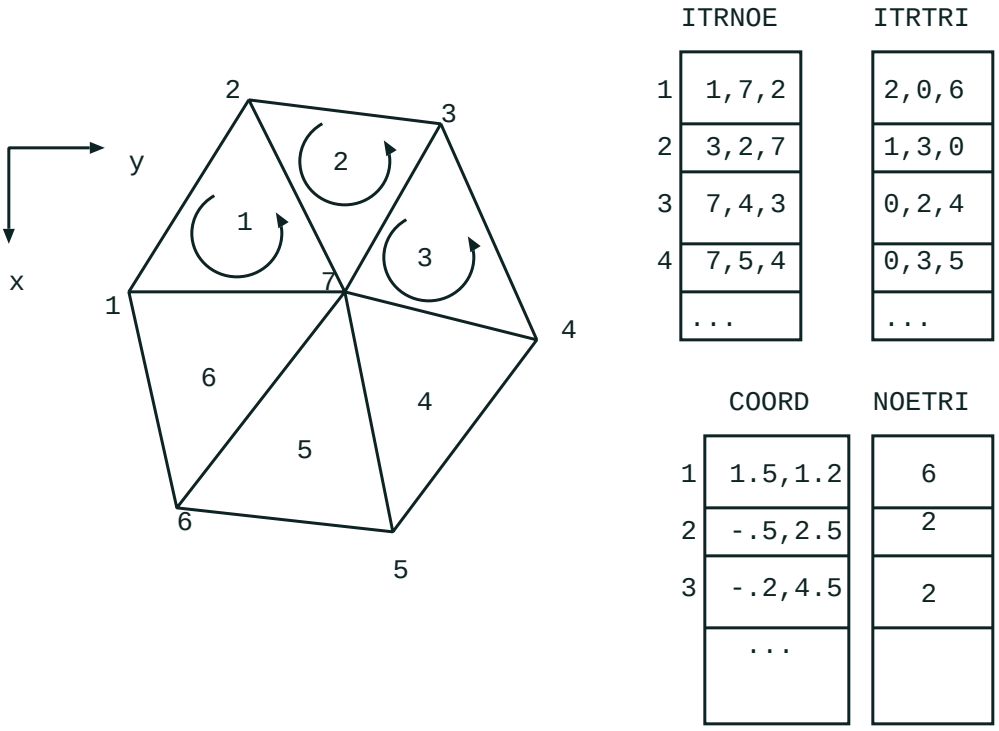


FIG. 2.11 – Structure informatique

La référence d'un triangle est un numéro, sa définition est donnée dans un tableau noté ITRNOE. ITRNOE est suffisant pour définir le maillage mais pour vérifier la définition et pour optimiser de nombreux traitements, on construit aussi le tableau des voisinages entre les mailles que l'on note ITRTRI. Un troisième tableau est construit pour établir un lien entre les noeuds et les triangles : NOETRI. Ce tableau n'est pas toujours nécéssaire et peut être économisé pour certains traitements. Dans l'exemple le triangle à l'opposé du 3^{eme} sommet du triangle 1 (c'est à dire du sommet 2=ITRNOE(1,3)) est le triangle 6 (ITRTRI(1,3)=6). Pour le triangle 1 l'arête est parcourue dans le sens (7,2), pour le triangle 2 elle est parcourue dans le sens inverse (2,7). L'orientation cohérente des triangles est nécessaire : pour parcourir une triangulation, pour faire des calculs géométriques... Elle peut être totalement arbitraire ou s'appuyer sur un critère géométrique (l'ordre qui donne un produit mixte positif par exemple).

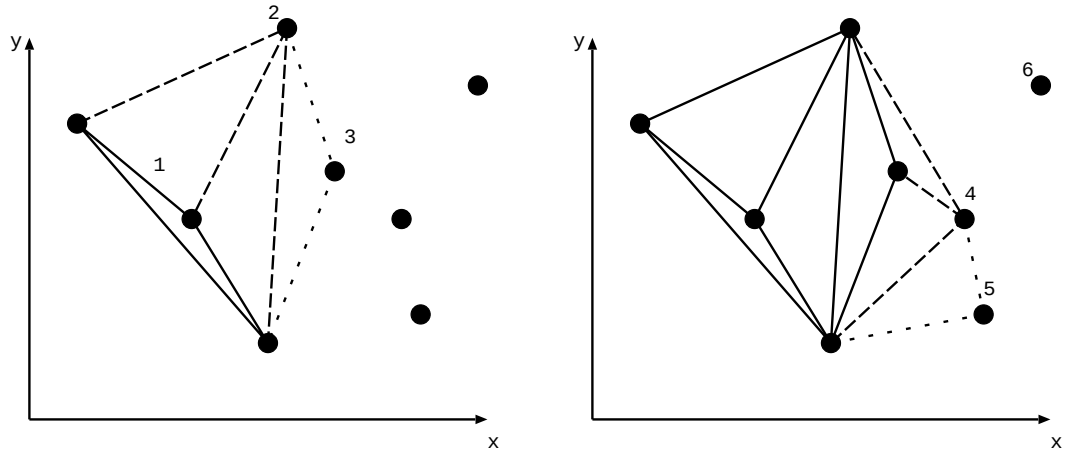


FIG. 2.12 – Triangulation incrémentale

2.4.3 "Qualité" d'une triangulation

Revenons au problème de triangulation d'un nuage de points. Deux questions viennent naturellement :

- Combien existe t-il de triangulations d'un nuage de n points ?
- Quels critères utiliser pour "choisir" la "meilleure" d'entre elles ?

Admettons que le nombre de triangulations est le plus élevé quand les points sont répartis sur une enveloppe convexe.

Théorème 7 *Soit N_m le nombre de triangulations d'un polygone convexe de $n = m + 2$ cotés. Il est donné par le nombre de Catalan [7] :*
par convention $N_0 = 1$
et pour $m \geq 1$

$$N_m = \sum_{i=1}^{m} (N_{i-1} * N_{m-i}) \tag{2.2}$$

ou encore :

$$N_m = \frac{2 * (2 * m - 1)}{m + 1} * N_{m-1} \tag{2.3}$$

Soit, pour quelques valeurs de n, N_m le nombre de triangulations :

n	3	4	5	6	7	8	9	10	11	12	13	14
N_n	1	2	5	14	42	132	429	1430	4862	16796	58786	208012

Exercice 26 *Démontrez le théorème précédent.*

Le critère d'évaluation de la qualité d'une triangulation dépend du type d'application pour laquelle elle est utilisée. Ceci dit pour des applications très

générales comme l'interpolation nodale, le calcul par éléments finis ou l'affichage de surfaces... On préfère souvent des triangles "réguliers" c'est à dire le moins allongé et le moins applati possible.

2.5 La triangulation de Delaunay

2.5.1 Les propriétés

C'est dans les années 1934 qu'un mathématicien soviétique B.Delaunay découvre les propriétés des triangulations satisfaisant "le critère de la sphère vide". Depuis les années 80 ces triangulations dites "de Delaunay" sont utilisées dans de nombreux algorithmes pour leurs propriétés géométriques.

Définition 11 *Soit N un ensemble de noeuds répartis dans un espace de dimension d. La triangulation T de l'enveloppe convexe de N s'appuyant sur tous les noeuds de N, est* **une triangulation de Delaunay** *si et seulement si :*

$$\forall t \in T, \forall n \in N, n \notin B(t) \tag{2.4}$$

où B(t) est la boule ouverte circonscrite au triangle t.

Dans le plan (pour d=2) il existe 2 triangulations possibles s'appuyant sur 4 noeuds : l'une est de Delaunay, l'autre pas (voir figure 2.13).

Définition 12 *Soit T une triangulation dans le plan. A chaque triangle de T est associée la valeur de son angle minimum. Soit V la matrice colonne dont les termes sont les angles minimums de T triés dans l'ordre croissant. Ce vecteur est appelé* **la "finesse" de la triangulation**.

Théorème 8 *Dans le plan une triangulation qui respecte le critère de Delaunay maximise la finesse pour l'ensemble de noeuds donnés.*

Autrement dit **Delaunay est un critère de connexion des noeuds qui minimise les triangles applatis** ($(a'_1 < a_1)$ figure 2.13).

Démonstration 2 *voir entre autre la publication de [16] et l'ouvrage [6].*

Une triangulation de Delaunay d'un nuage de point de R^d peut être construite en $O(nlog(n) + n^{(d/2)})$ opérations dans le pire cas [6] et [16] pour le cas d=2.

Remarque : en 3D de nombreuses propositions deviennent fausses. Citons par exemple : que l'existence d'une tétraédrisation d'un polyedre n'est pas toujours vérifiée, que le cardinal d'une tétraédrisation ne dépend plus linéairement du nombre de noeuds (de l'ordre de n^2 dans le pire cas), que l'angle minimum

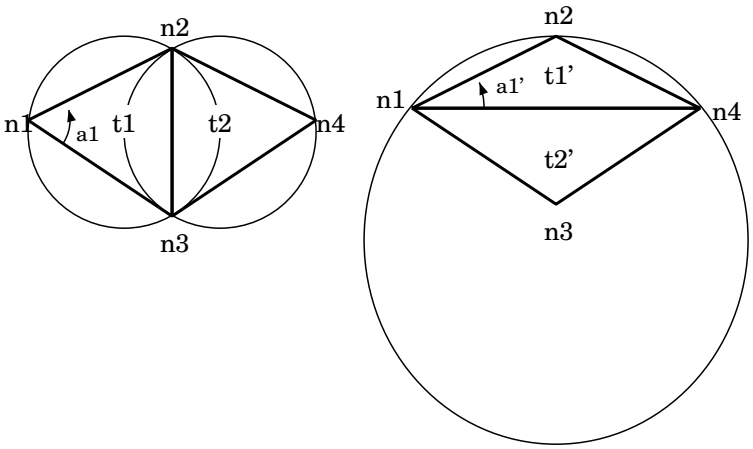


FIG. 2.13 – La triangulation de Delaunay
A gauche la triangulation respecte le critère de Delaunay. A droite elle ne le respecte pas : $n_3 \subset B(t'_1)$. Notons que cette propriété est reflexive : $n_3 \subset B(t'_1) \Leftrightarrow n_2 \subset B(t'_2)$. Notons aussi que l'on peut raisonner sur les arètes de la triangulation : une arête est de Delaunay ssi les sommets opposés ne sont pas à l'interieur des boules circonscrites des triangles incidents. Sur la figure de gauche : n_1 et n_4 sont les sommets opposés à l'arête (n_3, n_2) qui est de Delaunay puisque $n_4 \not\subset B(t_1)$ et $n_1 \not\subset B(t_2)$.

n'est plus maximisé par la propriété de Delaunay...

En 2D plusieurs algorithmes peuvent être mis en oeuvre, entre autre des algorithmes optimaux, en complexité pire cas $O(nlog(n))$, fondés sur le principe de diviser pour régner [16]. L'algorithme qui est présenté ici est utilisé pour le maillage adaptatif : il est incrémental et n'est pas optimal.

2.5.2 Algorithme incrémental

La méthode suit le schéma des algorithmes incrémentaux : on calcule une triangulation de Delaunay contenant 3 puis 4,5... $Card(N)$ noeuds. A l'itération i un noeud est ajouté à la triangulation de Delaunay des $(i-1)$ noeuds précédents. La triangulation est mise à jour pour satisfaire la propriété de Delaunay. On trouve plusieurs techniques dans la littérature :

Destruction/création : les triangles ne satisfaisant pas le critère de Delaunay sont détruits, des triangles satisfaisant le critère sont construits à la place. La structure d'un algorithme de ce type est donnée figure 2.14, il est illustré figure 2.15.

Inversion de diagonales : le triangle (ou les 2 triangles) contenant le point p est découpé en 3 (resp. 4). Si un triangle adjacents (aux triangles s'appuyant sur p) ne satisfait pas le critère de Delaunay il y a inversion de l'arète commune. Deux nouveaux trian-gles sont adjacents aux triangles contenant p, il faut les tester...L'algorithme s'arrête quand tous les triangles sont de Delaunay. On trouve cet algorithme décrit page 190 dans [5].

Théorème 9 *L'algorithme incrémental présenté ci-dessus construit une triangulation de Delaunay. On considère que les calculs sont exacts (à précision infinie).*

Exercice 27 *Démontrez le théorème précédent.*

Complexité :

L'algorithme de la figure 2.14 est en $O(n^2)$, où n est le nombre de noeuds. Même si la phase de recherche est améliorée, l'algorithme reste en $O(n^2)$ pire-cas à cause de l'étape de destruction. Un exemple de pire-cas est présenté figure 2.16

Fiabilité - Robustesse :

L'algorithme est sensible aux erreurs de précision numérique. Il repose sur le fait qu'à chaque étape -avant et après l'ajout d'un noeud- la triangulation est a la propriété de Delaunay. En fait, le test de la boule vide est effectué à une précision donnée et lorsque plusieurs points sont "presque" cocycliques, la décision peut être fausse (voir la figure 2.17)

Le problème se rencontre rarement en 2D mais est incontournable en 3D. Plusieurs solutions peuvent être mises en oeuvre :

Procedure Trianguler(N, T)
/***** Initialisation *****/
t_0 = Creer_Triangle_Englobant(N)
S_0 = Sommets(t_0)
T = { t_0 }
/***** Boucle sur les noeuds *****/
Pour tous les noeuds n de N
    Ajouter_Noeud(n, T)
FinPour
Pour tous les triangles t incidents a un sommet de S_0
    T = T - { t }
FinPour
FinProcedure

Procedure Ajouter_Noeud(n, T)
TD = $\emptyset$
/***** Recherche des elements non-Delaunay *****/
Pour tous les triangles t de T
    Si (n $\in$ B(t)) Alors TD = TD $\cup$ { t }
FinPour
/***** Destruction des elements *****/
Pour tous les triangles t de TD
    T = T - { t }
FinPour
/***** Construction des nouveaux elements *****/
A = Frontiere(TD)
Pour toutes les aretes (n_i,n_j) de A
    t = Creer_Triangle(n_i, n_j, n)
    T = T $\cup$ { t }
FinPour
FinProcedure

FIG. 2.14 – Algorithme incrémental de Delaunay

La création du triangle englobant permet de limiter le nombre de cas à traiter à l'insertion de chaque noeud. En effet ces derniers sont toujours à l'intérieur de la triangulation.

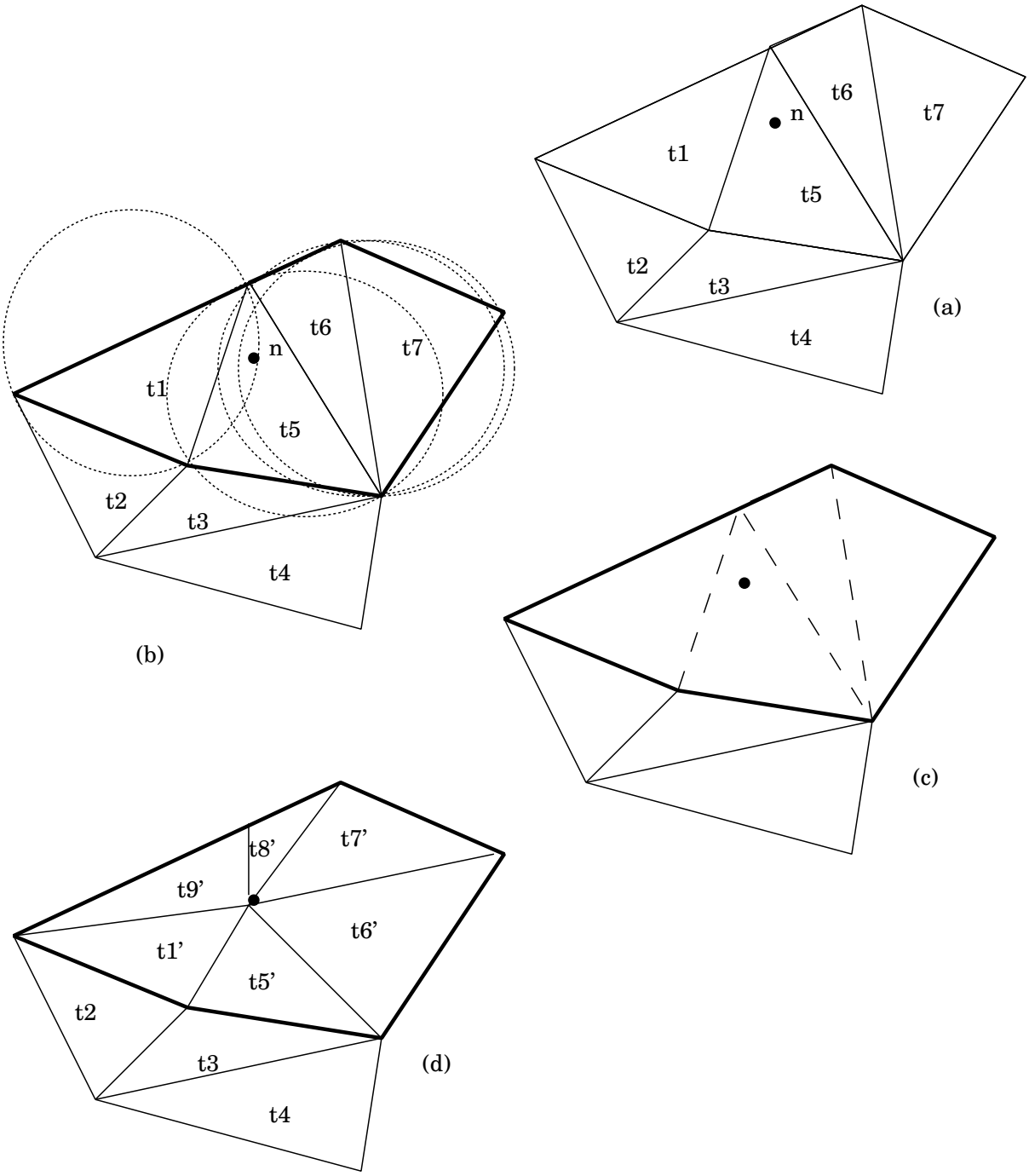


FIG. 2.15 – L'ajout d'un noeud dans une triangulation de Delaunay
(a) Le noeud n est ajouté à la triangulation de Delaunay T={t1,t2...,t7}.
(b) Les triangles t1,t5,t6 et t7 ne sont plus de Delaunay vis-à-vis de n.
(c) Ils sont détruits.
(d) De nouveaux triangles {t1',t5',t6',t7',t8',t9'} sont construits. Ils s'appuient sur le noeud n et la frontière de l'union des triangles détruits.

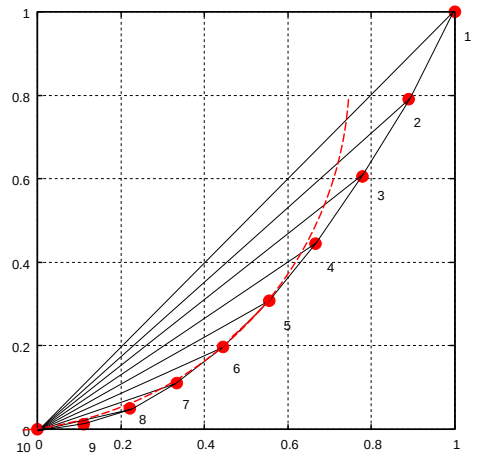


FIG. 2.16 – Cas particulier pour l'algorithme incrémental
Les points d'une parabole constituent un pire-cas pour un balayage en x décroissant : $O(n^2)$ (ordre donné par les numéros sur la figure). C'est en revanche le meilleur-cas pour un balayage en x croissant $O(n)$. On peut vérifier sur la figure que le triangle 0,6,5 satisfait Delaunay car son cercle circonscrit ne contient aucun autre point.

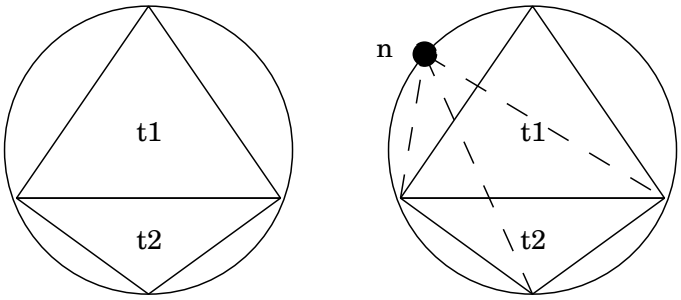


FIG. 2.17 – Erreur numérique
A l'ajout du point n faisons l'hypothèse que le test numérique donne : $n \in B(t_2)$ et $n \notin B(t_1)$. L'algorithme engendre alors des triangles intersectants. Notons que si les triangles sont orientés ; l'erreur est détecté à la création d'un triangle plat ou retourné (déterminant nul ou négatif).

Calculs exacts : les calculs sont réalisés sur des réels avec une précision adaptative jusqu'à un résultat exact [1]. Une autre façon consiste à passer toutes les données en entier, les calculs sont alors exacts.

Détection et euristiques : c'est la technique la plus simple à mettre en oeuvre mais la moins efficace. Il suffit de détecter les triangles non-valides avant de les créer, et de remettre l'insertion du point à une itération ultérieure (si c'est possible).

2.5.3 L'algorithme diviser pour régner

C'est un algorithme optimum en $O(nlog_2 n)$ ou n est le nombre de points. Le principe est le même quelque soit la dimension. Nous l'exposons dans le cas 2D (pour plus de détails consulter [16]). Soit N un ensemble de points répartis dans R^2. L'algorithme suit le schéma classique de l'approche Diviser pour Règner :

1. Trier N dans l'ordre lexicographique
2. Diviser récursivement N en 2 sous-ensembles N_L et N_R. N_L contient les points qui sont à gauche de la verticale, N_R contient les points qui sont à droite de la verticale. La recursion s'arrête quand chaque ensemble ne contient plus que 1,2 ou 3 noeuds.
3. Calculer la triangulation de chaque sous-ensemble N_i puis fusionner les triangulations 2 à 2.

La fusion des triangulations est réalisée en 2 étapes. La première fait appel á la fusion des enveloppes convexes : $CH(N_L \cup N_R)$ est obtenue linéairement à partir de $CH(N_L)$ et $CH(N_R)$. La seconde met á jour les triangles pour qu'ils respectent le critère de Delaunay. L'algorithme est présenté figure 2.18 et illustré sur un exemple figure 2.19.

Procedure Fusion_Triangulation(T_R, T_L, T **)**
TB = Tangente basse pour T_R et T_L
TH = Tangente haute pour T_R et T_L
$T = T_R \cup T_L$
$LR = TB$
Tantque ($LR \neq TH$) faire
1. $T = T$ + Inserer l'arête (L, R)
2. D etruire les arêtes (R, R_i) incidentes a R non-Delaunay.
3. D etruire les arêtes (L, L_j) incidentes a L non-Delaunay.
4. LR = Choisir l'arête de Delaunay $(L_j, R) ou (L, R_i)$

FinTantque $T = T$ + Inserer l'arête (L, R)
FinProcedure

FIG. 2.18 – Algorithme Diviser pour règner de Lee (Delaunay)

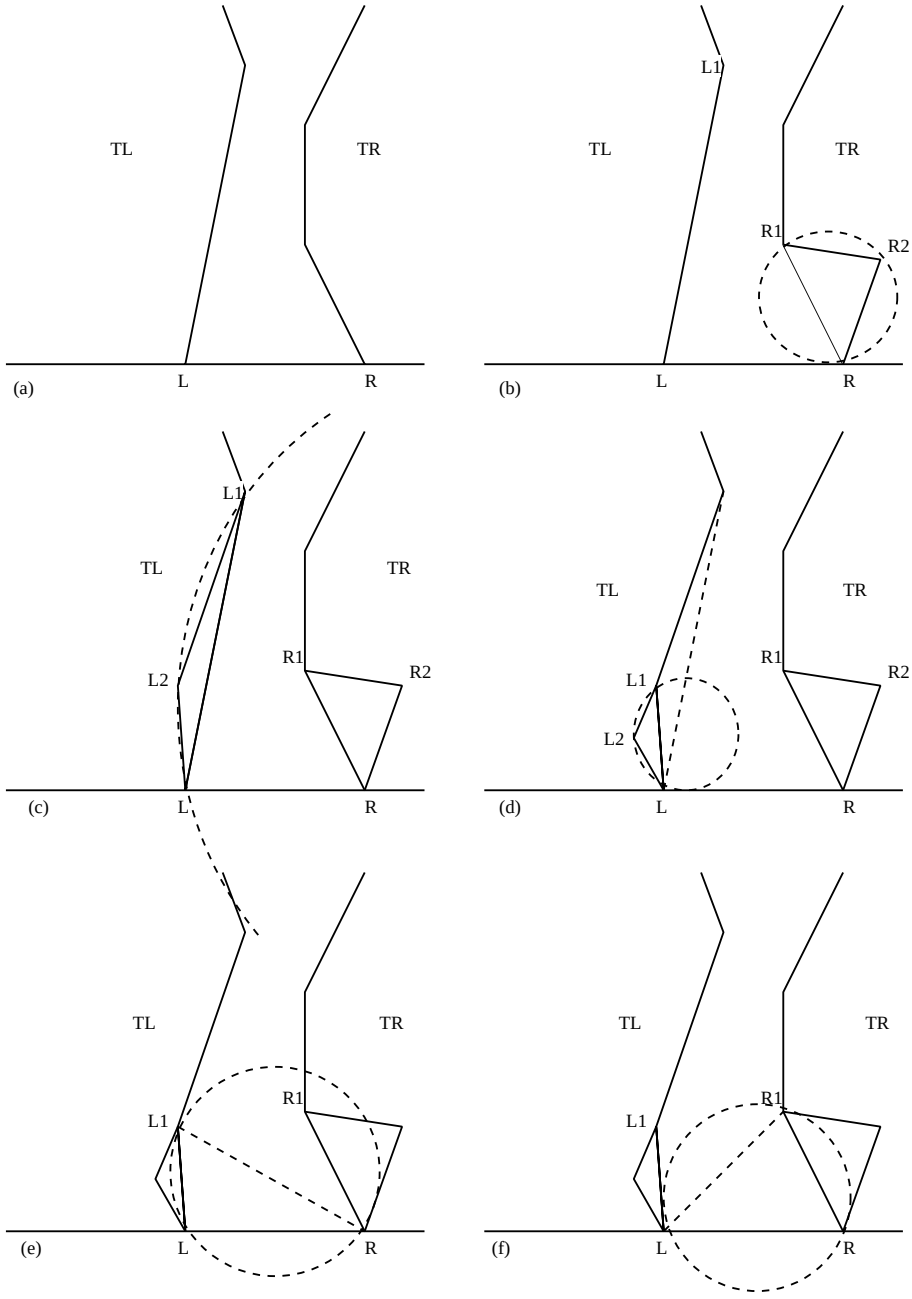


FIG. 2.19 – Étapes de l'algorithme Diviser pour règner
(a) Insertion de la tangente basse (L, R). (b) Les triangles incidents à R qui ne satisfont pas Delaunay vis à vis du point L sont détruits. (c) et (d) Les triangles incidents à L qui ne satisfont pas Delaunay vis à vis du point R sont détruits. (e) et (f) Le triangle de Delaunay s'appuyant sur l'arète (L, R) est contruit (ici L, R, R_1) Le nouveau segment devient (L, R) puis on recommence à l'étape (b).

2.6 Les diagrammes de Voronoi

2.6.1 Définition et dualité

C'est en 1850 que J.P.G.L Dirichlet remarque le premier que l'on peut partitionner le plan en cellules convexes avec des critères de distance. En 1908, l'Ukrainien Voronoi développe le concept tandis que Delaunay en 1934 trouve la structure duale qui s'appelle aujourd'hui la triangulation de Delaunay (voir la figure 2.21). Dans la littérature sur le sujet, les diagrammes de Voronoï peuvent prendre les noms de : tesselations de Dirichlet, polygones de Thiessen ou encore graphe, décomposition ou partition de Voronoï. Les points ou noeuds que séparent le diagramme sont généralement appelé les sites.

Définition 13 *Soit N un ensemble de noeuds répartis dans un espace de dimension d.* **Le diagramme de Voronoi** *de N est l'ensemble des régions $v(n_i)$ des points les plus proches de n_i :*

$$V(N) = \{v(n_i), \forall x \in v(n_i), dist(x, n_i) < dist(x, n_j) \forall n_i, n_j \in N^2, i \neq j\} \quad (2.5)$$

Théorème 10 La triangulation de Delaunay est le dual du diagramme de Voronoi *(quelque soit la dimension d de l'espace). Pour $d = 2$ on peut facilement démontrer que :*
- un point de voronoi est le centre du cercle circonscrit à un triangle de Delaunay (et réciproquement)
- à chaque arête de voronoi correspond une arête de Delaunay perpendiculaire (et réciproquement : l'arête de Voronoi est la médiatrice de l'arête de Delaunay).

Exercice 28 *Démontrez le théorème précédent qui est illustré figure 2.21.*

2.6.2 L'algorithme incrémental

L'algorithme incrémental, comme son nom l'indique, ajoute les noeuds un par un au diagramme de Voronoï qu'il met à jour à chaque étape. Son principe est le suivant. Soit D_{i-1} le diagramme de Voronoï avant la prise en compte du noeud n_i. La première étape consiste à rechercher dans D_{i-1} la cellule c_j contenant n_i. En coupant cette cellule par la médiatrice entre n_i et n_j on obtient une partie de la future cellule c_i. On passe ensuite aux cellules voisines de c_j que l'on découpe de la même manière pour finalement assembler les morceaux et constituer c_i.

2.6.3 L'algorithme diviser pour régner

On trouvera un algorithme diviser pour régner 2D dans [27] page 213 à 220.

2.6.4 L'algorithme avec balayage

Cet algorithme, connu aussi sous le nom de son auteur : Steven Fortune (1987), construit des cellules de Voronoï précèdant une droite de balayage. Les principales étapes de l'algorithme sont présentées sur la figure 2.20

2.7 Généralisation de Voronoï et applications

Le graphe de Voronoï peut se généraliser simplement. Rappelons la définition.

Définition 14 *Soit N un ensemble de noeuds (de sites) répartis dans un espace de dimension d.* **Le diagramme de Voronoi** *de N est l'ensemble des régions* $v(n_i)$ *des points les plus proches de* n_i. *La région* $v(n_i)$ *s'écrit :*

$$v(n_i) = \{x \in R^d, dist(x, n_i) <= dist(x, n_j) \forall j\} \tag{2.6}$$

Si l'on considère maintenant que n_i peut lui même être un ensemble de points, on obtient :

$$v(n_i) = \{X \in R^d, \forall m_{ik} \in n_i, \forall m_{jk} \in n_j dist(X, m_{ik}) <= dist(X, m_{jk}) \forall j\} \tag{2.7}$$

La définition peut être généralisée de différentes façons :

la dimension de l'espace peut être quelconque ($d \in N$)

la distance peut être pondérée en chaque noeud ;

la distance peut être exprimée dans une métrique non-euclidienne, par exemple la métrique "L un" ou "L infinie" :
$L_1 = \Sigma_{i=1}^{d} |x_i - m_i|$
$L_\infty = max_{i=1,d} |x_i - m_i|$

les sites peuvent être des ensembles de points, on parle alors de graphe de Voronoï d'ordre k (avec $k = Card(n_i)$) ;

les sites peuvent être des variétés d'ordre supérieur à zero (le points), on parle alors de graphe de Voronoï d'arêtes (ordre 1)...

Pour certaines applications il est nécessaire de combiner ces extensions. Par exemple pour l'analyse de circuits intégrés VLSI (Very Large Scale Integration), la détection des zones critiques est obtenue par un graphe de Voronoï d'ordre 2 en métrique L_∞.
En reconnaissance des formes le graphe de Voronoï des arêtes pzermet de construire le squelette ou l'axe médian, le dilaté ou l'érodé d'un polygone. Certains de ces traitements sont réalisés sur des modèles d'énumération spatiale (des images généralement), ils sont alors assez différents de ceux présentés ici.

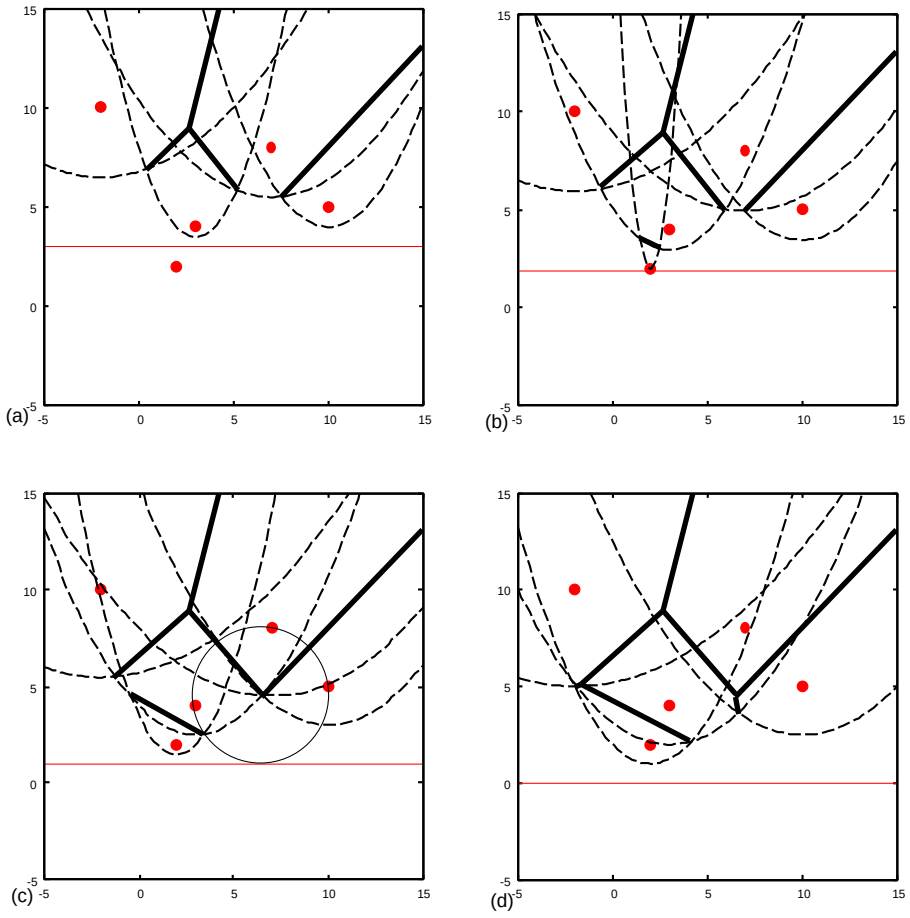


FIG. 2.20 – Voronoi avec balayage

Cet algorithme construit des cellules de Voronoï précèdant une droite de balayage. (a) Cette droite active des évenements de deux types : insertion d'un nouveau noeud, intersection de 2 arètes du graphe de Voronoï. Les cellules sont mises à jour derrière une ligne dite ligne de plage constituée d'arcs de parabole. (b) A l'activation d'un noeud l'arc de parabole associé est ajouté dans la ligne de plage. (c) A l'intersection de 2 arètes du graphe : un arc de parabole disparaît. L'évenement se produit quand la droite a totalement balayé le cercle circonscrit à 3 noeuds voisins. Notez que le centre du cercle correspond au point d'intersection.

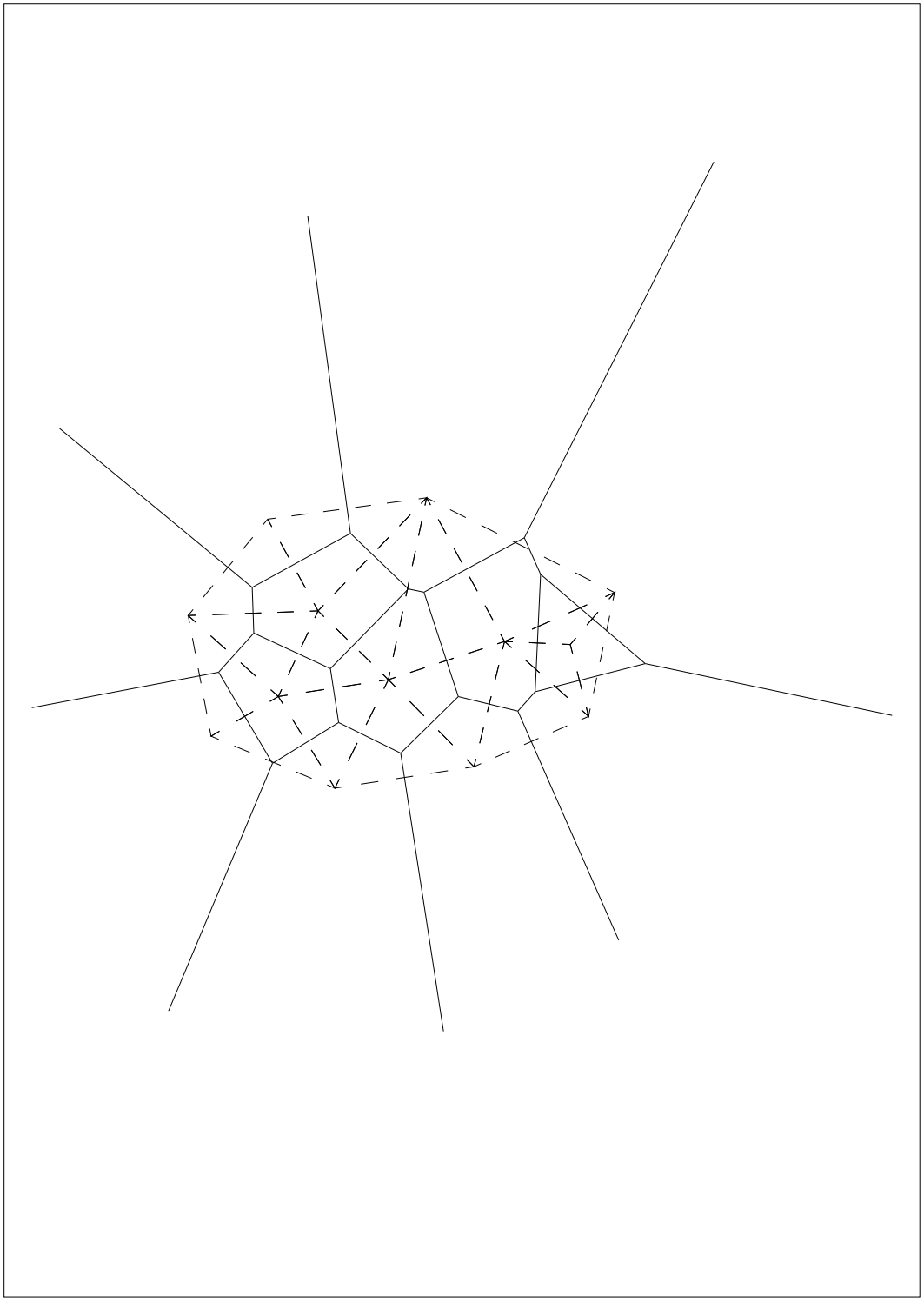


FIG. 2.21 – Dualité Delaunay - Voronoi

2.8 Les triangulations contraintes

On peut souhaiter imposer des contraintes aux triangulations. Par exemple de contenir des arêtes particulières (de la frontière d'un polygone si l'on veut le trianguler) des contraintes sur la taille et la forme des triangles si l'on souhaite faire des calculs par le méthode des éléments finis ou visualiser une surface discrétisée... Dans ce chapitre nous nous limiterons aux contraintes bidimensionnelles.

2.8.1 Le respect d'arête dans une triangulation

La triangulation de Delaunay s'appuyant sur l'ensemble des noeuds donnés forme un convexe. Si l'on souhaite par exemple extraire les triangles à l'intérieurs d'un polygone il est nécessaire de retrouver les arêtes de frontière du polygone dans la triangulation afin de distinguer les éléments extérieurs des éléments intérieurs.

Il y a un problème quand une triangulation ne respecte pas la frontière du domaine (i.e. un triangle intersecte une arête du bord comme l'illustre la figure 2.22). Ce problème se pose pour l'algorithme de triangulation exposée figure 2.14.

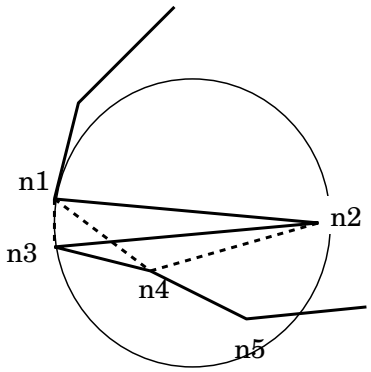


FIG. 2.22 – Le non respect de la frontière

Le respect du critère de Delaunay impose les triangles (n_1, n_3, n_4) et (n_1, n_4, n_2) provoque une intersection à la frontière en (n_3, n_2). On dira que l'arête (n_2, n_3) n'est pas "Delaunay admissible".

Plusieurs approches permettent de regler le problème de respect d'arête :

- Redécoupage des arêtes : modification de la frontière par ajout de noeud à posteriori ou à priori.
- Forçage des arêtes : modification de la connectique (à posteriori) sans ajout de noeud avec abandon du critère de Delaunay.

Ajout de noeuds

Une solution au problème consiste à générer un nouveau point sur l'arète (au milieu par exemple). L'ajout d'un seul point n'est pas toujours suffisant et le processus doit être répété jusqu'au respect effectif.
Si l'on traite le problème à posteriori les points ajoutés sont intégrés dans la triangulation au fur et à mesure jusqu'au respect.
Si l'on souhaite ajouter les noeuds a priori (avant triangulation) il faut que les arètes ainsi découpées deviennent "Delaunay admissible" (voir figure 2.23).

La méthode est simple mais elle a deux inconvénients :
- dans des cas particuliers on risque de générer beaucoup de points ;
- le maillage de la frontière est modifié et le raccord avec des maillages pré-existants n'est plus correct.

Si l'on souhaite ajouter les noeuds a priori (avant triangulation) il faut le faire afin de rendre la future frontière "Delaunay admissible".

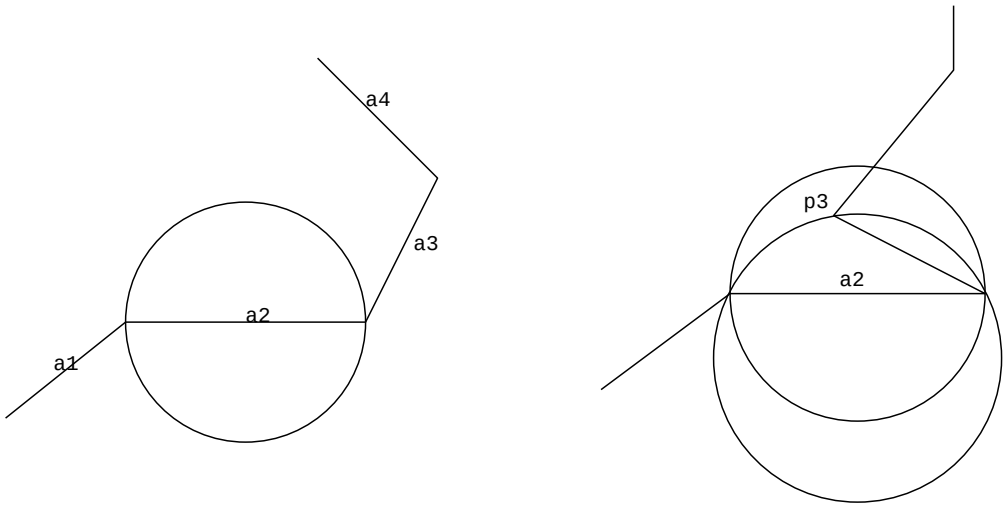


FIG. 2.23 – Détection des arètes de frontière "non Delaunay admissible"

Si le cercle de diamètre a_2 ne contient aucun point en son intérieur alors l'arète a_2 est "Delaunay admissible". Si le cercle contient un point (p_3) et que le cercle passant par les sommets de a_2, p_3 ne contient aucun point alors a_2 est "Delaunay admissible". Dans le cas contraire il faut redécouper l'arète a_2.

Forçage des arètes

Le traitement consiste à modifier une triangulation T=$\{(n_i,n_j,n_k), (n_i,n_j,n_k)\in N^3\}$ s'appuyant sur un ensemble de points N=$\{n_i=(x_i,y_i), (x_i,y_i) \in R^2\}$ afin qu'elle respecte un ensemble d'arêtes données A=$\{(n_i,n_j), (n_i,n_j)\in N^2\}$. On dira que l'on force les arêtes (n_i,n_j).

Le principe de forcage des arètes est le suivant. On considère que l'arête à

respecter (n_i,n_j) se trouve à l'intérieur de la triangulation T : c'est vrai dans le contexte de l'algorithme de la figure 2.14 puisque T forme un convexe. Soit T' $=\{t, t \in T, t \cap]n_i, n_j[\neq \emptyset\}$.

Deux approche sont possible ([24] :

Première approche : T' forme un polygone simple que l'arête (n_i, n_j) décompose en 2 polygones simples (cf. figure 2.24).

Le problème du respect d'arête est ramené au problème de triangulation d'un polygone simple non forcément convexe.

La méthode est simple mais elle a deux inconvénients :

- on peut perdre des points lors de la fusion des triangles de T'.
- la convergence n'est pas assurée (on retrouve le problème de respect d'arêtes).

Deuxième approche : Soit A' l'ensemble des arêtes de la triangulation intersectant le segment $]n_i, n_j[$. La deuxième approche consiste à inverser les arêtes de A' afin de diminuer progressivement leur nombre (c.a.d. qu'après inversion elle n'intersecte plus le segment). On peut montrer que le traitement est convergeant même si ce n'est pas au sens strict. Une technique d'inversions "randomisées" peut être utilisée (voir chap. 4 de [6]).

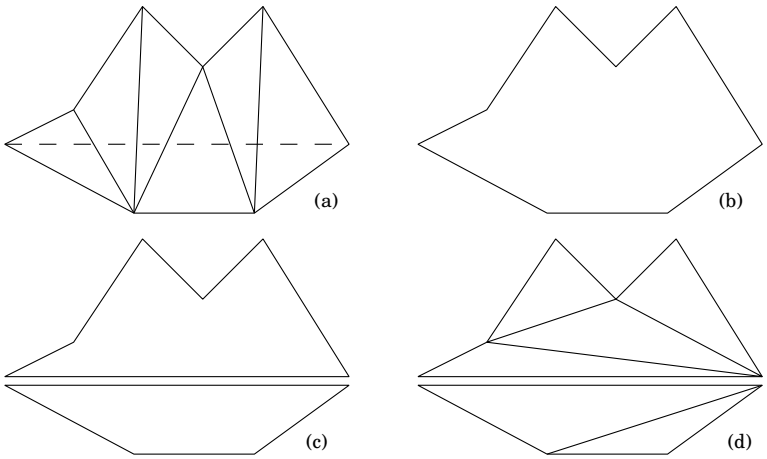


FIG. 2.24 – Une triangulation d'un polygone
(a) D'etection des triangles intersectants l'arête (pointillés). (b) Fusion des triangles. (c) Séparation des 2 polygones. (d) Triangulation des polygones.

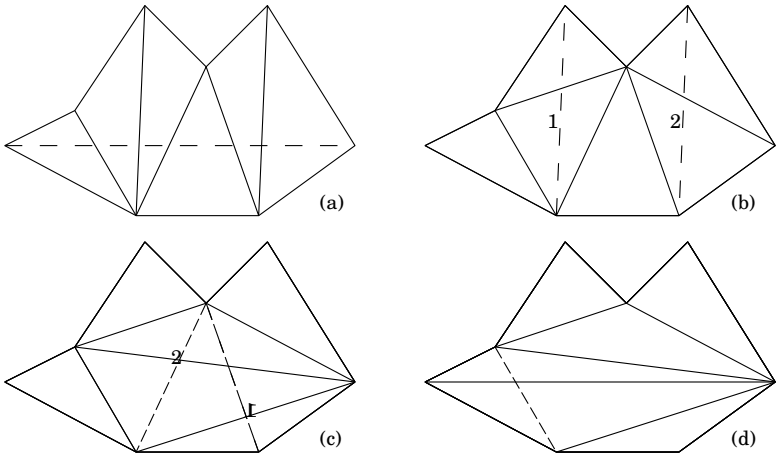


FIG. 2.25 – Inversion des arêtes de la triangulation
(a) Détection des arêtes intersectants l'arête imposée (pointillés). Card(A')= 5. (b) Card(A')=3. (c) Card(A')=1. (d) Card(A')=0.

2.9 Taille et forme des triangles imposées

L'objectif est ici d'obtenir une triangulation dont les triangles ont une taille et une forme imposées.

De nombreuses méthodes permettent de satisfaire des critères de taille tout en ne dégradant pas la forme des éléments :
- des techniques de bissection [26] et de découpage en motifs [4] ;
- des techniques de génération de noeuds sur les arêtes ;
-...
Mais l'approche exposée ci-dessous peut être appliquée aux 2 types de critères démonstration à l'appui [14] [29] [22]. Elle a de plus l'avantage d'utiliser le même algorithme d'insertion de points que celui présenté précédemment (voir figure 2.14).

2.9.1 Un algorithme de raffinement

Principe
Le principe du raffinement est le suivant :

si $T_r(R) > T_s(R) \Rightarrow CN(R)$

avec R : Une région de l'espace
T_r : La taille ou la déformation réelle des éléments dans la région R
T_s : La taille ou la déformation souhaitée des éléments dans la région R
CN : Ajout de noeuds dans R

Plusieurs choix sont possibles pour la définition de $R, T_s(R), T_r(R)$ et $CN(R)$. Nous avons choisi de prendre :

- R : La boule circonscrite à une maille
- $T_r(R)$: Le rayon de la boule
- $T_s(R)$: Une fonction qui est évaluée au centre de la boule
- $CN(R)$: Une procédure qui ajoute un noeud au centre de la boule

Cette définition est valable en 3D, 2D et en 1D. Ce choix pour R et pour CN(R) n'est pas nouveau [29][33]. La génération des points peut poser des problèmes si le centre de la boule est hors de T. Différentes solutions ont été proposées dans la littérature comme le rejet du point ou son déplacement.

Algorithme

L'algorithme de raffinement est le suivant : Soit T une triangulation de Delaunay d'un ensemble de points en 2D. Soit R_{cm} le rayon maximum de tous les cercles circonscrits des triangles de Delaunay de T. Soit T_s la taille souhaitée. Si $R_{cm} \leq 1.5 * T_s$ l'algorithme ne fait rien, si $R_{cm} > 1.5 * T_s$ un nouveau point est calculé au centre du cercle et s' il est à l'intérieur d'un triangle de T, il est ajouté à T (avec une variante de l'algorithme de la figure 2.14) et donne une nouvelle triangulation de Delaunay notée T'.

L'algorithme s'appuit sur la propriété des triangulations de Delaunay suivante :

Théorème 11 *Soit N un ensemble de points, le plus grand cercle vide inclus dans l'enveloppe convexe de N est un cercle circonscrit à un triangle de la triangulation de Delaunay de N. (Voir théorème 6.15 [27]).*

Exercice 29 *Faire la preuve de l'algorithme.*

2.9.2 Les fonctions "taille par défaut"

L'algorithme précédent fonctionne pour n'importe quelle fonction T_s donnée dans R^2. Quand T_s n'est pas donnée explicitement mais qu'il faut la construire à partir d'un maillage initial ou du maillage de la frontière, plusieurs possibilités s'offrent à nous.

On considère que :

- les arêtes du maillage initial donnent localement la taille souhaitée ;
- l'on veut des éléments de bonne qualité (réguliers) ;
- l'on veut le moins d'éléments possible.

Une fonction locale, prenant en argument un triangle permet de répondre simultanément à ces 3 exigences :

$$T_s(t) = \alpha * l(t) \tag{2.8}$$

où l(t) est la longueur de la plus petite arête du triangle t
et α un coefficient réel .

Cette définition est valable en 2D et en 3D.

Théorème 12 *L'algorithme converge si $\alpha = 2/3$. C'est à dire si l'on ne génère pas d'arête plus petites que celle données au départ.*

Théorème 13 *Si l'algorithme ne rejette aucun point, la triangulation résultante satisfait :*

$$\forall t \in T, \frac{l(t)}{\sqrt{3}} \leq R_c(t) \leq l(t) \tag{2.9}$$

Démonstration 3 *Pour la démonstration voir [22].*

Qualité :

La forme d'un triangle est évaluée à l'aide du critère suivant :

$$Q(t) = \beta * R_i(t)/L(t) \tag{2.10}$$

où $R_i(t)$ est le rayon du cercle inscrit à t
L(t) la longueur de l'arête la plus grande de t
et β un coefficient normalisateur ($\beta = \sqrt{12}$).

Théorème 14 *Si l'algorithme ne rejette aucun point, il donne une triangulation T dont les triangles ont une qualité bornée :*
$\forall t \in T, \frac{R_i(t)}{L(t)} > (1 - \sqrt{3}/2)$ *ou encore* $\forall t \in T, Q(t) > 2 * \sqrt{3} - 3 \approx 0,464$

Démonstration 4 *Pour la démonstration voir [22].*

Un autre algorithme :

On trouve dans [14] un autre algorithme qui suit à peu près le même schéma mais qui fonctionne sur les angles :

- R : Le triangle
- $T_r(R)$: L'angle minimum du triangle
- $T_s(R)$: Un angle imposé
- $CN(R)$: La fonction qui calcule un noeud au centre de la boule circonscrite au triangle.

Cet algorithme raffine aussi la frontière quand c'est nécessaire et traite des géométries polygonales presques quelconques. Il part du principe :

Théorème 15 *Si t est un triangle d'angle minimum θ, le triangle s'appuyant sur le centre du cercle circonscrit a un angle minimum de $2 * \theta$ (Voir figure 2.27).*

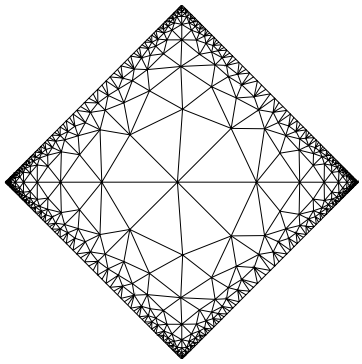


FIG. 2.26 – Un exemple de raffinement (Delos.1.0.0)
On remarquera sur cet exemple de maillage que l'on retrouve à peu de choses près la symétrie du maillage linéique. Le maillage linéique est constitué de 338 arêtes faisant un rapport maximum de 1.1. La triangulation résultante est composée de 1046 triangles et de 718 noeuds. $MIN(Q(t)) = 0.34, MIN(l(t)/L(t)) = 0.34, MIN(\theta(t)) = 19.5$.

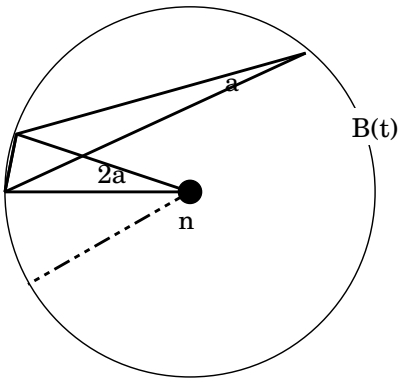


FIG. 2.27 – L'évolution de l'angle minimum
Si t est un triangle d'angle minimum θ, le triangle s'appuyant sur le centre du cercle circonscrit a un angle minimum de $2 * \theta$.

Bibliographie

[1] CGAL 2.4. *Computational Geometry Algorithms Library*, May 2002. http :www.cgal.org.

[2] Yerry M. A. and Shephard M. S. Automatic three-dimensionnal mesh generation by modified-octree technique. *Int. Journ. numerical methods in engineering*, 20 :1965–1990, 1984.

[3] P. Baehmann, S. Wittchen, M. Shephard, K. Grice, and M.A. Yerry. Robust, geometrically based, automatic 2d mesh generation. *Int. Journ. numerical methods in engineering*, 24 :1043–1078, 1987.

[4] R.E. Bank and A.H. Sherman. The use of adaptative grid refinement for badly behaved elliptic partial differential equations. *Math. Comput.*, 22 :18–24, 1980.

[5] M.de Berg, M. van Kreveld, M. Overmars, and O.Schwarzkopf. *Computational Geometry, Algorithms and Applications*. Springer, 1997.

[6] J.D Boissonnat and M. Yvinec. *Géométrie Algorithmique*. Ediscience International, 1995.

[7] A. Bouvier and M. George. *Dictionnaire des Mathématiques*. Presse Universitaire de France, 1979.

[8] R. Chacar. *Opérateurs d'Euler et Modélisation Géométrique*. PhD thesis, Univ. Paris VI, November 1989.

[9] J.M. Chassery and A. Montanvert. *Géométrie Discrète en analyse d'images - Traité des Nouvelles Technologies*. Hermes, 1991.

[10] M. Cocheril. Opérations booléennes sur les polyèdres à faces courbes. Technical report, Ecole Nationale Supérieure des Mines de Paris, 1993. Stage DEA MISI.

[11] M. Coster and J.L. Chermant. *Précis d'analyse d'images*. Presses du CNRS, 1989.

[12] Foley, van Dam, Feiner, and Hughes. *Computer Graphics : principles and practice*. Addison Wesley, 1995.

[13] M.C Gaudel, M. Sonia, and C. Froidevaux. *Recherche, Tri, Algorithme sur les Graphes Volume II*. Collection didactique INRIA, 1997.

[14] J.Ruppert. A delaunay refinement algorithm for quality 2-dimensional mesh generation. Technical report, Nasa Ames Research Center, February 1995. submission to J. of Algorithms.

[15] Yehuda E. Kalay. Determining the spatial containment of a point in general polyedra. *Computer Graphics and Image Processing*, 19 :303–334, 1982.

[16] D. T. Lee and B. J. Schachter. Two algorithms for constructing a delaunay triangulation. *Int. Journal of Computer and Information Sciences*, 9(3), 1980.

[17] P. Lienhardt. Subdivisions of n-dimensional spaces and n-dimensional generalized maps, 1989. Saarbrucken.

[18] M. Mantyla. Gwb : A solid modeler with euler operators. *I.E.E.E Computer Graphics and Applications*, 2 :17–31, 1982.

[19] M. Mortenson. *Geometric Modeling.* Wiley, 1985.

[20] V. Mounoury. *Maillage automatique en hexaèdres par analyse sémantique.* PhD thesis, Ecole Nationale Supérieure des Mines de Paris, December 1995.

[21] J.M. Oliva. *Reconstruction Tridimensionnelle d'Objets Complexes à l'aide de Diagrammes de Voronoi simplifiés.* PhD thesis, Ecole des Mines de Saint-Etienne, October 1995.

[22] O.Stab. *Delos 1.0.0 : notice de conception.* Ecole Nationale Supérieur des Mines de Paris, 1995.

[23] R. Perucchio and M. Saxena. Automatic mesh generation from solid models based on recursive spatial decompositions. *Int. Journ. numerical methods in engineering*, 28 :2469–2501, 1989.

[24] P.L.George and H. Borouchaki. *Triangulation de Delaunay et maillage : application aux éléments finis.* Hermes, 1997.

[25] A.A.G. Requicha and H.B. Voelcker. Solid modelling : A historical summary and contemporary assessement. *IEEE Comput. Graphics Applic.*, 2 :9–24, 1982.

[26] M.C. Rivara. Algorithms for refining triangular grids suitable for adaptative and multigrid technics. *Int. J. for Numerical Methods in Engineering*, 20 :745–756, 1984.

[27] M.I. Shamos and F.P. Preparata. *Computational Geometry, An introduction.* Springer-Verlag, 1988.

[28] O. Stab. *Maillage automatique tridimensionnel par opérations booléennes.* PhD thesis, Ecole Nationale Supérieure des Mines de Paris, December 1992.

[29] T.K.Dey, C.L.Baja, and K.Sugihara. On good triangulations in three dimensions, June 1991.

[30] A.-H. Wallace. *Introduction à la topologie algébrique.* Gauthier-Villars, 1957.

[31] Lorensen W.E. and Cline H.E. Marching cubes : A hight resolution 3d surface construction algorithm. *Comput. Graphics*, 21(3) :163–169, July 1987.

[32] K. Weiler. The radial edge structure : A topological representation for non-manifold geometric modeling, 1988. North-Holland.

[33] W.H.Frey. Selective refinement : A new strategy for automatic node placement in graded triangular meshes. *Int. Journal of Num. MEth. in Engineering*, 24 :2183–2200, 1987.