Rapport de Stage de D.E.A.
Pour l'obtention du
D.E.A. d'Intelligence Artificielle et Algorithmique
de l'Université de Caen

Homotopie, surfaces discrètes et squelettisation.

Sébastien FOUREY
Encadré par Rémy MALGOUYRES

Septembre 1997

"La théorie c'est quand on ne comprend rien,
la pratique c'est quand ça ne marche jamais.
Parfois, théorie et pratique se rencontrent:
Ca marche et on comprend tout..."

Anonyme.

Table des matières

Remerciements

Je tiens à remercier

– Mon maître de stage, Rémy Malgouyres, pour m'avoir encadré avec implication et enthousiasme. Pour son aide au peaufinage de ce rapport.

– Su Ruan pour son travail de séparation du L.C.R. sans lequel l'application trouvée serait orpheline.

– Serge Langlois pour m'avoir permis de travailler sur des images de cerveau qu'ils a eu tant de mal à acquérir au sein du service I.R.M. du C.H.U. de Caen.

– Marinette Revenu pour m'avoir accueilli au sein de son laboratoire.

– Donald Knuth, auteur de TEX et Leslie Lamport pour son outil LATEX avec lequel ce rapport a été réalisé.

– Alexandre Lenoir, dont l'outil *Surftool* a été utilisé de façon très intensive pour visualiser les images sur lesquelles j'ai travaillé. Le gain de temps apporté par ce "visualiseur" 3D a été précieux !

Introduction

La géométrie discrète est un domaine de recherche pour lequel l'intérêt actuel est grandissant. De nombreuses applications apparaissent là où l'on a l'habitude de travailler dans un espace continu. En effet, ce domaine, à cheval entre mathématiques et informatique, est très adapté au calcul sur machine puisqu'il y est fait essentiellement usage de l'arithmétique entière. Il offre alors de nombreux avantages tant du point de vue de la rapidité de calcul que de la clarté des algorithmes utilisés, formellement explicites.

Ce stage, effectué au sein du laboratoire du Groupe de Recherche en Informatique, Imagerie et Instrumentation de Caen (G.R.E.Y.C.) a permis d'étudier une application de ces théories de la géométrie discrète: la squelettisation. En s'appuyant sur un travail théorique commencé depuis longtemps par Gilles Bertrand [1] et Rémy Malgouyres [2] (qui a encadré ce stage), il a été possible de concevoir et d'implanter divers algorithmes de squelettisation. Ceux-ci ont pu être utilisés pratiquement avec la réalisation d'une application dans le domaine de l'imagerie médicale: le recalage d'Images par Résonance Magnétique (I.R.M.) pour laquelle la squelettisation intervient comme une étape intermédiaire.

Une première étape de ce stage a bien sûr été l'acquisition de notions de géométrie discrète pour aborder de façon efficace les aspects théoriques propres à la squelettisation. Ensuite, il a fallu passer à la réalisation d'une librairie de programmes pour représenter et manipuler les objets dont il est question avant de passer à l'implantation proprement dite des algorithmes de squelettisation. Il a aussi été indispensable de se familiariser avec les divers outils disponibles au sein du laboratoire, qui m'ont permis aussi de comprendre les bases de traitement d'images qui manquaient à ma formation. Avec au départ des connaissances en imagerie très générales, j'ai pu aborder des thèmes nouveaux qui se sont intégrés normalement dans un travail de recherche.

Après avoir implanté ces algorithmes, nous avons mis au point une méthode de traitement d'image, faisant appel à la squelettisation, aux notions de surfaces discrètes ainsi qu'a des méthodes mathématiques pour par exemple calculer des aires approchées de surfaces discrètes ou encore effectuer des rotations dans $\mathbb{R}^3$.

1. E.S.I.E.E: Ecole Supérieure d'Ingénieurs en Electronique et Electrotechnique, Paris
2. G.R.E.Y.C. / I.S.M.R.A.

Chapitre 1

Présentation

1.1 L'espace de travail

Notre domaine de travail est l'ensemble des objets que l'on peut représenter dans un espace discret à trois dimensions, typiquement $\mathbb{Z}^3$ noté E dans la suite. Même si la taille des objets manipulés est limitée par les temps de traitement parfois énormes, plus que par la capacité de stockage des ordinateurs actuels, tout objet acquis avec une résolution adaptée peut être manipulé.

Tous les objets que l'on manipule sont dit binaires, c'est à dire que la seule information dont on dispose est la présence (resp. l'absence) de "matière" en un point donné de l'espace. C'est la donnée d'un tableau binaire à trois dimensions que l'on peut voir comme une fonction:

$$\begin{aligned} f: \quad & \mathbb{Z}^3 \longrightarrow \{0,1\} \\ & (x,y,z) \longmapsto 0 \text{ ou } 1 \end{aligned}$$

Il est important de préciser que l'image à traiter est totalement dépourvue d'informations quantitatives (comme des niveaux de gris). Les algorithmes de base qui seront présentés ont pour seul intérêt l'étude des propriétés géométriques et topologiques d'un objet. Mais cela n'exclut pas des applications dans le traitement des images complexes, puisque l'image à manipuler peut très bien être le résultat de traitements effectués en amont qui ont abouti à une image binaire. On pense classiquement à une opération de segmentation d'image. Expliquons cela par une application classique de la squelettisation en dimension 2 : la reconnaissance de caractères. Prenons un texte écrit au feutre sur un papier aux couleurs arc-en-ciel (fig. 1.1). Chercher à squelettiser un tel objet ne nous mènera nulle part ! Toutefois, on imagine sans problème la difficulté du travail qui consiste à séparer l'écriture du fond. Ce travail, une fois effectué, fournit une image binaire (en noir et blanc) de l'écriture. C'est sur cette image que le processus de squelettisation, qui fait partie du programme de reconnaissance optique, pourra

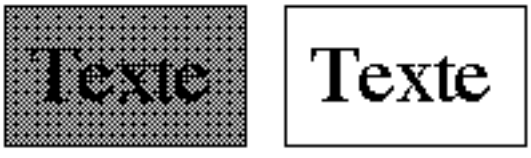


FIG. 1.1 – *Une image avant et après seuillage.*

s'appliquer.

La squelettisation n'est donc pas une opération fortement liée aux images binaires mais offre des applications variées si elle est utilisée en complément de méthodes classiques de traitement d'image.

1.2 Le principe général

Il existe des algorithmes de squelettisation qui reposent sur des propriétés métriques du squelette. Nous leur avons préféré des algorithmes topologiques qui reposent sur l'érosion. Ce terme "érosion" désigne en fait la méthode intuitive qui tend à copier la méthode manuelle utilisée pour amincir "physiquement" un objet. On enlève de manière séquentielle de la "matière" à un objet. Tout au long de cette opération, on prend soin de ne pas couper l'objet et de ne pas le trouer. On verra que si l'objet est envisagé comme un graphe construit à partir d'une relation d'adjacence entre points, alors ces "précautions" à prendre sont facilement exprimables en termes de la théorie des graphes.

L'algorithme brutal est le suivant:

Tant qu'il reste des points a enlever
  Choisir un point p de l'objet X
  Si p peut être enlevé sans créer de trou et sans couper l'objet
  Alors Effacer p de X

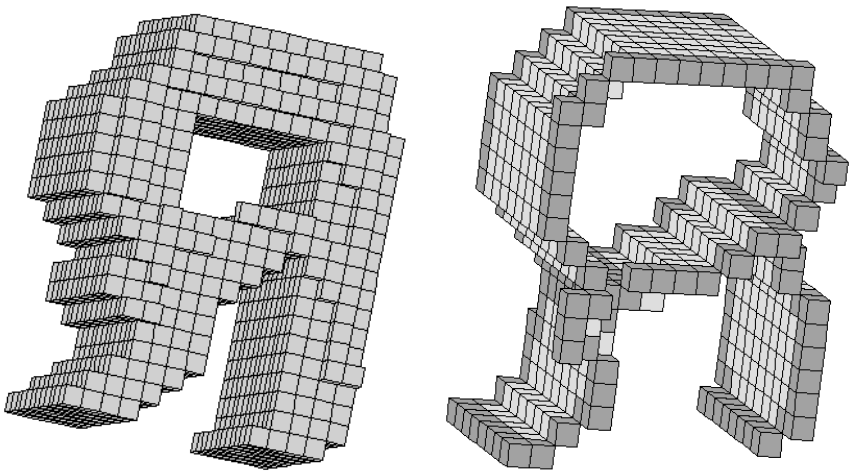
La condition *"Si P peut être enlevé..."* traduit le fait que la présence de ce point n'apporte rien à la topologie de l'objet. Cette propriété se définit de la manière suivante: on trouve le même nombre de composantes connexes, les mêmes trous dans X et $\overline{X}$ que le point p soit ou non dans X. On dit alors que p est un *point simple*.

Nous verrons que cette méthode fournit bien ce qu'on lui demande, à savoir ce que l'on appelle un *noyau homotopique* d'un objet donné. Toutefois, ce noyau, s'il est représentatif de la topologie de l'objet de départ, n'en conserve pratiquement pas les propriétés géométriques comme par exemple les convexités. Pour cela, on apporte des améliorations en ajoutant des conditions d'arrêt qui empêchent la suppression de certains points afin d'obtenir un squelette ayant de meilleures propriétés métriques. Plus précisément, nous chercherons à obtenir des squelettes filaires ou des squelettes surfaciques ce qui nous conduira à considérer des notions de surfaces discrètes.

De plus, on verra qu'il existe des méthodes qui rendent cet algorithme parallèle en permettant des suppressions simultanées de points. Une autre méthode simule ce parallélisme par un parcours séquentiel mais ordonnancé des points de l'image.

1.3 Exemples de résultats attendus

La figure 1.2 montre un exemple de squelette surfacique (notion à préciser par la suite). Le squelette est représentatif de la topologie de l'objet (a). Il a le même nombre de composantes connexes et le même trou. Mais il est aussi représentatif de la géométrie de l'objet (a) car les "branches" et convexités sont elles aussi conservées.



(a) (b)

FIG. 1.2 – *Un gros R en dimension 3 et son squelette.*

1.4 Les applications

Les applications de ces algorithmes sont trouvées dans les domaines où on a besoin de ne garder d'une image que le minimum d'information concernant la topologie et la géométrie d'un objet ou d'un ensemble d'objets. Dans le cadre de ce stage, on a mis au point une méthode de recalage d'images I.R.M. dans laquelle la squelettisation intervient pour "réduire" de façon considérable la taille de la donnée. On applique préalablement la segmentation (évoquée précédemment) d'une image pour extraire une partie de l'objet sur laquelle la squelettisation a un sens.

Cette méthode fait aussi appel aux définitions de surfaces qui seront étudiées. Tout d'abord parceque le squelette recherché est de type surfacique. Ensuite parce qu'on ne conservera du squelette que l'ensemble des points vérifiant une certaines définition de surface discrète afin d'obtenir des régions significatives et stables. Toute cette application est exposée au chapitre 4.

Il existe en effet plusieurs types de squelettes: le plus simple étant appellé le *noyau homotopique* qui est en quelque sortes un squelette "minimum". Par contre on verra qu'il est aussi possible d'obtenir des squelettes à l'aspect filaire ou encore surfacique. Ces différents types de squelettes seront exposés dans la suite (voir 3.3).

Chapitre 2

Préliminaires Théoriques

Tous les algorithmes décrits par la suite sont basés sur l'examen de configurations locales. On attribue en effet à chaque point d'un objet une propriété qui n'est fonction que de l'examen de son voisinage.

2.1 Les relations de voisinage

2.1.1 Définitions

On notera toujours $\overline{X}$ le complémentaire de X, soit le *fond* de l'image. On peut considérer 3 (en fait 4) types de voisinages dans $\mathbb{Z}^3$, chacun munissant cet espace d'une structure de graphe. On définit pour chaque point $x = (x_1, x_2, x_3)$ de $\mathbb{Z}^3$ les ensembles:

- $N_{26}(x) = \{x' \in E;\ Max[|x_1 - x'_1|; |x_2 - x'_2|; |x_3 - x'_3|] \leq 1\}$
- $N_{18}(x) = \{x' \in E;\ |x_1 - x'_1| + |x_2 - x'_2| + |x_3 - x'_3| \leq 2\} \cap N_{26}(x)$
- $N_6(x) = \{x' \in E;\ |x_1 - x'_1| + |x_2 - x'_2| + |x_3 - x'_3| \leq 1\}$

On notera aussi $N^*_n(x) = N_n(x) \setminus \{x\}$ pour $n \in \{6, 18, 26\}$.
A partir de ces ensembles, on définit une relation d'adjacence: deux points x et y sont dit n-adjacents si $y \in N^*_n(x)$. On parle alors de 6-voisinage, de 18-voisinage et de 26-voisinage qui sont présentés dans la figure 2.1.
Cette relation permet de représenter toutes les parties de $\mathbb{Z}^3$ par une structure de donnée de type graphe. Les sommets de ce graphe étant les points à 1 de l'objet, une arête relie deux sommets si les points associés sont n-adjacents.

On définit sur $\mathbb{Z}^3$ les chemins en utilisant la définition classique des chemins dans un graphe. Un n-chemin est une liste ordonnée de points $x_0, \ldots, x_k$ telle que x_i est n-adjacent à x_{i-1} pour $i = 1 \ldots k$. La longueur d'un chemin valant $k - 1$ (le nombre d'arêtes). Un chemin est dit élémentaire si tous les x_i sont distincts, sauf x_0 et x_k. Il est dit *simple* si tout point possède au plus deux voisins. Un chemin est *fermé* si $x_0 = x_k$.

Cette notion de chemin permet d'introduire la relation de connexité entre deux points. Ainsi, deux points sont dit n-connectés si il existe un n-chemin reliant x à y. Cette relation d'équivalence partitionne toute partie de $\mathbb{Z}^3$ en classes d'équivalence qui sont les *composantes n-connexes* de l'objet considéré. L'ensemble des classes d'équivalence par la relation de connexité d'un objet X est noté

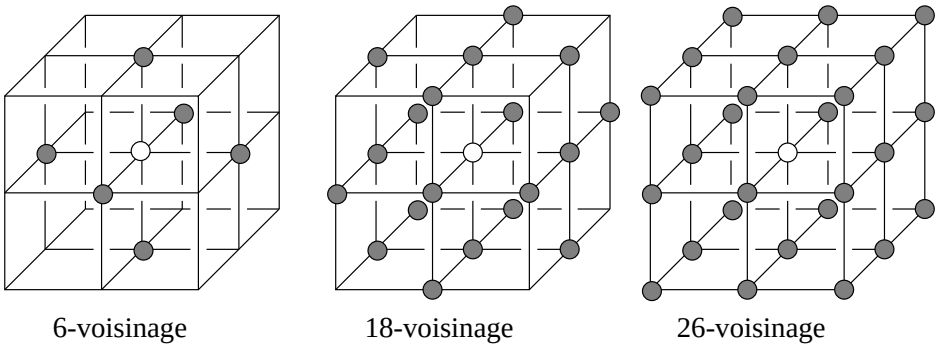


FIG. 2.1 – *Les voisinages classiques de $\mathbb{Z}^3$, (le point x est le point central)*

$C_n(X)$, c'est un ensemble de parties et non de points. Pour tout point x de $\overline{X}$, on notera $C_n^x(X)$ l'ensemble des composantes connexes de X qui sont n-adjacentes à x.

On dira notamment qu'un objet $X \subset \mathbb{Z}^3$ est n-connexe s'il ne possède qu'une classe d'équivalence par la relation précédente, soit $card(C_n(X)) = 1$. Ceci équivaut à dire que pour tout point x et y de X, il existe un n-chemin reliant x à y dans X.

Muni de cette notion de connexité, on se rend compte qu'un objet n'aura pas les mêmes propriétés topologiques selon le type d'adjacence considéré. En effet, un objet connexe (i.e. composé d'une seule classe d'équivalence par la relation ci-avant) en 26-adjacence aura toutes les chances d'être composé d'un grand nombre de composantes connexes en 6-adjacence. D'autre part, un objet sans trou du point de vue de la 6-adjacence pourra en avoir si on le considère avec la 26-adjacence.

2.1.2 Dualité des adjacences entre un objet et son fond

En géométrie discrète, si on veut traiter de la topologie d'un objet c'est à dire de ses composantes connexes, de ses trous et de ses cavités (notions expliquées en 2.2); il nous faut considérer une adjacence différente selon qu'on se place dans l'objet ou dans son complémentaire. Ceci se justifie immédiatement par la nécessité de vérifier le théorème de Jordan. La figure 2.2 doit convaincre qu'en 2D, une courbe fermée du point de vue de la 8-adjacence (dans $\mathbb{Z}^2$) sépare toujours son complémentaire en 2 composantes 4-connexes. Dans $\mathbb{Z}^3$, on peut utiliser les couples $(n, \overline{n})$ suivants (n-adjacence pour l'objet, $\overline{n}$-adjacence pour le fond):

$$(n, \overline{n}) = (6, 26), (6+, 18), (26, 6), (18, 6+).$$

Notez le besoin de distinguer par la notation la 6-adjacence associée à la 26-adjacence de la 6+ qui est associée à la 18. On verra plus loin que certaines notions ne coïncident pas entre la 6-adjacence et la 6+-adjacence.

2.2 Trous et cavités

On a décrit le principe général des algorithmes de squelettisation en posant la condition de conservation du nombre de trous, de cavités et de composantes connexes de l'image de départ. On a déjà défini ce que sont les composantes connexes d'un objet. Les trous dans $\mathbb{Z}^2$ correspondent à des composantes connexes du complémentaire, différentes du fond. Leur équivalent dans $\mathbb{Z}^3$ sont les cavités.

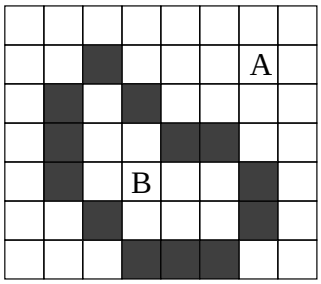


FIG. 2.2 – *Une 8-courbe fermée sépare le fond en deux composantes 4-connexes.*

Ainsi, une sphère creuse possède une cavité. Mais on a aussi en 3D une notion de trous, dont la détection est moins simple et fait appel à la notion de déformation de chemins. Pour formaliser ceci on définit le Groupe Fondamental.

2.3 Le Groupe Fondamental

On détecte la présence d'un trou par l'existence d'un chemin dans l'objet qui ne puisse être déformé en un point unique. On définit d'abord une relation de n-déformation élémentaire entre deux chemins.

Définition 1 *pour* $(n, \overline{n}) = (26, 6)$ *ou* $(18, 6+)$. *Soit* $p \in X$ *un point appelé* point de base. *Soient* $\gamma \subset X$ *et* $\gamma' \subset X$ *deux* n*-chemins fermés de* p *à* p. *On dit que* γ' *est une* n-déformation élémentaire *de* γ, *que l'on note* $\gamma \sim \gamma'$, *si* γ *et* γ' *sont de la forme :* $\gamma = \pi_1.\pi.\pi_2$, $\gamma' = \pi_1.\pi'.\pi_2$ *avec* π *et* π' *ayant les mêmes extrémités et inclus dans un même cube unité (un cube 2×2×2).*

Brièvement, elle met en relation deux chemins issus d'un même point p s'ils sont identiques sauf dans un petit voisinage. Ensuite on définit par fermeture transitive la relation de n-déformation:

Définition 2 *On dit que* γ' *est une* n-déformation *de* γ, *noté* $\gamma \simeq \gamma'$ *s'il existe une suite de* n*-chemins fermés* $\gamma_0..\gamma_k$ *tels que* $\gamma = \gamma_0$, $\gamma' = \gamma_k$ *et* $\gamma_{i-1} \sim \gamma_i$ *pour* $i = 1..k$.

L'ensemble des classes d'équivalence par $\simeq$, muni de la loi interne de concaténation de chemins, forme le Groupe Fondamental $\Pi(p, X)$. Ce groupe est indépendant à isomorphisme près du choix de p pour peu qu'il se trouve dans une même composante connexe. Ce Groupe Fondamental caractérise bien les

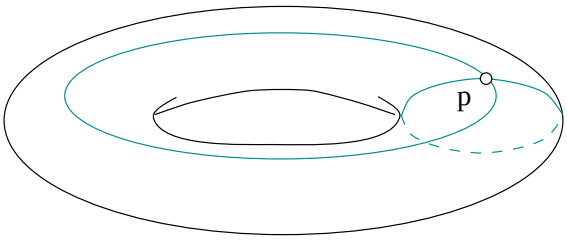


FIG. 2.3 – *Le tore creux et deux chemins générateurs.*

trous de chaque composante connexe d'un objet. Ainsi, la figure 2.3 montre que le groupe fondamental d'un tore creux est $\mathbb{Z}^2$. En effet, on peut trouver deux chemins (appelés des générateurs du groupe) non homotopes et qui ne peuvent se déformer en un point; de plus ces deux générateurs commutent.

Remarquons que dans ce cas, la cavité du tore forme aussi un trou dans l'objet. Ce n'est généralement pas le cas (pour une sphère creuse par exemple).

Soient maintenant $X \subset Y \subset \mathbb{Z}^3$. Un chemin dans X est un cas particulier de n-chemin dans Y. De plus, si deux n-chemins sont n-équivalents dans X, ils sont notament n-équivalents dans Y. Ces deux dernières propriétés indiquent que l'inclusion $i : Y \longrightarrow X$ *induit* un morphisme de groupes $i_* : \Pi(p, X) \longrightarrow \Pi(p, Y)$.

2.4 Voisinages géodésiques

On a besoin de caractériser localement les points qui peuvent être effacés sans modifier la topologie de l'objet. G. Bertrand [1] a introduit pour ce faire des ensembles très pratiques, les voisinages géodésiques.

Définition 3 *Soit $X \subset \mathbb{Z}^3$ et $x \in \mathbb{Z}^3$. Le* voisinage géodésique *de x dans X d'ordre k, noté $N_n^k(x, X)$ est défini par:*

$$N_n^1(x, X) = N_n^*(x) \cap X$$

et

$$N_n^k(x, X) = \bigcup_{y \in N_n^{k-1}(x,X)} (N_n(y) \cap N_{26}^*(x) \cap X)$$

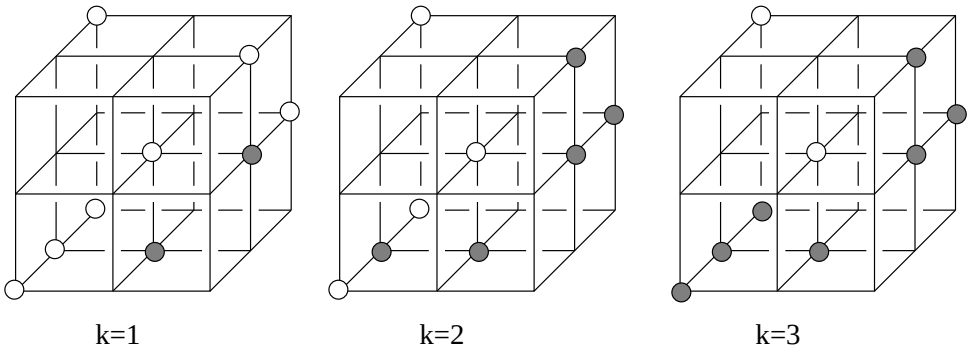


FIG. 2.4 – *Voisinages géodésiques: En gris, $N_6^k(x, X)$ pour k=1,2 et 3.*

C'est l'ensemble des points y du 26-voisinage de x tels qu'il existe un chemin de longueur inférieure ou égale à k reliant x à y, tous les points du chemin appartenant à $X \cup \{x\}$. La figure 2.4 montre un exemple en 6-adjacence.

On définit ensuite des *voisinages géodésiques* usuels $G_n(x, X)$:

- $G_6(x, X) = N_6^2(x, X)$
- $G_{6+}(x, X) = N_6^3(x, X)$
- $G_{18}(x, X) = N_{18}^2(x, X)$
- $G_{26}(x, X) = N_{26}^1(x, X)$

Et enfin le *nombre topologique* est défini en tout point de X par:

$$T_n(x, X) = Card(\ C_n[G_n(x, X)])$$

2.5 Homotopie et Points Simples

Définition 4 *Un point $x \in X$ est dit n-simple si le fait de l'effacer (le supprimer de X) ne change pas la topologie de l'objet X. C'est à dire qu'il existe un bijection entre d'une part les composantes connexes de X et $\overline{X}$; et d'autre part les composantes connexes de $X \setminus \{x\}$ et $\overline{X} \cup \{x\}$. En considérant la n-adjacence dans X et la $\overline{n}$-adjacence dans $\overline{X}$.*
De plus, il faut que:

- *le morphisme $i_* : \Pi(p, X \setminus \{x\}) \longrightarrow \Pi(p, X)$ induit par l'inclusion soit un isomorphisme;*
- *le morphisme $i'_* : \Pi(p, \overline{X}) \longrightarrow \Pi(p, \overline{X \setminus \{x\}})$ induit par l'inclusion soit un isomorphisme.*

Définition 5 *Une partie $X \subset Y$ est dite (sous) n-homotope à Y si et seulement si X peut être obtenu à partir de Y en lui enlevant de manière* séquentielle *des points n-simples. Si X est (sous) n-homotope à Y, alors l'ensemble $S = Y \setminus Z$ est appelé ensemble n-simple.*

Remarque: L'homotopie peut être vue comme une relation symétrique entre deux ensembles. Toutefois, dans la suite, on utilisera toujours le terme d'homotopie entre deux ensembles X et Y dans le seul cas où le premier est une partie du second.

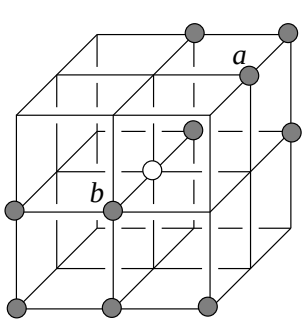
Théorème 1 *[1, Bertrand] Soit $X \subset E$ et $x \in X$: x est un point n-simple si et seulement si $T_n(x, X) = 1$ et $T_{\overline{n}}(x, \overline{X}) = 1$.*

On dispose ainsi d'une caractérisation locale des points n-simples à l'aide des voisinages géodésiques. Ceci donne un algorithme simple qui teste si un point peut être effacé en examinant seulement le voisinage géodésique $N_n^k(x, X)$ inclus dans son 26-voisinage $N_{26}^*(x)$. Il suffit pour cela de construire deux voisinages géodésiques, un pour X et un pour $\overline{X}$; et d'en déterminer le nombre de composantes connexes.

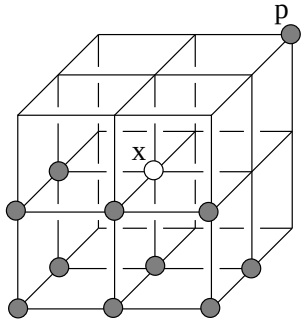
Pour se convaincre, le lecteur peut examiner la figure 2.5 dans laquelle on donne la justification par les nombres topologiques de la (non) simplicité de deux configurations. Ainsi, on voit bien que si on enlève le point central de la configuration (a), alors on déconnecte "localement" l'objet du point de vue de la 18 connexité. Par contre, si on regarde cette même configuration avec l'oeil de la 26 connexité, alors le point central peut être effacé puisque deux points (a et b) maintiennent connexe le tout. Le cas de la figure (b) est laissé à titre d'exercice !

Plus simplement, on voit bien que l'ensemble des points gris de la figure 2.5(c) forment un 18 ou un 26-chemin fermé dans X. Quand le point central est dans X, ce chemin peut être déformé (cf définition 2) en un unique point. Or si on enlève le point central, cette déformation devient impossible. On a créé un trou dans l'objet, le point central n'est donc pas simple. Pour être rigoureux, il faut aussi considérer le cas ou le chemin est tout de même déformable en un point en le déformant dans l'objet total. Or, on démontre alors que l'objet possède une cavité que l'on a supprimé en effaçant le point x. On peut penser au fait de percer en un point une sphère fine et creuse.

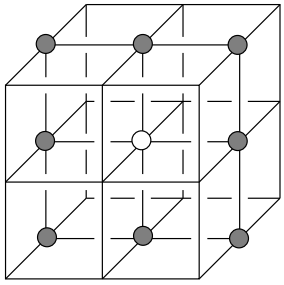
Dans le cas de la figure 2.5(d), il est clair que la situation est identique si on considère la 26-adjacence. Le point central n'est donc pas 26-simple. Par contre, du point de vue de la 18-adjacence, les point gris ne forment plus un 18-chemin fermé (car les point a et b ne sont pas 18-adjacents). Enlever le point central ne créera donc pas de trou. Ce point est 18-simple. Cela est confirmé par le fait que dans ce



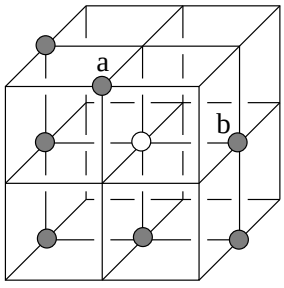
(a)
$T_{26}(x, X) = T_6(x, \overline{X}) = 1$
$\Rightarrow$ Point 26-simple
$T_{18}(x, X) = 2$
$\Rightarrow$ Point non 18-simple



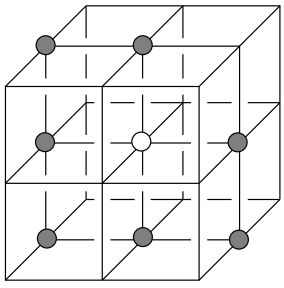
(b)
$T_{26}(x, X) = 2$
$\Rightarrow$ Point non 26-simple
$T_{18}(x, X) = T_6(x, \overline{X}) = 1$
$\Rightarrow$ Point 18-simple



(c)
Non 18-simple et non 26-simple



(d)
18-simple mais non 26-simple



(e)
non 18-simple et non 26-simple

FIG. 2.5 – *Des configurations locales de points simples et non-simples*

cas $T_{6+}(x, \overline{X}) = Card[N_6^3(x, \overline{X})] = 1$ pour $(n, \overline{n}) = (18, 6+)$ alors que $T_6(x, \overline{X}) = Card[N_6^2(x, \overline{X})] = 2$ pour $(n, \overline{n}) = (26, 6)$.

Un autre cas peut aussi se produire, celui illustré par la figure 2.5(b). Dans le cas de la 26-adjacence, le point p est connecté aux autres points via le point x. Si on supprime le point central, alors on aura créé *localement* 2 composantes connexes dans ce voisinage. Deux choses peuvent alors se produire du point de vue global. Soit p n'est pas connecté aux autres points gris "par l'extérieur", auquel cas on aura effectivement déconnecté globalement l'objet, p devenant une nouvelle composante connexe. L'autre cas demande un peu plus d'imagination ! Si p est en effet connecté aux autres points gris, c'est qu'il existe un 26-chemin dans l'objet reliant p à l'un de ces points. On peut se convaincre en examinant la configuration locale que ce chemin n'est pas déformable en un unique point. Cette hypothèse met en évidence la présence d'un trou dans l'objet, trou qui disparaîtrait si on supprimait le point x puisqu'on couperait alors le chemin évoqué précédemment. Ceci justifie "intuitivement" la condition $T_n(x, X) = 1$ qui dit simplement que le voisinage géodésique du point x doit être connexe (et pas composé de plusieurs composantes connexes qui se trouveraient séparées si on enlevait x).

2.6 Homotopie forte et Points P-Simples

La définition 5 montre que la transformation utilisée pour déformer un objet en un autre est une suppression *séquentielle* de points. Cela signifie que les points doivent être enlevés en testant la simplicité à chaque étape si on veut que la topologie de l'objet soit conservée. On voudrait disposer d'une notion plus forte, avec laquelle on pourrait enlever plusieurs points en parallèle sans détruire la topologie. Ceci nous amène à la notion d'homotopie forte puis à la définition des *points P-simples.*

Définition 6 *Soient $X \subset Y \subset E$. L'ensemble X est dit* fortement (sous) n-homotope *à Y si, $\forall Z$ tel que $X \subset Z \subset Y$, Z est (sous) n-homotope à Y.*
Si X est fortement n-homotope à Y alors l'ensemble $S = Y \setminus X$ est appelé ensemble fortement n-simple

Définition 7 *Soient $X \subset E$, $P \subset X$ et $x \in X$.*
x est dit P_n-simple si $\forall S \subset P \setminus \{x\}$, x est n-simple pour $X \setminus S$. On note $S_n(P)$ l'ensemble des points P_n-simples. Un ensemble D est dit P_n-simple si $D \subset S_n(P)$.

La définition 6 exprime le fait que l'on peut effacer n'importe quelle partie de $Y \setminus X$, l'objet obtenu est toujours homotope à Y. De là, il est immédiat de montrer que cela est équivalent au fait que l'on peut supprimer les points de $Y \setminus X$ dans un ordre quelconque sans jamais modifier la topologie de l'objet.

La définition 7 introduit une propriété forte de certains points, à savoir que l'on peut les supprimer quelque soit la partie d'un ensemble donné P précédemment effacée. Cette propriété, dont on va voir qu'elle est aussi caractérisée localement, permettra de paralléliser les algorithmes de squelettisation (voir 3.2).

Maintenant, on sent bien que les notions d'homotopie forte et d'ensemble P-simple sont très liées. En effet, il est facile de démontrer le théorème suivant:

Théorème 2 *Soit $X \subset E$. Une partie Y de X est* fortement sous n-homotope *à X si et seulement si l'ensemble $P = Y \setminus X$ est un ensemble P_n-simple pour X. Autrement dit si et seulement si $\forall x \in Y \setminus X$, x est $(Y \setminus X)$-simple.*

On définit la simplicité d'un point de façon locale. Si on utilise cette caractérisation dans la définition des points P_n-simples on obtient:

$$x \text{ est } P_n\text{-simple si et seulement si } \forall S \subset P \cap N_{26}^*(x),\ T_n(x, X \setminus S) = 1 \text{ et } T_{\overline{n}}(x, \overline{X \setminus S}) = 1.$$

Cette caractérisation, si elle est suffisante, impose tout de même un grand nombre de calculs de voisinages géodésiques. En effet, il faudrait alors vérifier que $T_n(x, X \setminus S) = 1$ et $T_{\overline{n}}(x, \overline{X \setminus S}) = 1$, et ce pour tous les ensembles $S \subset P \cap N_{26}^*(x)$. Au pire (si $P \cap N_{26}^*(x) = N_{26}^*(x)$), il en existe exactement 2^{26} ! Il faut garder à l'esprit le fait que ces tests sont répétés en pratiquement chaque point d'une image.
A des fins d'efficacité évidentes, on expose une deuxième propriété qui mènera à une méthode moins combinatoire.

Théorème 3 *[1, Bertrand] Soient $P \subset X$ et $x \in P$, on note $R = X \setminus P$.*
x est P_n-simple si et seulement si

$$\begin{cases} T_n(x, R) = 1 & (1) \\ T_{\overline{n}}(x, \overline{X}) = 1 & (2) \\ \forall y \in N_n^*(x) \cap P,\ T_n(x, R \cup \{y\}) = 1 & (3) \\ \forall y \in N_{\overline{n}}^*(x) \cap P,\ T_{\overline{n}}(x, \overline{X} \cup \{y\}) = 1 & (4) \end{cases}$$

Preuve ($\Rightarrow$) Supposons que x soit P_n-simple, par définition x est n-simple pour X, le théorème 1 donne (2).
De même, x est n-simple pour $X \setminus (P \setminus \{x\}) = R \cup \{x\}$. Donc $T_n(x, R \cup \{x\}) = 1$ et comme $T_n(x, R \cup \{x\}) = T_n(x, R)$ on a bien (1).
Soit $y \in N_n^*(x) \cap P$, x est n-simple pour $R \cup \{x, y\}$ donc $T_n(x, R \cup \{x, y\}) = T_n(x, R \cup \{y\}) = 1$. Soit $y \in N_{\overline{n}}^*(x) \cap P$, x est n-simple pour $X \setminus \{y\}$, donc $T_{\overline{n}}(x, \overline{X \setminus \{y\}}) = T_{\overline{n}}(x, \overline{X} \cup \{y\}) = 1$.

($\Leftarrow$) Supposons que les 4 conditions soient vérifiées.
Dans un premier temps, on va montrer que x est un point simple.
D'après la condition (1), $G_n(x, R)$ ne comporte qu'une seule composante n-connexe. Comme $R \cup P = X$ on a $G_n(x, R) \subset G_n(x, X)$. Comme tous les points de P n-adjacents à x remplissent la condition (3), cela signifie qu'ils appartiennent à la même composante n-connexe de $G_n(x, X)$, celle qui contient précisément $G_n(x, R)$. Si tous les points de $N_n^*(x) \cap P$ sont dans une même composante n-connexe de $G_n(x, X)$, on en déduit que $G_n(x, X)$ est lui même composé d'une seule composante n-connexe et donc $T_n(x, X) = 1$. Avec (2) on en conclu que x est n-simple pour X.

Il nous faut maintenant montrer que pour toute partie $S \subset P$ que l'on enlève de X, x reste simple pour $X \setminus S$. Comme on a montré que:

$$(1),(2),(3),(4) \Rightarrow x \text{ est } n\text{-simple pour } X,$$

On va utiliser ce résultat en montrant que ces 4 propriétés restent vraies pour tout ensemble $S \subset P$ ôté de X. Pour cela utilisons une récurrence sur le nombre d'éléments de S. On suppose (ce qui est vrai compte tenu de ce qui précède) que x est n-simple pour X, on montre que cela reste vrai pour $X \setminus \{s\}$, où $s \in P \cap N_{26}^*(x)$. En effectuant à chaque pas le changement de variable $X := X \setminus \{s\}$ et $S := S \cup \{s\}$, on aura construit pas à pas tout sous-ensemble $S \subset P \cup N_{26}^*(x)$ que l'on peut effacer de X.

1) $T_n(x, X \setminus P) = 1$ est toujours vraie puisque le point effacé s faisait partie de P.

2) D'après la condition (4) supposée vraie pour X on a $T_{\overline{n}}(x, \overline{X}) = 1$.
Si $s \in N^*_{\overline{n}}(x)$, alors, toujours d'après (4) on a $T_{\overline{n}}(x, \overline{X \setminus \{s\}}) = T_{\overline{n}}(x, \overline{X} \cup \{s\}) = 1$.
Si $s \notin N^*_{\overline{n}}(x)$, alors on montre que le fait d'enlever s (l'ajouter à $\overline{X}$) ne peut créer de nouvelle composante connexe dans $G_{\overline{n}}(x, \overline{X})$. En effet, supposons le contraire, c'est à dire que $s \in G_{\overline{n}}(x, \overline{X} \cup \{s\})$ soit dans la nouvelle composante connexe créée. Plaçons nous dans le cas $(n, \overline{n}) = (26, 6)$ pour exemple. Il existe alors un 6-chemin de longueur au plus 2 (définition 3 de $G_{\overline{n}}(x, \overline{X} \cup S)$) dans $N_{26}(x) \cap \overline{X}$ reliant s à x. Soit p le dernier point avant x de ce chemin. D'après la condition (2) appliquée à X, le point p ne peut pas être dans $G_{\overline{n}}(x, \overline{X})$ sans quoi p serait connecté à l'unique composante connexe de $G_{\overline{n}}(x, \overline{X})$. Or $p \in (N^*_6(x) \cap \overline{X}) \subset G_6(x, \overline{X})$. D'où la contradiction et le fait que $T_6(x, \overline{X \setminus \{s\}}) = 1$.

Ainsi, dans tous les cas on a $T_n(x, \overline{X \setminus \{s\}}) = 1$.

3) Les points restant dans $N^*_n(x) \cap (P \setminus \{s\})$ vérifient toujours la propriété (3) (R n'étant pas affecté par la suppression d'un point de P):
$\forall y \in N^*_n(x) \cap (P \setminus \{s\}),\ T_n(x, R \cup \{y\}) = 1$.

4) D'après (4) $T_{\overline{n}}(x, \overline{X \setminus \{s\}}) = 1$, $G_{\overline{n}}(x, \overline{X \setminus \{s\}}) = G_{\overline{n}}(x, \overline{X} \cup \{s\})$ contient une seule composante $\overline{n}$-connexe. Soit $y \in N^*_{\overline{n}}(x) \cap (P \setminus \{s\})$. Si y satisfait (4) pour P, l'ajout de y à $\overline{X}$ ne peut créer de nouvelle composante connexe dans $G_{\overline{n}}(x, \overline{X} \cup \{s\})$. En effet, supposons que $G_{\overline{n}}(x, \overline{X} \cup \{s\} \cup \{y\})$ comporte plus de composantes connexes que $G_{\overline{n}}(x, \overline{X} \cup \{s\})$. Or $y \in N^*_{\overline{n}}(x) \cap (P \setminus S) \Rightarrow y \in G_{\overline{n}}(x, \overline{X} \cup \{y\})$. y étant dans la nouvelle composante connexe de $G_{\overline{n}}(x, \overline{X} \cup \{s\} \cup \{y\})$, il n'existe pas de $\overline{n}$-chemin dans cet ensemble de y à un point de $G_{\overline{n}}(x, \overline{X} \cup \{s\}) \supset G_{\overline{n}}(x, \overline{X})$. A fortiori, il n'y en a pas non plus dans $G_{\overline{n}}(x, \overline{X} \cup \{y\})$ joignant y à un point de $G_{\overline{n}}(x, \overline{X})$. Donc $G_{\overline{n}}(x, \overline{X} \cup \{y\})$ n'est pas $\overline{n}$-connexe, ce qui contredit la condition 4 vérifiée pour le pas précédent.
Finalement, $\forall y \in N^*_{\overline{n}}(x) \cap (P \setminus \{s\}),\ T_{\overline{n}}(x, \overline{X \setminus \{s\}} \cup \{y\}) = 1$.

En conclusion, x vérifie toujours les 4 conditions pour $X \setminus (S \cup \{s\})$, donc x est n-simple pour $X \setminus (S \cup \{s\})$. Par induction, on voit que l'on peut effacer les uns après les autres des points choisis dans $S \subset P \cap N^*_{26}(x)$ et qu'au cours de cette opération, le point x reste simple pour l'objet intermédiaire, quelque soit le S choisi. Ceci signifie (définition 7) que x est P_n-simple.

Cette caractérisation rend les tests déjà beaucoup plus rapides puisqu'on ne construit maintenant qu'au plus 26 voisinages géodésiques pour tester la P-simplicité d'un point [1]. Mais on peut faire encore mieux, on dispose en effet d'un lemme très utile pour rendre plus efficace la vérification des propriétés (3) et (4) du théorème 3.

Lemme 4 *Soit $m \in \{6, 6+, 18, 26\}$ et $d \in \mathbb{N}$. Soit $D \subset \mathbb{Z}^3$ et $x \in \overline{D}$ tel que $Card[C_m(N^d_m(x, D))))] = 1$, et $y \in N_m(x) \cap \overline{D}$. Alors, $Card[C_m(N^d_m(x, D \cup \{y\}))] = 1$ si et seulement si il existe un m-chemin π de y a x, de longueur l, $3 \leq l \leq 2d + 1$, tous les points de π appartenant à D sauf x et y.*

Notation On pose $l(6) = 5$, $l(6+) = 7$, $l(26) = 3$ et $l(18) = 5$.
Grâce à ce lemme, la condition (3) du théorème 3 (la (4) d'une manière similaire) peut être vérifiée en construisant *un seul* voisinage géodésique $G^{2.l(n)-1}_n(x, R)$. Ensuite, il reste à tester que tous les y de $N^*_n(x) \cap P$ sont n-adjacents à un point de ce voisinage géodésique ce qui vérifie l'existence du chemin π. On en déduira alors que $Card(C_m(N^d_m(x, D \cup \{y\}))) = 1$.

1. Comparés aux 2^{26} évoqués précédemment, quel progrès !

2.7 Les surfaces discrètes

Le but de cette partie est de rappeler brièvement les principales notions de surfaces discrètes qui ont été définies à ce jour. Cette diversité est intéressante à considérer dans le cadre de ce travail sur la squelettisation. En effet, on verra que si on cherche à produire des squelettes ayant un aspect *surfacique*, alors on a besoin de caractériser les points dont le voisinage respecte des propriétés dites de *points de surface*. Considérer tel ou tel type de surface mènera alors à des squelettes différents.

2.7.1 Surfaces primaires

Cette notion de surface considère tous les ensembles *n-séparants*. On définit ces ensembles de points de la manière suivante:

Définition 8 *Soit $X \subset E$ une partie n-connexe de E. Soit $|X|^x = N_{26}^*(x) \cap X$. On dit de X qu'il est* n-fin *si, $\forall x \in X,\ Card(C_{\overline{n}}^x(|\overline{X}|^x)) = 2$.*
Un ensemble X est dit n-séparant *s'il est n-fin et $\overline{X}$ possède exactement 2 composantes $\overline{n}$-connexes. Appelons A et B les deux composantes connexes du complémentaire d'un tel ensemble. Si de plus, $\forall x \in X$, x est $\overline{n}$-adjacent à la fois à A et B, alors on dit que X est un* ensemble fortement n-séparant.

Notez qu'un gros problème de la géométrie discrète est de trouver des caractérisations *locales* pour ce type d'objet. La figure 2.7 montre deux configurations locales *pouvant* appartenir à un objet 26-séparant. Il est déjà utile de noter que dans le cas de la figure 2.7 (b), le point P forme une troisième composante de $C_{\overline{n}=26}(|\overline{X}|^x)$. Toutefois, seuls deux éléments de ce dernier ensemble sont 6-adjacents à x. D'où la vérification de cette partie "locale" de la caractérisation des objets séparants.

2.7.2 Surfaces de Morgenthaler

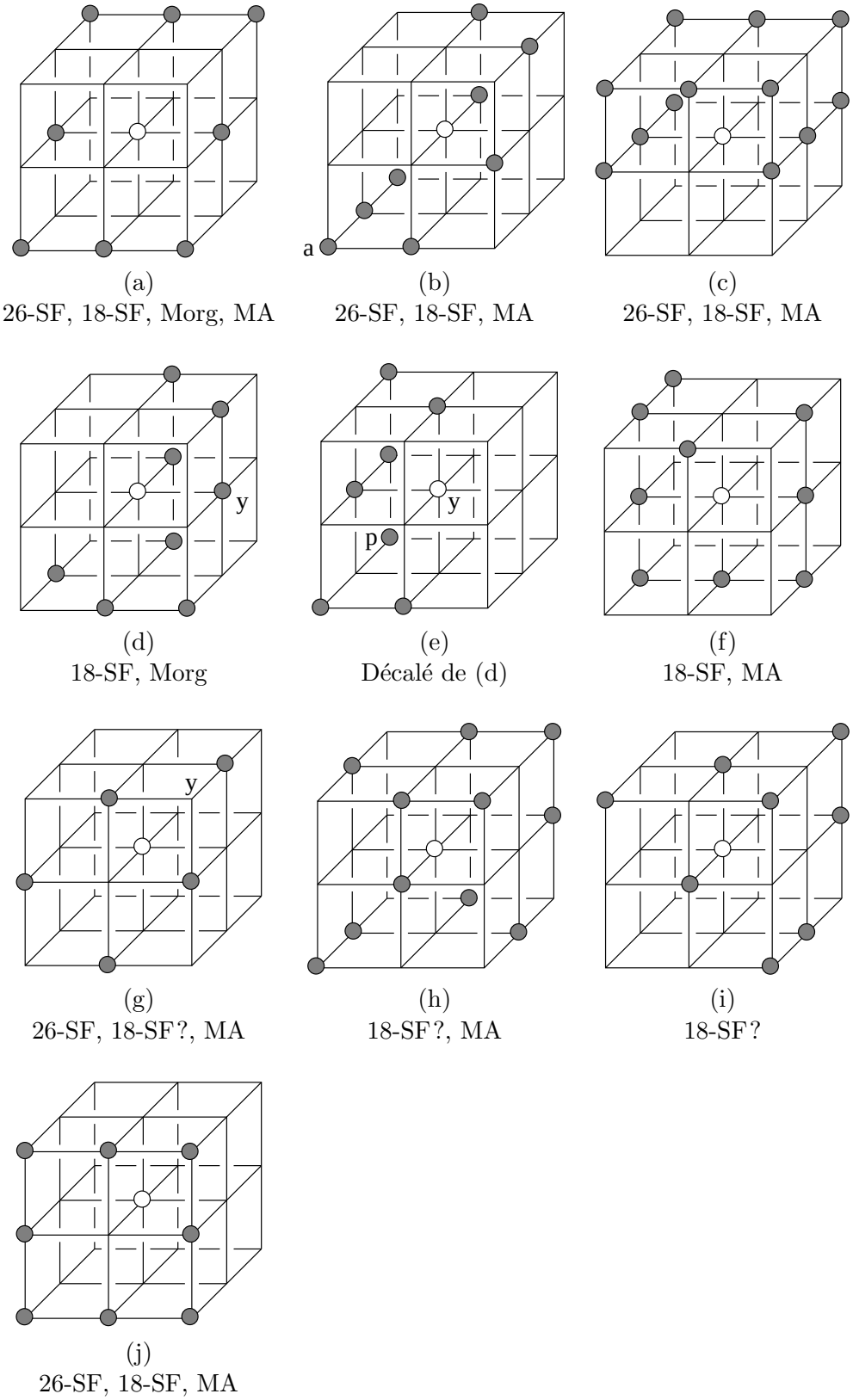
Ce type de surface à été introduite par D.G. Morgenthaler [7], elle se définit localement de la manière suivante:

Définition 9 *Pour $(n, \overline{n}) = (6, 26)$ ou $(26, 6)$ et X une partie de E.*
Un point $x \in X$ est dit point de n-surface (simple) de Morgenthaler *si les 3 propriétés suivantes sont satisfaites:*

1. $Card(C_n^x(|X|^x)) = 1$;
2. $Card(C_{\overline{n}}^x(|\overline{X}|^x)) = 2$; *on note* $C_{\overline{n}}^x(|\overline{X}|^x) = \{C^{xx}, D^{xx}\}$;
3. $\forall y \in N_n(x) \cap X,\ N_{\overline{n}}(y) \cap C^{xx} \neq \emptyset$ *et* $N_{\overline{n}}(y) \cap D^{xx} \neq \emptyset$.

De plus, si $Card(C_{\overline{n}}^x[N_{124}(x) \cap \overline{X}]) = 2$ on dit que le point de n-surface est orientable.
Une n-surface fermée (simple) de Morgenthaler *est un ensemble constitué uniquement de points de n-surface de Morgenthaler orientables.*

D'ores et déjà, on peut se rendre compte que cette notion est beaucoup plus restrictive que la précédente. On trouve en effet en 2) la condition des objets n-séparants mais on impose en 3) le fait que chaque point de $N_n(x) \cap X$ soit en contact (du point de vue de l' adjacence du complémentaire) avec chacune des deux composantes connexes C et D.



(a) 26-SF, 18-SF, Morg, MA
(b) 26-SF, 18-SF, MA
(c) 26-SF, 18-SF, MA
(d) 18-SF, Morg
(e) Décalé de (d)
(f) 18-SF, MA
(g) 26-SF, 18-SF?, MA
(h) 18-SF?, MA
(i) 18-SF?
(j) 26-SF, 18-SF, MA

FIG. 2.6 – *Des configurations locales de points de surface SF: Surface forte, MA: MA-surface, Morg: Morgenthaler*

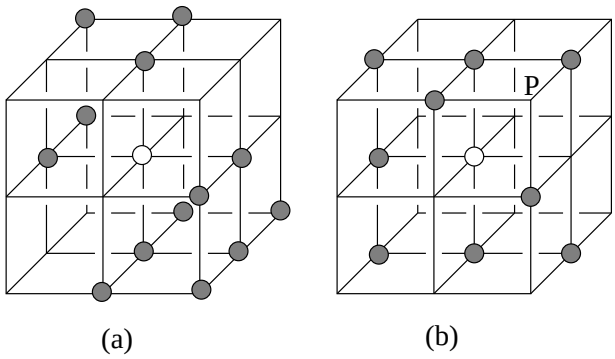


FIG. 2.7 – *Deux configurations locales possibles pour un objet 26-séparant.*

Tout de suite on s'aperçoit que de nombreux points gris ($\in N_{26}(x) \cap X$) de la figure 2.7(a) ne vérifient pas cette condition 3). Par contre la figure 2.8 montre une configuration possible d'un point de surface de Morgenthaler, sauf qu'il n'est pas question ici de représenter le 124-voisinage. Il suffit de se convaincre que cette configuration vérifie la condition de *point orientable*.

Intéressons nous à la configuration repésentée par la figure 2.6(b). Celle ci ne vérifie pas la condition (3) des points de surface, en effet, le point a n'est pas 6-adjacent aux deux composantes 6-connexes de $|\overline{X}|^x$. Il est intéressant de noter que de nombreuses configuration locales qui vérifient les conditions de la définition 9 ne sont en faite pas "extensibles", c'est à dire qu'il existe un point de $N_{26}(x) \cap X$ dont la configuration locale ne pourra en aucun cas vérifier la définition. La figure 2.6(d) est un exemple de cette situation. Ce point vérifie bien les conditions voulues pour une surface de Morgenthaler, par contre, la configuration du point y (figure 2.6(e)) ne pourra en aucun cas les vérifier (notamment à cause du point p qui ne sera jamais 6-adjacent aux deux composantes 6-connexes de $|\overline{X}|^x$). Nous verrons que la configuration de la figure 2.6(d) ne vérifie pas les conditions locales de MA-surface ni de 26-surface forte. Par ailleurs nous verrons aussi qu'une surface de Morgenthaler est un cas particulier de ces dernières. Il n'y a pas de contradiction, ceci est dû au fait que la figure 2.6(d) n'est pas extensible, c'est à dire qu'elle ne peut pas apparaître non plus dans une surface de Morgenthaler.

L'exemple de la figure 2.6(j) montre que cette définition de surface peut être jugée trop restrictive. En effet, les 4 points situés aux coins du carré gris ne vérifient pas la condition (3). Or il est flagrant que ce type de configuration est tout à fait acceptable dans une surface discrète (pensez au sommet d'un cône !).

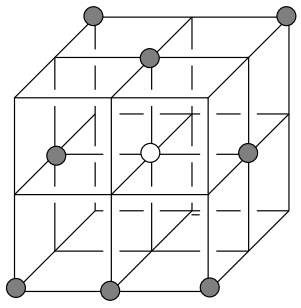


FIG. 2.8 – *Un configuration possible pour un point de 26-surface de Morgenthaler.*

2.7.3 MA-Surfaces

Ce type de surface dues à Rémy Malgouyres[5] utilise une autre approche. Elle fait appel à la notion de *18-quasi courbe fermée*. On doit tout d'abord rappeler la notion de courbe fermée.

Définition 10 *Soit X une partie de E, X et une n-courbe fermée simple si $\forall x \in X$, x et n-adjacent a exactement 2 autres points de X.*

Définition 11 *Soit X une partie de E. Un point $x \in X$ est dit* n-coin *s'il est n-adjacent à exactement 2 points y et z de X tels que y et z soient eux-mêmes n-adjacents. On dit que x est un* n-coin simple *si y et z ne sont pas des n-coins et si x est le seul point à la fois n-adjacent à y et z.*

Définition 12 *Un ensemble X est appelé un* 18-quasi-courbe fermée *si l'ensemble obtenu en enlevant de X tous les n-coins simples est une courbe fermée simple.*
La figure 2.9 (droite) montre un exemple de 18-quasi-courbe fermée dans le 26-voisinage d'un point (partie gauche de la figure).

Intuitivement, une 18-quasi-courbe fermée est une courbe fermée simple à laquelle on a "rajouté" éventuellement des triangles (voir figure 2.9).

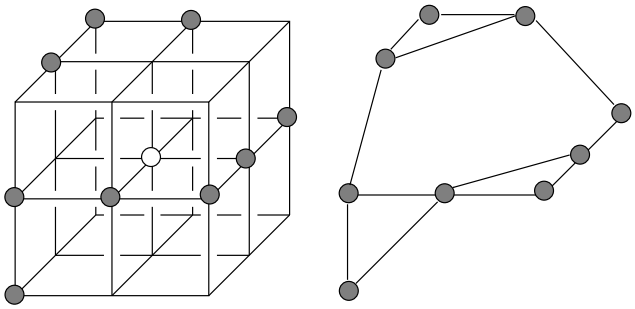


FIG. 2.9 – *Une configuration locale (à gauche), la 18-quasi-courbe fermée associée (représentée par graphe).*

Définition 13 *Un sous-ensemble fini X de E est appelé une* MA-surface, *si X est 18-connexe et que pour tout point $x \in X$, l'ensemble $|X|^x$ est une 18-quasi-courbe fermée.*

La figure 2.9 donne un exemple de configuration vérifiant les conditions de point de MA-surface. Par contre, la figure 2.10 montre le graphe (en 18-adjacence) associé à la configuration de la figure 2.6(d). Il est clair que ces points ne forment pas une quasi courbe fermée telle qu'elle a été définie.

2.7.4 Surfaces fortes

C'est une définition introduite par Gilles Bertrand et Rémy Malgouyres [2] et qui se veut elle aussi plus restrictive que les surfaces primaires. Par contre, elle est plus générale que celle définie par Morgenthaler & Rosenfeld. De même, les 18-MA-surfaces sont elles aussi un sous ensemble de celui des objets qui vérifient la définition des 18-surfaces fortes.

Pour un objet séparant X (cf. définition 8), on appelle *éléments du fond* les composantes connexes de $\overline{X}$, souvent appelées A et B. D'autre part, on appelle *fermeture* d'un élément de fond A l'ensemble

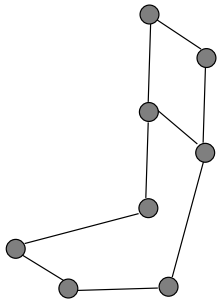


FIG. 2.10 – *Ceci n'est pas une quasi-courbe !*

$A \cup X$. Munis de cette terminologie et des définitions précédentes, la notion de surface forte se décrit précisément de la manière suivante:

Définition 14 *Soit $X \subset E$ un ensemble n-séparant. On dit que X est une* n-surface (fermée) forte *si chaque élément de fond est* fortement sous n-homotope *à sa fermeture.*

Cette idée peut se concevoir sur un exemple concret:
Considérons un ensemble A de points formant une *boule pleine* dans $\mathbb{Z}^3$. Soit X un ensemble construit en "collant" des points sur le bord externe de cette boule de façon à la recouvrir totalement. Alors A forme une des composantes connexes de $\overline{X}$, appelons l'autre (qui est infinie) B. X forme ainsi le bord de l'objet $S = A \cup X$. Maintenant, on dira que l'objet X obtenu est une n-surface forte si A est fortement sous n-homotope à $A \cup X = S$. Clairement, cela signifie que l'on peut enlever dans un ordre quelconque n'importe quel sous ensemble de X (bord de S) sans modifier la topologie de l'objet.

L'intérêt de cette notion est que l'on a presque immédiatement une caractérisation locale de ce type de surfaces puisque la notion d'homotopie forte est elle même caractérisée localement par le théorème 3 (points P-simples) associé au théorème 2 qui relie les notions d'homotopie forte et de point P-simple.

Munis des théorèmes 2 et 3, on en déduit tout de suite la caractérisation *quasi-locale* des points de surfaces fortes. En effet, si A et B forment les deux composantes connexes du fond $\overline{X}$, alors X est une surface forte si A (resp. B) est fortement sous n-homotope à $A \cup X$ (resp. $B \cup X$). D'après le théorème précédent, cela équivaut à dire que X est un ensemble P-simple pour $A \cup X$. En utilisant le théorème 3 pour vérifier que tous les points de l'ensemble X sont P-simples pour $A \cup X$ et aussi pour $B \cup X$; on trouve immédiatement le théorème suivant:

Théorème 5 *Soit $X \subset E$ un ensemble n-séparant, A et B les 2 composantes $\overline{n}$-connexes de $\overline{X}$. X est une n-surface forte si et seulement si :*

$$\begin{cases} T_n(x, |A|^x) = 1 \text{ et } T_n(x, |B|^x) = 1 \\ T_{\overline{n}}(x, |A|^x) = 1 \text{ et } T_{\overline{n}}(x, |B|^x) = 1 \\ \forall y \in N_n^*(x) \cap X,\ T_n(x, |A|^x \cup \{y\}) = 1 \text{ et } T_n(x, |B|^x \cup \{y\}) = 1 \\ \forall y \in N_{\overline{n}}^*(x) \cap X,\ T_{\overline{n}}(x, |A|^x \cup \{y\}) = 1 \text{ et } T_{\overline{n}}(x, |B|^x \cup \{y\}) = 1 \end{cases}$$

La figure 2.6(b) montre une configuration possible pour un point de 28 ou 18-surface forte. Par contre, montrons pour l'exemple que la configuration 2.6(f) ne vérifie pas la 3ème condition du théorème 5 dans le cas de la 26-adjacence. Il apparaît en effet que le point y de la figure 2.11 est

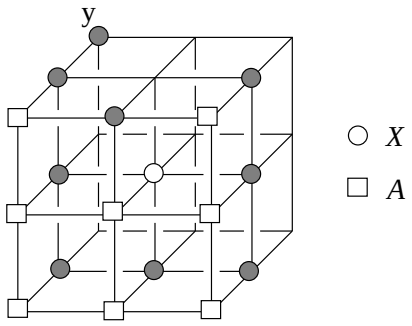


FIG. 2.11 – *Le point y invalide la propriété de point de surface forte pour n=26*

26-adjacent au point central (et pas 18-adjacent). Donc, si on "colorie" $\overline{X}$ de façon à ce que les points carrés soient les éléments de A, alors $G^3_{26}(x, |A|^x \cup \{y\})$ comportera 2 composantes connexes: l'une correspondant au singleton $\{y\}$ et l'autre aux points carrés. D'où $T_{26}(x, |A| \cup \{y\}) = 2$.

Toutefois, il faut préciser que cette caractérisation n'est pas totalement locale puisqu'on a besoin de connaître la répartition des points entre A et B. En effet, rien n'empêche de trouver dans le voisinage de x une composante connexe non $\overline{n}$-adjacente à x et dont on ne peut affirmer a priori qu'elle appartienne à A ou B. Le point y de la figure 2.6(g) en est un exemple. Toutefois, on donne dans [6] une méthode permettant de trouver cette répartition pour les objets 26-séparants. Ceci explique la présence de "?" dans la légende des figures 2.6(g), 2.6(h) et 2.6(i). En effet, ces deux configurations ont la particularité de posséder une composante 6-connexe dans $|X|^x$ qui ne soit pas 6-adjacente au point central. Mais la méthode qui suit ne s'appliquant qu'aux objets 26-séparants, on ne peut l'appliquer dans le cas de la 18-adjacence. La méthode décrite ci-après est illustrée par la figure 2.12.

Tout d'abord, on définit la notion de coloriage pour $\overline{X}$.

Définition 15 *Soit X un objet fortement séparant. La donnée d'un* coloriage *pour X est la donnée, pour tout $x \in X$ d'une application f_x: $N_{26}(x) \cap \overline{X} \longrightarrow \{0, 1\}$, constante sur les composantes 6-connexes de $N_{26}(x) \cap \overline{X}$.*

Définition 16 *Soit $X \subset \mathbb{Z}^3$ un ensemble fortement séparant. On dit que X est* coloriable *si pour tout $x = (i, j, k) \in X$ on a:*

1. $t_{\overline{n}}(x, \overline{X}) = 2$;
2. $\forall C \in C_6(N_{26}(x) \cap \overline{X}) \setminus C^x_6(N_{26}(x) \cap \overline{X})$,
   (a) *C est un singleton: $C = \{M\}$ avec $M = (a, b, c)$;*
   (b) *M n'est pas 18-adjacent à x;*
   (c) *$Z = \{(a, j, k), (i, b, k), (i, j, c)\} \subset \overline{X}$ (figure 2.12).*

Définition 17 *Soit un objet X coloriable, on définit* le coloriage associé à X *de la manière suivante: Pour $x = (i, j, k) \in X$,*

1. *Comme $T_{6+}(x, \overline{X}) = 2$ et X est fortement séparant, alors $Card[C^x_6(N_{26}(x) \cap \overline{X})] = 2$. Soit*

$C_6^x(N_{26}(x) \cap \overline{X}) = \{\chi_1, \chi_2\}$ [2]. *On définit* $f_x(M) = 0$ *si* $M \in \chi_1$ *et* $f_x(M) = 1$ *si* $M \in \chi_2$. *Il reste à définir le coloriage des composantes 6-connexes de* $(N_{26}(x) \cap \overline{X}) \setminus (\chi_1 \cup \chi_2)$.

2. *Soit* $C \in (N_{26}(x) \cap \overline{X}) \setminus (\chi_1 \cup \chi_2)$. *Comme* X *est coloriable,* $C = \{M\} = \{(a, b, c)\}$ *avec* $M \notin N_{18}(x)$. *Soit* $Z = \{(a, j, k), (i, b, k), (i, j, c)\} \subset \overline{X}$. *Le coloriage* f_x *est déjà définit sur* Z. *On définit:*
$\begin{cases} f_x(M) = 1 \text{ si } \chi_1 \cap Z \text{ contient au moins 2 points;} \\ f_x(M) = 0 \text{ sinon.} \end{cases}$

L'unique coloriage $\{f_x; x \in X\}$ *ainsi défini est appelé* le coloriage associé à X. *Soulignons que la construction de ce coloriage associé à* X *est totalement locale.*

Dans le cas de la figure 2.12, on obtiendra $f_x(C) = 0$ car Z contient deux points $p_1 = (i, j, c)$ et $p_2 = (a, j, k)$ tels que $f_x(p_1) = f_x(p_2) = 1$. Muni de ces définitions on dispose alors du théorème

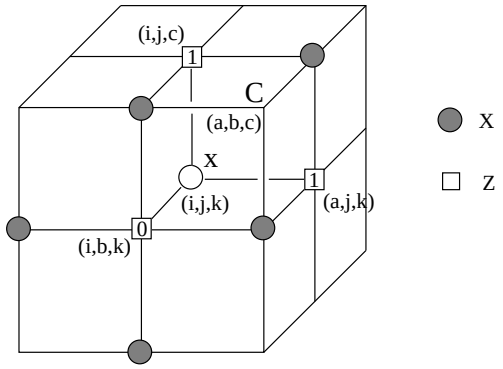


FIG. 2.12 – *Etapes du coloriage*

suivant qui fournit la caractérisation locale complète des surfaces fortes parmi les objets fortement séparants.

Théorème 6 *([6]) Soit* X *un ensemble fortement séparant. Alors* X *est une n-surface forte si et seulement si* X *est coloriable (soit* $\{f_x;\ x \in X\}$ *le coloriage associé à* X*), et que pour tout* $x \in X$ *on a:*

1. $T_n(x, f_x^{-1}(0)) = 1$ *et* $T_n(x, f_x^{-1}(1)) = 1$;
2. $T_{\overline{n}}(x, f_x^{-1}(0)) = 1$ *et* $T_{\overline{n}}(x, f_x^{-1}(1)) = 1$;
3. $\forall y \in N_n(x) \cap X,\ T_n(x, f_x^{-1}(0) \cup \{y\}) = 1$ *et* $T_n(x, f_x^{-1}(1) \cup \{y\}) = 1$;
4. $\forall y \in N_{\overline{n}}(x) \cap X,\ T_{\overline{n}}(x, f_x^{-1}(0) \cup \{y\}) = 1$ *et* $T_{\overline{n}}(x, f_x^{-1}(1) \cup \{y\}) = 1$.

Vient enfin la définition d'un *point de surface forte.*

Définition 18 *Soit* $X \subset \mathbb{Z}^3$ *et* $x \in X$. *On dit que* x *est* un point de n-surface forte *si* $N_{26}(x) \cap X$ *satisfait les propriétés locales du théorème 6.*

2. On rappelle que $C_n^x(\ldots)$ est un ensemble de parties

On a ainsi rendu les conditions du théorème 5 totalement locales si on se place dans le cadre – ce qui est admettons le, une restriction – des objets fortement n-séparants.

Résumons la situation: On dispose du théorème 6 qui dit :

$$X \text{ est une } n\text{-surface forte} \Leftrightarrow \begin{cases} X \text{ est fortement } n\text{-séparant} & (1) \\ \forall x \in X,\ P(n, N_{26}(x)) & (2) \end{cases}$$

Où P est un propriété locale. Or, il a été prouvé que le Théorème de Jordan est vérifié par les ensembles décrits par la condition (2) pour la 26-adjacence. Pour ce faire on a utilisé des surfaces continues (de $\mathbb{R}^3$) qui s'appuient sur de tels objets et pour lesquelles on a appliqué le théorème de Jordan. Ceci permet d'affirmer que dans le cas de la 26-adjacence, (2) $\Rightarrow$ (1), et donc (1) devient inutile. La caractérisation devient de ce fait totalement locale pour les 26-surfaces fortes. Mais il faut savoir que la démonstration évoquée ici ne s'applique pas au cas des 18-surfaces, il s'agit donc d'une partie encore inachevée de la recherche sur les notions de surfaces discrètes...

2.8 Le corps des Quaternions

On verra dans le chapitre 4 qu'il nous a fallu utiliser le corps des quaternions afin de réaliser des rotations dans $\mathbb{R}^3$. Une librairie a été réalisée pour manipuler les éléments de ce corps. Ce qui suit donne les bases nécessaires à la compréhension des opérations décrites dans le chapitre 4.

Théorème 7 *[8] Il existe une algèbre $\mathbb{H}$ de dimension 4 sur $\mathbb{R}$, appelée algèbre des* quaternions, *munie d'une base 1,i,j,k telle que:*

a) L'élément 1 est élément neutre pour la multiplication,

b) on a les formules :
$i^2 = j^2 = k^2 = -1$,
$j.k = -k.j = i$, $k.i = -i.k = j$ *et* $i.j = -j.i = k$

Le corps des nombres réels est isomorphe à la sous-algèbre des quaternions de la forme $a.1$, $a \in \mathbb{R}$, avec laquelle on l'identifie. Une quaternion s'écrit alors:
$q = a + b.i + c.j + d.k$, *avec* $a, b, c, d \in \mathbb{R}$.

Définition 19 *Soit $q \in \mathbb{H}$, $q = a + b.i + c.j + d.k$ avec $a, b, c, d \in \mathbb{R}$. On définit le conjugué $\overline{q}$ de q par : $\overline{q} = a - b.i - c.j - d.k$.*

Remarques

1. On a $\overline{\overline{q}} = q$.
2. $q \in \mathbb{R} \Longleftrightarrow \overline{q} = a$.
3. Soit $P = \{q = b.i + c.j + d.k \mid b, c, d \in \mathbb{R}\}$, on appelle P l'espace des *quaternions purs* et on a : $q \in P \Longleftrightarrow \overline{q} = -q$.

Définition 20 *Soit $q = a + b.i + c.j + d.k \in \mathbb{H}$, on définit la* norme $N(q)$ *de q par :*

$$N(q) = \sqrt{q.\overline{q}} = \sqrt{a^2 + b^2 + c^2 + d^2}.$$

Théorème 8 *On a*

1. *L'algèbre* $\mathbb{H}$ *est un corps (non commutatif).*
2. *Le centre*[3] *de* $\mathbb{H}$ *est* $\mathbb{R}$*.*
3. $N(q_1.q_2) = N(q_1).N(q_2)$*, donc* N*:* $\mathbb{H}^* \longrightarrow \mathbb{R}^{+*}$ *est un homomorphisme de groupes, surjectif, dont le noyau, groupe des quaternions de norme 1, sera noté* G*.*

Remarques

1. Si $q \in G$ alors $q^{-1} = \overline{q}$.
2. Si on identifie $\mathbb{H}$ à $\mathbb{R}^4$ muni de sa topologie naturelle, alors, le groupe G des quaternions de norme 1 s'écrit $G = \{(a, b, c, d) \in \mathbb{R}^4 \mid a^2 + b^2 + c^2 + d^2 = 1\}$. Il est donc homéomorphe à la sphère $\mathbb{S}^3$. En particulier G est connexe.

On définit une opération r_h de $\mathbb{H}^*$ sur $\mathbb{H}$ qui pour tout x de $\mathbb{H}^*$ et tout h de G associe $r_h(x) = h.x.h^{-1}$.

Théorème 9 *La restriction à l'espace des quaternions imaginaires purs de l'opération* r_h *est toujours une rotation de* $\mathbb{R}^3$*. Et toute rotation peut être obtenue ainsi.*

Un fois toutes ces notions théoriques acquises, il a fallu passer au "codage" de la plupart d'entre elles. Le travail à réaliser dans le cadre de ce stage consiste à implanter divers algorithmes de squelettisation. Aussi, ces algorithmes devaient s'appuyer sur une librairie de classes et de fonctions permettant de manipuler les objets présentés ici. Pour des raisons d'efficacité mais aussi d'affinité avec ce langage, elle a été écrite en `C++`, et est en partie (très réduite) décrite dans l'annexe A à laquelle le lecteur curieux pourra se reporter.

3. On rappelle que le centre d'un groupe est le sous-groupe formé des éléments qui commutent avec tous les autres.

Chapitre 3

Algorithmes de Squelettisation

Les algorithmes présentés ici ont été implantés à l'aide des nombreuses fonctions décrites dans l'annexe A. Les fonctions correspondantes sont présentées à la suite de chaque algorithme.

3.1 Algorithme Séquentiel

3.1.1 Version rudimentaire

Cette version n'est donnée ici que pour introduire l'idée générale de l'algorithme définitif. On utilise la notion de point simple qui permet de caractériser de façon locale un point qui peut être effacé sans modifier la topologie (voir chapitre 2).

Répéter
    fin:=VRAI
    Pour tout point x **de** X
        Si Simple(x)
        Alors Effacer(x)
            fin:=FAUX
Jusque fin

Le gros inconvénient de cet algorithme est que le squelette obtenu est fortement dépendant de l'ordre dans lequel les points sont parcourus ("**Pour tout** x **de** X"). En effet, si on teste ces points dans l'ordre lexicographique de leur coordonnées, alors le squelette obtenu sera bien homotope à l'image de départ mais ne sera en rien *représentatif* de cette image. D'une part, si on change l'ordre de parcours, alors le résultat obtenu sera très différent. D'autre part, ce squelette est *décentré* par rapport à l'objet de départ, ce qui n'est pas une bonne caractéristique pour un squelette !

La figure 3.1 montre le squelette (b) obtenu à partir d'un objet assez épais en 2 dimensions (a), ceci pour illustrer le principe qui est le même en 3D. L'algorithme précédent se comporte comme un "rayon" qui balaye l'image et supprime à chaque passe un ensemble de points simples (le test est effectué en chaque point avant effacement).

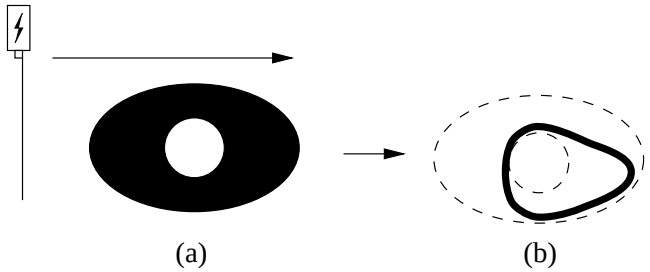


FIG. 3.1 – *Un squelette décentré !*

3.1.2 Version convenable

Pour améliorer cela, on reprend une idée de [9] qui propose de distinguer en 2D quatre familles de points: les points Nord, Sud, Est et Ouest. Ce sont des points du bord externe de l'objet (ceux qui ont un 4-voisin dans le complémentaire). A savoir qu'un point est dit "Nord" si son voisin situé au dessus de lui (au sens de la 2ème coordonnée) est dans le complémentaire. Muni de cette classification, on effectue à chaque étape un balayage pour chaque orientation. Dans le cas 3D, on a simplement ajouté les orientations Haut et Bas pour la troisième coordonnée.

Autre amélioration de taille, on ne parcourt ni la totalité de la matrice (pour ne traiter en fait que les points $\neq 0$), ni la totalité des points non nuls. En effet, un point simple est obligatoirement $\overline{n}$-adjacent avec le complémentaire de l'objet. Sinon, son effacement créerait un trou dans l'objet ! Par conséquent, on construit la liste des points du bord de l'objet, et c'est sur cette liste de points que le parcours s'effectue. Durant chaque passe, on reconstruit la liste en enlevant bien sur les points effacés et en ajoutant les points "découverts" par les suppressions.
Dans le cas 3D cela donne l'algorithme de la figure 3.2. La librairie d'outils dédiés à la squelettisation se voit complétée d'une fonction `Sq_Sequentiel` dont la pseudo en-tête est:
`Sq_Séquentiel(Objet3D, Adjacence, Type_Arret, Nombre_de_passes_max)`
Où `Type_Arret` est une constante qui précise la condition définissant les points terminaux à utiliser comme par exemple `POINT_SURFACE_FORTE`. Le nombre de passes effectuées peut être limité par le dernier paramètre.

3.2 Algorithme Parallèle

L'algorithme séquentiel présenté précédemment fonctionne de manière satisfaisante. Toutefois, il est possible d'améliorer encore plus la qualité des squelettes obtenus en utilisant la notion de points P-simples. Celle-ci présente deux avantages:

1. L'algorithme séquentiel supprime des points, mais la suppression d'un point modifie souvent le caractère de simplicité de ses voisins. Ainsi, selon l'ordre dans lequel il est visité, un point donné ne sera peut être plus simple alors qu'il l'aurait été avec un autre ordre d'apparition. C'est ce problème que tente de prévenir la méthode directionnelle: éviter de modifier le caractère de simplicité des voisins. Il est en effet vrai qu'examinant les points orientés dans une même direction, on diminue ce risque d'inter-dépendance.
   La notion de point P-simple répond totalement à ce problème. Un tel point reste simple quelque

Algorithme séquentiel directionnel

Soit M un tableau à 3 dimensions contenant l'objet.
B:={(x,d) / x est point du bord, marqué à 2 et d'orientation d}
L : Liste de points (le *nouveau* bord)
Répéter
fin:=VRAI
L:=∅
Pour tout (x,d) **de** B, mettre à jour d **Fin pour**
Pour dir **dans** (N,S,E,O,H,B)
Pour (x,orientation) de B
Si (M(x)==2 **et** (orientation&dir) **et** Simple(x))
Alors Effacer(x)
Pour tout x' 6-voisin de x
Si (M(x')==1) **Alors** M(x'):=2, L:=L∪{x}
Fin Pour
fin:=FAUX
Sinon Si (M(x)==2) **Alors** L:=L∪{X}
Fin Pour
Fin Pour
B:=L
Jusque fin

FIG. 3.2 – *Algorithme Séquentiel*

soient les points de $P \cap N_{26}^{*}(x)$ qu'on efface. Reste à définir cet ensemble P. La chose est simple, on travaille comme pour l'algorithme séquentiel –et pour la même raison– sur le bord de l'image. On va donc tester si les points de ce bord peuvent être supprimés quelque soient les points déjà effacés. Donc l'ensemble P à considérer n'est autre que le bord de l'image.

2. L'algorithme obtenu est dit parallèle dans le sens où on peut effectuer de manière simultanée les tests de P-simplicité pour tous les points du bord de l'objet. On peut donc répartir ce travail sur différentes machines. La suppression des points "acceptés" se fait de manière globale une fois les tests effectués pour une liste de bord donnée.

On aboutit alors à l'algorithme de la figure 3.3.

Cet algorithme s'arrête lorsqu'il n'y a plus de points Bord-simples dans l'image. Le résultat obtenu n'est donc pas le vrai squelette minimum que l'on peut trouver. On peut en effet appliquer sur le résultat l'*algorithme séquentiel* qui trouvera lui encore des points simples (et pas P-simples) et les effacera.
On obtient alors ce qu'on appelle un *noyau homotopique* de l'image de départ. Ce type de squelette (voir 3.3.1) présente l'inconvénient de transformer une composante connexe sans trous ni cavité en un point unique. De même, un objet connexe ne comportant qu'un trou aura pour squelette une courbe fermée simple. Aussi, on cherche à améliorer l'algorithme pour que le résultat soit plus représentatif de la géométrie des objets. Cela est réalisé en ajoutant des conditions qui figent certains points: on

Algorithme parallèle

Soit M un tableau à 3 dimensions contenant l'objet.
B:={x / x est point du bord, marqué à 2}
L : liste de points, (ceux qui sont à supprimer)
B2 : liste de points, (le nouveau bord)

```
Répéter
        fin:=VRAI
        L:=∅
        B2:=∅
        Pour tout x de B
                Si (x est B-simple)
                   Alors L:=L∪{x}
                   Sinon B2:=B2∪{x}
        Fin Pour
        Pour tout x de L
                fin:=FAUX
                M(x):=0
                Pour tout x' 6-voisin de x
                        Si (M(x')==1)
                        Alors B2:=B2∪{x'}
                              M(x'):=2
                Fin Pour
        Fin Pour
        B:=B2
Jusque fin
```

FIG. 3.3 – *Algorithme Parallèle I*

Algorithme parallèle II

Soit M un tableau à 3 dimensions contenant l'objet.
B:={x / x est point du bord, marqué à 2}
L : liste de points, ceux qui sont à supprimer.
B2 : liste de points, le nouveau bord

```
Répéter
        fin:=VRAI
        L:=∅
        B2:=∅
        Marquer les points terminaux de B à 3
        Pour tout x de B
                Si ( M(x)≠3 et x est B-simple)
                   Alors L:=L∪{x}
                   Sinon B2:=B2∪{x}
        Fin Pour
        Pour tout x de L
                fin:=FAUX
                M(x):=0
                Pour tout x' 6-voisin de x
                        Si (M(x')==1)
                        Alors B2:=B2∪{x'}
                              M(x'):=2
                Fin Pour
        Fin Pour
        B:=B2
Jusque fin
```

Appliquer l'algorithme séquentiel directionnel en marquant et conservant de la même manière les points terminaux.

FIG. 3.4 – *Algorithme Parallèle II*

impose que certains types de points rencontrés restent dans le squelette. On appelle ces points des *points terminaux*.

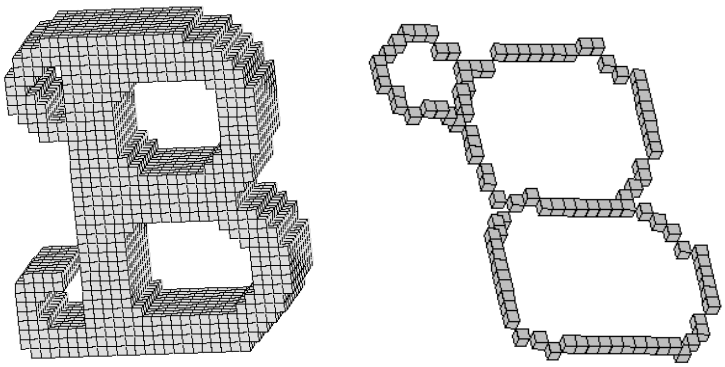
Conserver par exemple les points extrémités (ceux qui n'ont qu'un voisin dans l'objet) permet de garder des branches. Alors que les points de surfaces (on en possède plusieurs définitions, voir chapitre 2) permettent de conserver les parties "étalées" d'un objet. Plus précisément, on conservera aussi les voisins de points de surface, ainsi, toute une configuration au voisinage d'un point de surface est figée. En utilisant et en combinant ces conditions terminales, on peut obtenir différents types de squelettes, avec des degrés de fidélité différents (voir chapitre 3.3).

L'algorithme de squelettisation définitif est donné par la figure 3.4.

La fonction correspondante à cet algorithme est `Sq_Parallele` dont l'en-tête est:

`Sq_Parallele(Objet3D, Adjacence, Type_Arret, Nombre_de_passes_max)`

Avec les mêmes options que pour le cas séquentiel.



(a) (b)

FIG. 3.5 – *Un objet (a) et un noyau homotopique possible (b).*

3.3 Quatre types de squelettes

3.3.1 Noyau Homotopique

Définition 21 *Soit X un sous ensemble de $\mathbb{Z}^3$, alors tout ensemble $Y \subset X$ est appelé* noyau homotopique *de X s'il vérifie:*

i) Y est sous homotope à X, et

ii) $\forall x \in Y$, x n'est pas n-simple pour Y.

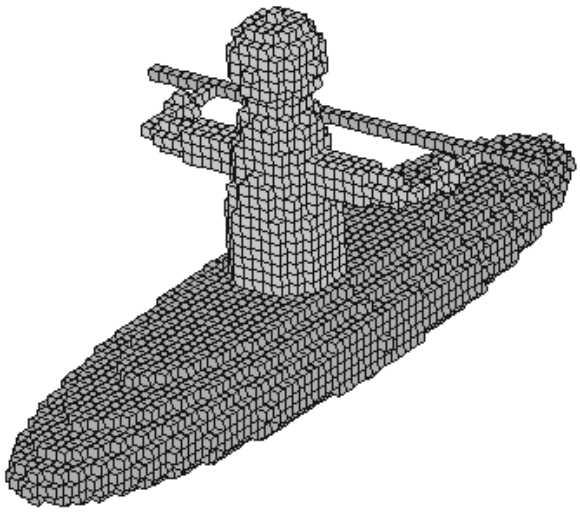


FIG. 3.6 – *Un kayakiste "discret"*

3.3.2 Squelette filaire

Ce type de squelette (figure 3.7) est obtenu en posant comme points terminaux tous les points de l'objet qui n'ont qu'un voisin. Ainsi, tout au long de l'algorithme, tous les points terminaux qui

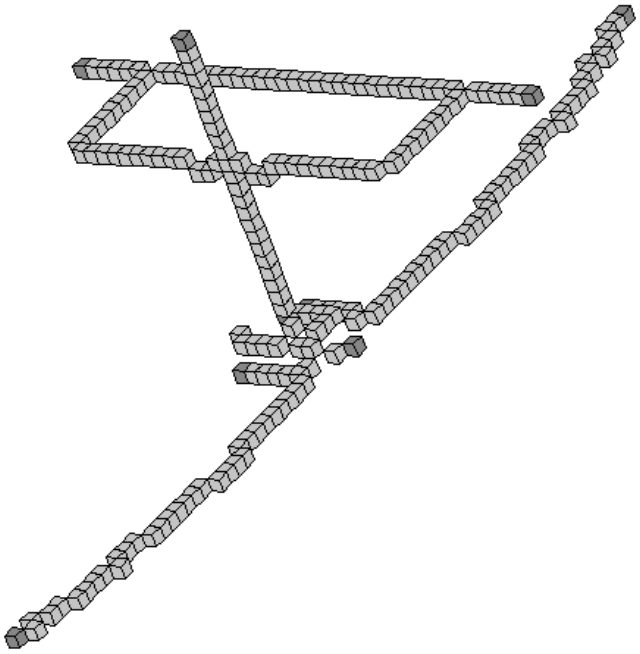


FIG. 3.7 – *Un squelette filaire de kayakiste*

apparaissent sont marqués à la valeur 3 (valeur choisie arbitrairement). Cela garantit que ces points ne seront plus effacés.

On remarque que c'est bien le fait de "figer" les points extrémités (en foncé) qui impose la conservation des branches. Il devient impossible de "couper" ces branches ou de les diminuer car on déconnecte alors l'image.

3.3.3 Squelette Surfacique

Dans ce cas, on oblige la conservation des points dont le voisinage répond aux conditions prévues pas l'une ou l'autre des notions de surface. Les points voisins de points de surface sont eux aussi figés. Selon la définition choisie, le résultat sera différent.

3.3.4 Combinaison

Afin de conserver à la fois les parties aplaties et les branches d'une image, on pourra combiner les deux en s'arrêtant sur les points de surface et sur les extrémités. On obtiendra par exemple une image telle que celle présentée en figure 3.9.

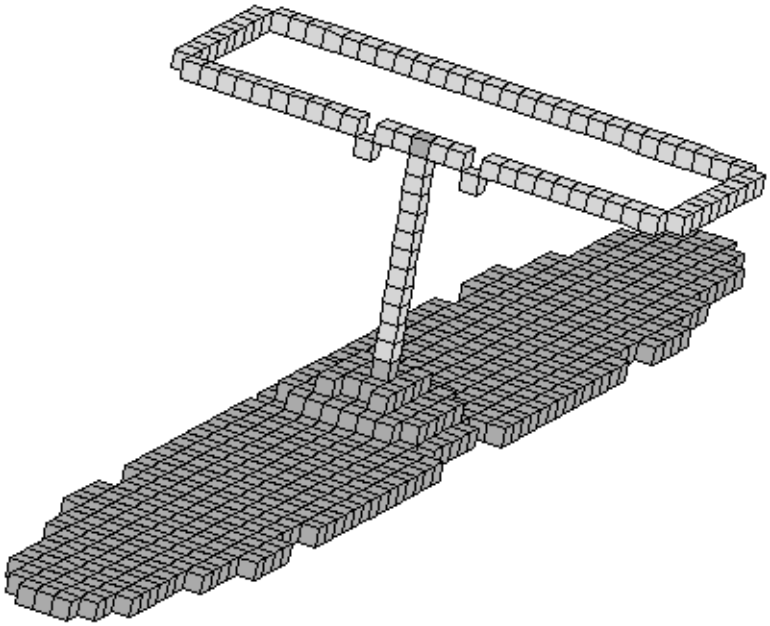


FIG. 3.8 – *Un squelette surfacique avec arrêt sur les voisins des points de surface forte.*

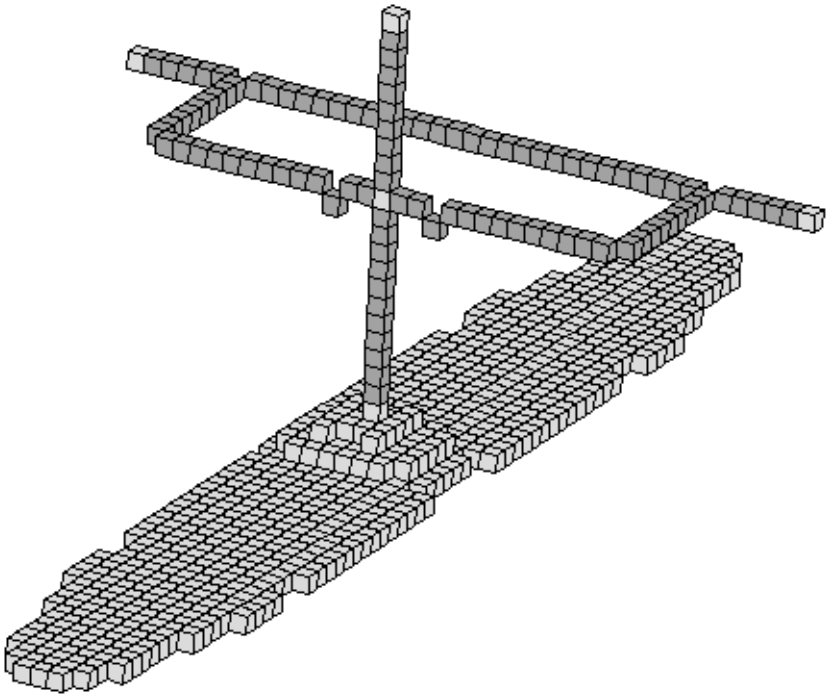


FIG. 3.9 – *Un squelette surfacique pour lequel on a aussi conservé les points extrémités*

Chapitre 4

Application à l'imagerie médicale

Ce chapitre présente une application de la notion de surface forte et des algorithmes de squelettisation présentés ici dans le cadre du recalage d'images acquises par I.R.M.

4.1 Position du problème

Le problème posé, très largement étudié au sein du groupe "Cerveau" du G.R.E.Y.C. est le suivant:
On dispose de deux images 3D du cerveau d'une même personne mais acquises à des époques différentes. Aussi, les deux images ne se superposent (sauf heureux hasard) pas parfaitement car les conditions d'acquisition ne sont bien sûr jamais totalement identiques.
Or, il est primordial de pourvoir les superposer précisément afin de comparer par exemple une zone particulière d'une image à l'autre. Dans d'autres cas, ce sont des images issues de méthodes d'acquisitions de natures différentes que l'on souhaite superposer. On parle alors d'images issues de plusieurs *modalités*.

Aussi, on s'intéresse au problème suivant:

Soient deux images $Im_1([x,y,z])$ et $Im_2([x,y,z])$, trouver une translation $\tau_{x,y,z}$ ainsi qu'une rotation ρ de $\mathbb{R}^3$ telles que:
Im_1 et Rotationé(Translaté(Im_2,τ);ρ) se recouvrent au mieux.

4.2 Algorithme

L'algorithme se base sur la géométrie de l'objet formé de la substance qui entoure l'encéphale: le *liquide céphalo-rachidien* (L.C.R.). Cette substance apparaît avec une couleur gris claire dans les images I.R.M. Dans le laboratoire du G.R.E.Y.C, Su Ruan a mis au point une méthode de segmentation permettant d'extraire d'une image cet objet. L'intérêt de cette matière est qu'elle offre une image "négative" fidèle des sillons de l'encéphale. Du fait de sa forme, c'est un objet représentatif de la position du cerveau. Mais surtout: ses propriétés géométriques, et plus particulièrement celles de son squelette, permettent de comparer à moindre frais des rotations entre les deux images.

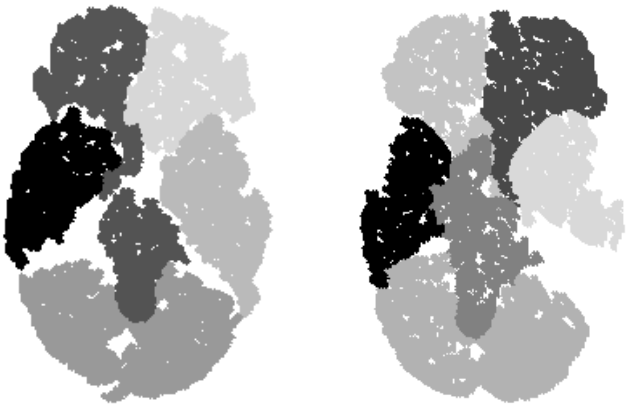
Les étapes de cette méthode sont les suivantes: Soient IM1 et IM2 les deux images de cerveaux.

1. On construit le bord externe[1] de chaque image ($\Rightarrow$ BORD1 et BORD2)

1. C'est l'ensemble des points du complémentaire de l'objet qui sont 6-adjacents à un point de cet objet

2. On (Su Ruan) extrait le L.C.R. de chacune des deux images (⇒ LCR1 et LCR2).
3. On réunit les images LCRi avec les BORDi.
4. Eventuellement, si l'image obtenue est trop bruitée, on applique une dilatation morphologique (en 26-adjacence) (⇒ DIL1 et DIL2).
5. On applique l'algorithme de squelettisation (figure 3.4) avec comme points terminaux les points de 18-surfaces fortes (⇒ SQ1 et SQ2).
6. On ne conserve que les points de 18-surfaces fortes présents dans SQ1 et SQ2 (⇒ REG1 et REG2). Ceci a pour effet de supprimer des objets SQ les points qui correspondent aux intersections entre des surfaces que l'on veux séparer (déconnecter entre elles).
7. On ne conserve ensuite que les 7 plus grosses régions en nombre de points (⇒ 7REG1 et 7REG2).
8. On supprime dans 7REG1 et 7REG2 la composante 18-connexe dont le centre de gravité a la seconde coordonnée la plus faible (⇒ 6REG1 et 6REG2). En effet, il apparaît que les 6 régions inférieures obtenues sont stables, invariantes d'une image à l'autre. Ce n'est pas le cas de la région supérieure, c'est pourquoi cette dernière n'est pas conservée.
9. Pour chaque région de 6REG1 et 6REG2, on calcule son aire approchée ainsi que les coordonnées de son centre de gravité. (⇒ $\{(A1_i, G1_i), i \in [1,6]\}$ et $\{(A2_i, G2_i), i \in [1,6]\}$; $A_i \in \mathbb{R}$ et $G_i \in \mathbb{R}^3$).

Il se trouve que les régions obtenues à l'avant dernière étape sont stables. On a ainsi obtenu les régions de la figure 4.1 pour deux images différentes.



(a) (b)

FIG. 4.1 – *Les régions significatives obtenues au point 9*

Maintenant, on cherche la rotation à opérer sur l'une des images pour la superposer à l'autre. Pour cela, on commence par recentrer les deux images sur leurs centres de gravité. Les listes de centres de gravité des régions sont elles aussi mises à jour afin que les origines des repères utilisés coïncident.

On définit ensuite pour $i \in \{1,2,3,4,5,6\}$ un unique $j(i)$ de ce même ensemble telle que la quantité $(A1_i - A2_{j(i)})^2 + \|G1_i - \rho(G2_{j(i)})\|^2$ soit minimale. Ce $j(i)$ est tout simplement trouvé par un parcours des deux listes et le calcul de cette quantité pour chaque j possible.

Ensuite, pour toute rotation ρ de $\mathbb{R}^3$, on considère la fonction suivante:

$$f(\rho) = \sum_{i=1}^{6} (A1_i - A2_{j(i)})^2 + \|G1_i - \rho(G2_{j(i)})\|^2$$

.

Cette fonction mesure la "distance" qui sépare les listes de centres de gravité et d'aires. Notre but est alors de minimiser cette fonction. La méthode utilisée consiste à suivre les courbes de gradient. On travaille pour plus de précision dans le corps des quaternions qui permet d'effectuer des rotations sur $\mathbb{R}^3$. Plus précisément: pour h un quaternion de module 1 on considère l'application suivante:

r_h: $\mathbb{H} \longrightarrow \mathbb{H}$
$x \longmapsto h.x.h^{-1}$

La restriction à l'espace des imaginaires pures de cette action de h par automorphismes intérieurs sur $\mathbb{H}$ est toujours une rotation de $\mathbb{R}^3$ (qui s'identifie à l'espace des quaternions imaginaires purs). Toute rotation peut être obtenue ainsi.

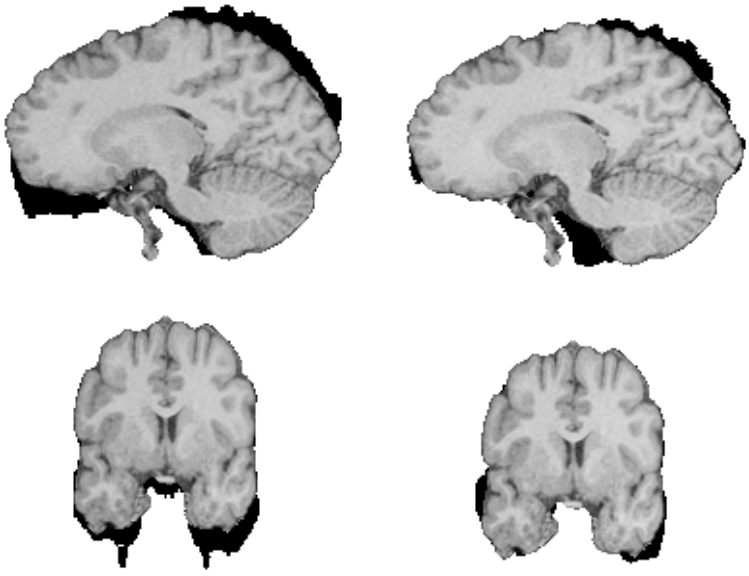
L'avantage de considérer les quaternions est qu'il est plus facile de générer aléatoirement un quaternion proche de 1 et, qu'il est plus facile de s'assurer que le quaternion construit, approximant sur machine un produit de quaternions proches de l'identité, reste dans l'espace considéré des quaternions de norme 1.

L'algorithme de minimisation se résume de la manière suivante:

Algorithme *Minimise*(f)
//Retourne un quaternion h tel que $f(h)$ soit minimal
h : quaternion initialisé à l'identité
q, ρ, ρ_{min} : quaternions
valeur=0, minimum=∞ : des flottants
μ : un flottant tel que $0 < \mu < 1$
ε : un flottant initialisé à ε_0
Alea(e) : fonction qui retourne un quaternion q de norme 1
tiré au hasard tel que $\|1 - q\| < e$
Tant que $\varepsilon > \varepsilon_{fin}$ **Faire**
Répéter NOMBRE_TESTS **fois**
q = Alea(ε)
ρ = h.q
valeur=$f(\rho)$
Si valeur < minimum
Alors minimum=valeur, $\rho_{min} = \rho$
Fin Répéter
h=ρ_{min}
$\varepsilon = \varepsilon * \mu$
Fin Tant que
Résultat : h

Une fois le quaternion h trouvé, on peut alors superposer les deux images en opérant une translation (correspondante à la différence des centres de gravité) puis la rotation de chaque point de la deuxième image. La figure 4.2 montre une coupe sagitale[2] et transversale de la réunion des deux images; avant et après recalage par l'algorithme présenté ici. Les zones en noir correspondent à la "différence" des deux images.

4.3 Résultats



(a) Avant correction (b) Après correction

FIG. 4.2 – *2 coupes de la superposition des 2 images avant et après correction.*

4.4 Structures & Méthodes complémentaires

Afin d'implanter cette méthode, il a fallu compléter la librairie décrite dans l'annexe A. La principale nouveauté est la classe `GrapheRegion`, qui est destinée à construire puis contenir sous forme de graphe une composante connexe quelconque. Ensuite, on doit calculer le centre de gravité associé mais surtout une approximation de son aire.

Une nouvelle structure de liste a aussi été écrite pour accueillir les centres de gravité ainsi que les aire calculées: `ListeParams`. Vu le grand nombre de régions qu'on peut trouver dans les images produites au cours de l'algorithme de recalage, on a préféré effectuer là encore une allocation en un seul bloc continu. Chaque élément de cette liste contient donc un flottant (l'aire de la région) et un vecteur de 3 flottants représentant les coordonnées du centre de gravité de la région. Cette liste constitue l'équivalent pratique des valeurs $A1_i$,$A2_i$,$G1_i$ et $G2_i$ qui apparaissent dans la formule de la fonction à minimiser.

2. correspondante au plan de symétrie du cerveau

Enfin, pour une précision accrue, on a choisi d'effectuer les rotations en utilisant le corps des quaternions qui permettent d'effectuer en série un grand nombre de rotations proches de l'identité sans perte de précision.

Méthodes de la classe `GrapheRegion`

`Construit(objet3d,x,y,z)` est la méthode importante de cette classe, elle parcourt en largeur (à l'aide d'une file) l'image en partant du point donné en paramètre. A la sortie de la fonction, le graphe correspondant à la composante connexe dont le point fait partie est construit.

`Ajoute(x,y,z)` Ajoute un sommet et met à jour les arêtes du graphe.

`Aire()` Calcule et retourne un flottant, l'aire approchée de la surface de la région. Cette aire est calculée comme la somme des aires associées à chaque *surfel*[3]. Pour chaque face d'un *voxel*[4] en contact avec le vide, on effectue le produit scalaire entre la normale du surfel et la normale du plan tangent à la surface de l'objet. Ces plans tangents sont calculés par la méthode des moindres carrés sur un voisinage du point en question. Remarquons au passage qu'une méthode `GenererPlan` a été ajoutée à la classe des `Objet3D` afin de pouvoir visualiser les plans ainsi calculés. On a pu ainsi s'assurer de leur cohérence.

`CentreDeGravite(&x,&y,&z)`

`MarquerObjet(&objet3d, valeur)` Marque tous les points de la région dans l'image `objet3d` avec la valeur donnée. Ceci peut notament servir à effacer une région (avec la valeur 0).

La classe des `Quaternions` Le corps des quaternions est composé d'éléments de $\mathbb{R}^4$, (a, b, c, d) souvent notés $q = a + b.i + c.j + d.k$ puisqu'il existe dans ce corps trois *imaginaires pures* i, j et k tels que $i^2 = j^2 = k^2 = -1$ (voir 2.8). On utilise ce corps pour effectuer des rotations dans $\mathbb{R}^3$, il a donc fallu définir une telle classe ainsi que toutes les fonctions permettant d'effectuer les opérations usuelles.

Ensuite il a aussi fallu écrire l'algorithme de minimisation de la fonction décrite dans la section précédente.

3. "surfel" est la compression pour "élement de surface"
4. "voxel" est la compression pour "élément de volume"

Conclusion

L'application trouvée aux algorithmes de squelettisation qui ont été implémentés au cours de ce stage montre que ce travail n'était pas dénué d'intérêt pratique. Le travail bibliographique d'une part, l'écriture d'outils en `C++` ajouté au travail de "tâtonnement" expérimental pour mettre au point les algorithmes ont permis de remplir l'objectif d'un tel stage: une initiation à la recherche.

J'ai notament pu étudier les notions d'homotopie, d'homotopie forte ainsi que les différentes définitions de surfaces discrètes. Le seule rédaction de ce rapport a aussi permis de "revoir" de façon encore plus approfondie la théorie assimilée assez rapidement en début de stage. L'état de "manque" vis à vis de la programmation et de résultats "palpables" de ces théories m'avait en effet poussé à commencer suffisamment tôt l'implantation des algorithmes qui ont été présentés ici. La programmation était aussi motivée par l'intention de tester ces algorithmes et de mettre au point la méthode de recalage d'images de cerveau. Cette méthode dont les résultats sont prometteurs montre une application concrète du travail effectué dans la cadre des activités de recherche du G.R.E.Y.C. / Image en imagerie médicale. Toutes les applications possibles de ces algorithmes d'amincissement n'ont toutefois pas été explorées.

Concernant les objectifs initiaux, précisons que la mise à l'épreuve de l'isotropie des algorithmes de squelettisation n'a pu être menée par manque de temps. Cela peut sembler incohérent au vue de l'application trouvée mais en fait cette application, justement basée sur la rotation d'images 3D, constitue en quelque sorte une "preuve de fait" de cette propriété d'isotropie. Tout de même, quelques tests visuels rapides ont permis de s'assurer de la bonne ressemblance entre deux squelettes issus d'images identiques à une rotation près. Il faudrait pour être complet mettre au point un méthode fiable d'évaluation de la ressemblance entre deux squelettes. De telles méthodes existent, il reste à les implanter.

Je n'ai pas traité dans ce rapport de la complexité des algorithmes décrits ni des temps d'éxécution sur machine. Précisons simplement que la squelettisation du L.C.R. dont il est question dans le chapitre 4 dure en moyenne entre 60 et 75 minutes pour des objets d'environ 150.000 points (image contenue dans un volume de $256 \times 256 \times 124$ voxels). Ces temps ont été observés avec l'implantation faite sur des stations de travail Sun^{tm} de type Ultra Sparc (bi-processeurs).

D'autre part, j'espère que ce rapport pourra servir de base à qui désirera prendre connaissance de l'état des recherches dans le domaine de la squelettisation dans $\mathbb{Z}^3$ mais aussi servir d'introduction aux notions de topologie et de géométrie discrète nécessaires (chapitre 2) pour aborder le domaine de la squelettisation (chapitre 3).

Bibliographie

[1] G. Bertrand. *On P-simple points.* Number 321. C.R. Acadmie des Sciences, 1995.

[2] G. Bertrand and R. Malgouyres. Topological properties of discrete surfaces. *Lecture notes in computer sciences*, pages 325–336, 1996.

[3] S. Durand. *Algorithmes de squelettisation dans le domaine discret en 2D et en 3D, Application sur des images R.M.N. 3D afin d'obtenir le squelette des sillons du cerveau humain.* G.R.E.Y.C./I.S.M.R.A., 1995. Rapport de stage de D.E.A.

[4] J.M. Chassery et A. Montanvert. *Gomtrie discrte en analyse d'images.* Hermes, 1991.

[5] R. Malgouyres. A definition of surfaces of $\mathbb{Z}^3$. *Theorical Computer Science*, 186:1–41, 1997.

[6] R. Malgouyres. Local characterization of strong surfaces within strongly separating objects. *Pattern Recognition Letters*, 1997. A paraitre.

[7] D. G. Morgenthaler and A. Rosenfeld. *Surfaces in three-dimensional images*, volume 51. 1981.

[8] D. Perrin. *Cours d'Algbre.* Ellipses, 1996.

[9] R. Stefanelli and A. Rosenfeld. Some parallel thinning algorithms for digital pictures. *Journal of ACM*, 18:255–264, 1971.

Annexe A

Librairie de base

Cette annexe décrit de façon rapide la librairie `C++` qui a été développée au cours de ce stage et dont l'objet est d'implanter les notions théoriques présentées au chapitre 2. Grâce à cela, les algorithmes exposés au chapitre 3 ont pu être traduits sans grande difficulté et bénéficient, je l'espère, d'une grande lisibilité. Je préfère ne décrire que brièvement les principales classes et fonctions de cette librairie plutôt que d'en donner brutalement l'ensemble des fichiers d'en-tête.

A.1 Structures de données

A.1.1 Les objets 3D

Une classe `Objet3D` a été écrite pour manipuler des tableaux de dimension 3 contenant des entiers. Les images issues par exemple de procédés d'acquisition type I.R.M fournissent une "carte" $f : E \subset \mathbb{Z}^3 \longrightarrow [0 - 255]$. Les valeurs trouvées en chaque point donnent une information utile sur la nature de l'objet acquis en ce point. Dans notre cas, on a simplement besoin de l'information de présence ou d'absence de matière. Toutefois, pour des raisons propres aux algorithmes qui vont suivre, il sera utile de distinguer plusieurs catégories de points aux sein d'une même image (pour des systèmes de marques apposées en certains points) . Aussi, on ne se contentera pas d'un *bitmap* de l'objet mais toujours d'une matrice d'entiers [0-255].

Cette matrice est allouée dynamiquement. Dans la mesure du possible, on tente l'allocation d'un seul et unique "bloc mémoire" puis de tableaux de pointeurs permettant d'accéder rapidement aux plans puis aux "axes". L'intérêt de cette méthode est de permettre un accès plus rapide aux éléments de la matrice. En effet, une méthode plus simple consistait à n'allouer qu'un seul tableau d'octets contigus, la position d'une case est alors calculée par multiplication : $Tableau[x][y][z] = Espace[x + taille_x * (y + z * taille_y)]$. Or, compte tenu du nombre très important d'accès effectués à ce tableau pendant l'exécution des programmes, il est très important d'optimiser au maximum cette structure. C'est ce qu'offre celle qui est présentée ici puisque l'on remplace deux produits par deux sommes entières (sur des pointeurs), bien plus simples et rapides pour une machine.

Formats des fichiers de données Les fichiers utilisés représentent des images en 3 dimensions. Pour contenir ces images, une classe `Objet3D` a été construite autour d'un structure de type matrice m∗m∗t. Cette classe bénéficie de deux méthodes principales de lecture/écriture:

– `Objet3D.Lire(`*`nom de fichier`*`)`

– `Objet3D.Ecrire(`*`nom de fichier`*`)`

Ces fonctions servent d'interfaces avec 6 autres qui manipulent 3 types de fichiers différents (selon l'extension du nom de fichier).

Les fichiers .3d C'est un format de fichier propre à la librairie. Il a été conçu pour permettre une création rapide *à la main* de petits fichiers représentant un objet dans $\mathbb{Z}^3$. Il comporte un courte en-tête donnant la taille de la matrice de points à suivre. Puis, de manière lisible, une succession de matrices 2D précédées chacune d'un numéro de plan (correspondant à la 3ème coordonnée):

```
OBJET3D
TAILLE=5,5,5
PLAN 1
0 0 0 0 0
0 1 1 1 0
0 0 1 0 0
0 1 1 1 0
0 0 0 0 0
PLAN 2
...
```

Les fichiers .vff C'est un format de fichier standard très utilisé au sein du laboratoire du G.R.E.Y.C. et notament celui qu'est capable de lire l'outil *Surftool* d'Alexandre Lenoir, très largement utilisé pour visualiser les images des nombreux tests effectués au cours de l'écriture des programmes. Ce format est composé d'une en-tête (parfois très longue) [1] en clair donnant de nombreux renseignements sur le contenu du fichier. Suit ensuite un flot continu d'octets correspondant à un *niveau de gris* de l'image en chaque point.

Les fichiers .3dz C'est un format vff "amélioré" pour l'occasion afin de résoudre le gros problème de ces fichiers: leur taille. En effet, contrairement aux nombreux formats d'images 2D qui font appel à des méthodes de compression, le format vff utilise 1 octet par point de l'espace. Comme les images destinées à être squelettisées sont binaires (pas de niveaux de gris), un principe R.L.E. (Run Length Encoding) a été utilisé pour compresser le flot d'octets. Cette méthode simple permet tout de même de produire des fichiers de quelques kilo-octets pour des objets qui occupent plus de 8Mo (256x256x124) dans leur format vff.

A.1.2 Les listes de points

Dans tous les algorithmes implémentés, il sera souvent plus efficace de parcourir des listes de points plutôt que de reconsidérer la totalité de la matrice à chaque nouvelle recherche. En effet, seuls les points dont la valeur est différente de 0 nous intéressent et cela suffit largement à justifier ce choix. On a donc écrit une classe `GrosseListe` qui permet de stocker un nombre suffisamment grand de coordonnées de points. La structure de cette liste est décrite par la figure A.1. Plutôt que d'utiliser une liste chaînée, on a préféré allouer 256 tableaux au fur et à mesure du remplissage de la liste. La

1. L'auteur apporte cette précision simplement afin d'évoquer un "bogue" dû à une mauvaise estimation de la taille maximale de cette en-tête. Les erreurs induites furent très coûteuses en temps et neurones humains.

taille de chaque tableau, fixée au moment de la création de la liste, permet en pratique de s'adapter à toutes les tailles possibles. Cela tout en évitant de lancer une multitude de *petites allocations* pour construire une liste chaînée classique. Ce type d'allocation rend en effet la tâche de gestion de mémoire très lourde pour le système et participe à l'allongement des temps d'exécution.

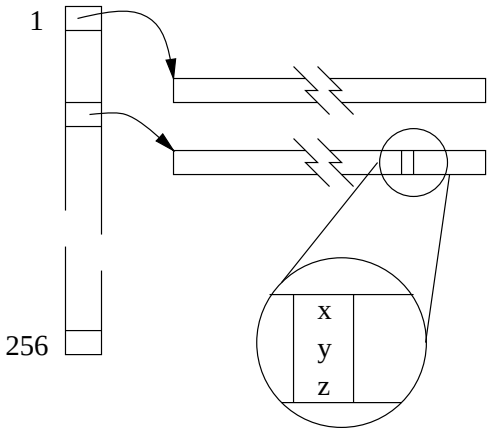


FIG. A.1 – *Aperçus de la structure* `GrosseListe`

A.1.3 Les configurations locales

A la lecture des préliminaires théoriques, il apparaît que les méthodes à implanter sont toutes basées sur l'examen du voisinage de chaque point. On doit considérer pour chaque coordonnée le 26-voisinage qui forme ce qu'on appelle une *configuration locale*. Dans l'examen des configurations, il est primordial de pouvoir accéder de façon instantanée aux voisins de n'importe quel point. Bien sûr, cela nous amène à munir ces configurations d'une structure de graphe, donnant lieu à la classe `Configuration`. On pose d'abord les conventions d'étiquetage des différents points du 26-voisinage comme le montre la figure A.2(a); les points sont repérés par l'ordre lexicographique de leur coordonnées. Ensuite, on représentera une configuration par son graphe donné par liste de successeurs. La figure A.2(b) montre le graphe associé en 18-adjacence alors que la figure A.2(c) montre la même configuration vue du point de vue de la 26-adjacence. Un tableau, indicé par les étiquettes des points, donne pour chacun la liste des numéros de points voisins si le champ "présence" a la valeur 1. Ces listes sont des tableaux de taille maximale fixe (=27) tous localisés dans un unique bloc mémoire, comme le montre la figure A.3.

A.2 Méthodes

A.2.1 Méthodes de la classe `Objet3D`

`Lire("fichier[.vff|.3d|.3dz]")` et `Ecrire("fichier[.vff|.3d|.3dz]")` pour la gestion des fichiers de données.

`SetValeur(x,y,z,val)` et `GetValeur(x,y,z)` pour accéder aux valeurs des points par leurs coordonnées.

`GetConfig(x,y,z)` renvoie une `Configuration` locale correspondant au point demandé.

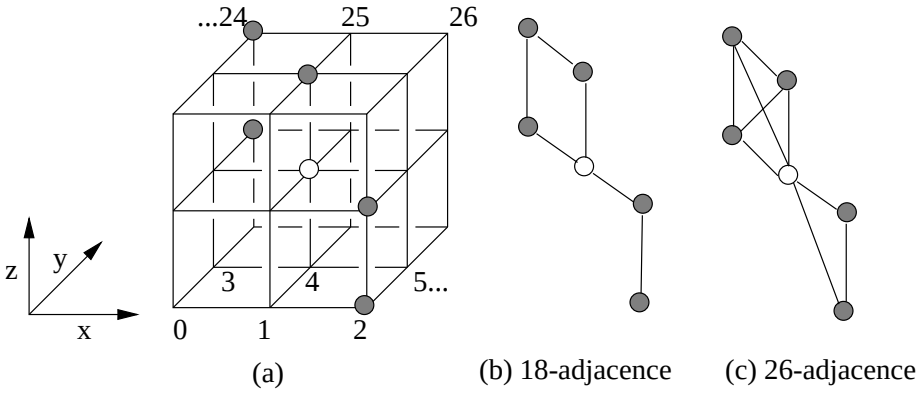


FIG. A.2 – *Configuration locale et graphes associés*

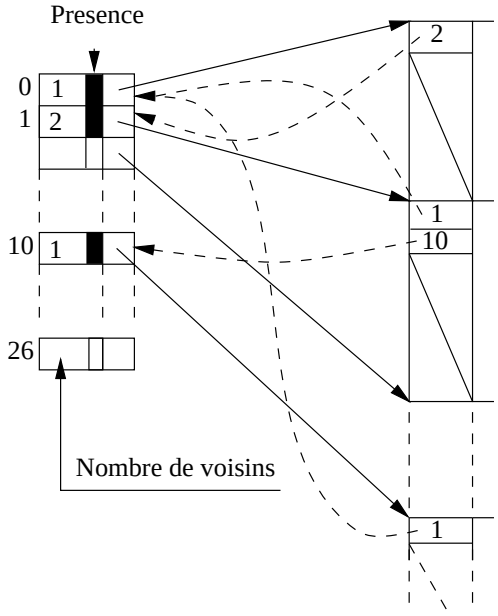


FIG. A.3 – *Structure interne des graphes de configuration locale*

`MarquerBord(GrosseListe &)` Marque tous les points du bord de l'objet à la valeur 2 et construit une liste (de type `GrosseListe`) de ces points. Un point est dit *point de bord* s'il est 6-voisin d'un point du complémentaire de l'image (point dont la valeur est nulle).

`NombreDePoints()` Renvoie le nombre de points de l'image.

`Union(Objet3D &)` Effectue la réunion avec une autre image donnée en paramètre.

`Orientation(x,y,z)` Renvoie un octet dont les bits indiquent l'orientation d'un point de l'image (notion expliquée en 3.1), en bref, les bits 1 à 6 sont utilisés (éventuellement en même temps) pour les directions Nord, Sud, Est, Ouest, Haut et Bas.

A.2.2 Méthodes de la classe `GrosseListe`

`GrosseListe([n])` est le constructeur, si n est précisé, alors les allocations en cours de remplissage se font par tableaux de `[n]` octets.

`Raz()` vide la liste.

`Ajoute(x,y,z[,orientation])` Sans commentaire.

`Debut()` Place le pointeur de parcours en début de liste.

`Suivant(&x,&y,&z [, &orientation ])` Place dans x, y et z les coordonnées du point courant et avance le pointeur de parcours.

`Fin()` Est vrai si le pointeur de parcours a dépassé la fin de la liste (il vaut alors NULL).

A.2.3 Méthodes de la classe `Configuration`

On rappelle que les points d'une configuration locale sont numérotés de 0 à 26. Les fonctions suivantes utilisent ce numéro pour les désigner.

`Configuration([Adjacence])` Constructeur qui a pour paramètre le type d'adjacence à considérer pour la configuration. Des constantes sont définies: `ADJ6, ADJ6P, ADJ18, ADJ26`. Notez que ce paramètre est un champ de cette classe, à tout moment, une configuration est associée à un type d'adjacence bien précis. C'est à ce niveau que se joue la flexibilité de tous les algorithmes. En effet, pour travailler dans l'une des adjacences disponibles, il suffit de manipuler des configurations locales du type désiré. La fonction charnière qui teste l'adjacence de deux points est décrite plus loin (A.3).

`SetAdjacence(Adjacence)` Permet de modifier l'adjacence considérée et de remettre à jour le graphe de la configuration (crée ou enlève des arêtes).

`Ajoute(n) / Retire(n)` un point de la configuration (et donc du graphe).

`Point(n)` Teste la présence d'un point.

`Voisins(n)` Renvoie une `ListePoints` (petite structure de données capable de contenir un petit nombre de numéros de points), avec les numéros des points voisins du point n.

`Affiche()` Effectue un semblant de sortie graphique sur un terminal texte pour afficher un cube 3x3x3 avec les points de la configuration. La liste des points est aussi affichée avec pour chacun la liste de ses voisins.

`AffichGr()` Dessine dans une fenêtre X11 un cube 3x3x3 avec les points de la configuration, figure semblable à celle déjà apparues dans ce document.

`Set_Marque(n, marque)` `Test_Marque(n, marque)` permet de marquer chaque point avec une combinaison de 16 marques différentes.

`Marquer_Connexes()` Pour chaque point de la configuration, un champ "numéro de composante connexe (ncc)" est prévu. Cette fonction lance un parcours du graphe afin d'en trouver les composantes connexes qui seront numérotées. Les champs `ncc` sont mis à jour et le nombre de composantes trouvées est renvoyé.

`UneCC()` Renvoie vrai si le nombre de composantes connexe est 1. C'est une fonction nécessaire car optimisée par rapport à `Marquer_Connexe()` puisqu'elle échoue dès qu'une 2ème composante connexe est mise en évidence, ou si aucune n'est trouvée.

`Label(&a, &b)` Place dans a et b le résultat de l'assignation trouvée (voir les préliminaires théoriques 2.7.4) correspondant à la configuration.

`Extremité()` Renvoie vrai si le point n'a qu'un seul voisin (définition d'un point extrémité).

A.3 Fonctions

La plupart des fonctions présentées ici ont pour but de permettre d'écrire de façon "naturelle" d'autres fonctions. Celles ci permettront de tester si les configurations locales extraites d'un image vérifient telle ou telle caractérisation donnée dans les préliminaires théoriques.
Elle sont données ici dans un ordre non trivial car les dernières font appel à celles qui les précèdent, allant des outils de base vers les outils les plus abstraits.

`Fonction Adj(adj,n,m)` Cette fonction est utilisée de manière très intensive par toutes les méthodes liées aux configurations. Elle a pour but de "coder" les relations de voisinages décrites dans les préliminaires. Elle renvoie la valeur logique `VRAI` si les deux points passés en paramètres sont `adj`-voisins (rappel: un point est en fait un nombre compris entre 0 et 26 (figure A.2).

Au départ, cette fonction était programmée en utilisant les différences de coordonnées à la manière des définitions (2.1.1). Mais il est très vite apparu nécessaire de l'optimiser (en examinant le temps passé par les programmes à l'intérieur de cette fonction, de l'ordre de 45% du temps CPU). La méthode choisie a été de construire par programme 3 tableaux 26x26 (un par type d'adjacence 8,18 et 26), qui câble cette relation pour tous les couples possibles de points. Ceci fait, le tableau a été intégré au code source de la fonction qui devient ainsi un simple accès à un élément de tableau.

`Fonction Complémentaire(configuration)` Construit et retourne une configuration dans laquelle seuls les points absents de la configuration donnée en paramètre sont mis à 1.

`V_Geodesique(configuration[, ordre])` Retourne une configuration correspondant au voisinage géodésique (définition 3) du centre de la configuration donnée. Si le paramètre ordre n'est pas précisé, alors le voisinage géodésique usuel est calculé.

`Nombre_Topologique(configuration)` Retourne un entier.

`NbTopoEgaleUn(configuration)` Retourne `VRAI` si le nombre de composantes connexes du voisinage géodésique usuel est 1. Ceci afin d'optimiser ce type de test (fréquent). En effet, de même qu'il existe une méthode `UneCC()` dans la classe configuration, il est plus rapide de tester l'absence de 2 composantes connexes que de les compter toutes pour ensuite s'apercevoir qu'il y en a 4 !

`Point_Simple(configuration)`.

`Point_PSimple(configuration)` Suppose que les points de la configuration sont différenciés par

une marque (les point de "P" doivent être distingués des autres).

`Point_PSimple_Optimisé(configuration)` Utilise une optimisation possible (voir lemme 4 pour les conditions (3) et (4) du théorème 3 qui permet de ne calculer qu'un seul voisinage géodésique pour ensuite effectuer un test de simple voisinage pour tous les y ($\forall y \in N_{26}^{*} \cap \dots$).

`Point_Surface(configuration)`.

`Point_Surface_Forte(configuration)`.

`Point_Extremite(configuration)`.

`Point_Surface_Forte_Ou_Extremite(configuration)`

`Label(configuration, &a , &b)` Trouve le coloriage de `configuration` qui est une assignation et place $|A|^x$ dans `a`, $|B|^x$ dans `b`.

A.4 Utilitaires

Cette section décrit divers utilitaires écrit en `C++`, basés pour certain sur la librairie "minimum" décrite précédemment, et qui ont été très largement utilisés dans le cadre de ce travail.
La plupart acceptent comme fichier d'entrée ou de sortie les trois formats décrits plus haut: `vff`, `3d` et `3dz`. Tous acceptent (quand cela a un sens) un paramètre optionnel pour préciser le type d'adjacence dans laquelle on souhaite travailler.
Par exemple: `select` *`canoe.vff sf_canoe.3dz -ADJ18 -sf`*
marquera à 10 (valeur par défaut, modifiable bien sûr) tous les points de 18-surfaces fortes contenus dans le fichier *canoe.vff*, le résultat de l'opération sera le fichier *sf_canoe.3dz*.

`bord` *`fichier_entree fichier_sortie`* Extrait le bord d'une image.

`calotte` *`fichier_entree fichier_sortie`* génère le bord externe (celui qui l'envelopperait) d'une image 3D.

`calc_cdg` *`fichier`* Calcule le centre de masse de l'objet et place ses coordonnées dans un champ de l'en-tête du fichier.

`convert` *`entree sortie`* permet la conversion entre les 3 formats de fichiers.

`cutvff` *`entree sortie ...`* Extrait une partie rectangulaire de l'objet. Les coordonnées du sous-espace sont saisies au clavier.

`dilate` *`entree sortie`* `[n]` Effectue au choix n 6-dilatations ou n 26-dilatations de l'objet. Une 6-dilatation consiste à placer dans le résultat tous les points de la source avec leurs 6-voisins. De même pour la 26-dilatation. Ceci est souvent utilisé pour supprimer de façon "brutale" le bruit présent dans une image.

`reunion` *`image1 image2 sortie`* Sans commentaire, si ce n'est qu'une option permet de forcer le mode "monochrome".

`select` *`entrée sortie`* `[...]` Outil très utile qui permet de marquer les points qui vérifient une des caractérisations locales suivantes, dans l'une des 4 adjacences possibles:

- Points simples
- Points Bords-simples
- Points de surface primaire

– Points de surfaces fortes

– Points de bord

Cet outil a aussi été complété pour extraire et sélectionner les régions manipulées dans le cadre de l'application (chapitre 4).
Enfin, l'un des programmes principaux écrit dans le cadre de ce travail est le programme `sqc` (pour **sq**uelettisation **c**ombinée[2]) qui correspond à l'implantation des algorithmes présentés au chapitre 3. A titre d'illustration, voici l'aide de la ligne de commande du programme `sqc`.

```
uranus% sqc
Usage:  sqc fichier_entree fichier_sortie
            [-sp|-sf] [-par[N]] [-seq[N]]  [-a26|-a18|-a6P|-a6]]

 Effectue une squeletisation surfacique
   -sf  : [DEFAUT] Arret sur les points de surface forte
   -sfe : Arret sur les points de surface forte ou extremites
   -ext : Arret sur les points extremites
   -sp  : Arret sur les points de surface primaires
   -nonstop  : pas de test d'arret
   -par[N] : squeletisation en parallele, N passes
   -seq[N] : squeletisation sequentielle, N passes
   -ordreNSEOHB | NOBSEH ... : fixe l'ordre de la sq. sequentielle
   -tempo : genere un fichier entre les deux passes

 Phase I  (par): Supprime les points P-Simples sauf les points de surface
 Phase II (seq): Erode de maniere sequentielle N-S-E-O-H-B les points simples.
uranus%
```

2. Combinée car au départ existaient deux outils différents: `sqseq` et `sqpar`, respectivement pour les algorithmes séquentiels et parallèles. Ensuite, par des options, il est devenu possible de réaliser l'un et/ou l'autre via un même programme.

Annexe B

Plan de l'exposé

Ce stage donnant lieu à une soutenance orale, le plan de cet exposé est donné ici.

Introduction

1 Préliminaires Théoriques

- 1.1 Configurations locales, voisinages.
- 1.2 Voisinage géodésique, groupe fondamental.
- 1.3 Points simples et points P-simples.
- 1.4 Notions de surfaces discrètes.

2 Algorithmes de squelettisation sans conditions terminales

- 2.1 Algorithme séquentiel.
- 2.2 Algorithme Parallèle.
- 2.3 Exemple de noyau homotopique.

3 Algorithmes avec conditions terminales

- 3.1 Trois types de squelettes.
  - 3.1.1 Noyau homotopique.
  - 3.1.2 Squelette filaire.
  - 3.1.3 Squelette surfacique.
- 3.2 Algorithme parallèle.
- 3.3 Algorithme séquentiel.
- 3.4 Complémentarité : Parallèle puis séquentiel.

4 Application à l'imagerie médicale: recalage d'images

- 4.1 Position du problème.
- 4.2 Algorithme.
- 4.3 Résultats.

Conclusion