Définition et étude d'un modèle topologique minimal de représentation d'images 2d et 3d

Guillaume Damiand

ACADÉMIE DE MONTPELLIER

UNIVERSITÉ MONTPELLIER II

— SCIENCES ET TECHNIQUES DU LANGUEDOC —

Thèse

présentée à l'Université des Sciences et Techniques du Languedoc
pour obtenir le diplôme de DOCTORAT

Spécialité	: **Informatique**
Formation Doctorale	: **Informatique**
École Doctorale	: **Information, Structures et Systèmes**

Définition et étude d'un modèle topologique minimal de représentation d'images 2d et 3d

par

Guillaume DAMIAND

Soutenue le 14 Décembre 2001 devant le Jury composé de :

M. Jean-Claude Bajard, Professeur, Univ. Montpellier II, LIRMM ..... Président du Jury

M. Jean-Pierre Braquelaire, Professeur, Univ. Bordeaux I, LABRI ..... Rapporteur
M. Walter Kropatsch, Professeur, Technische Universität Wien, PRIP ..... Rapporteur
Mme Annick Montanvert, Professeur, Univ. Grenoble, ENSIEG-LIS ..... Rapporteur

M. Jean Françon, Professeur, Univ. L. Pasteur Strasbourg, LSIIT ..... Examinateur

M. Yves Bertrand, Professeur, Univ. Poitiers, IRCOM-SIC ..... Co-directeur de Thèse
M. Christophe Fiorio, Maître de Conférences, Univ. Montpellier II, LIRMM ..... Co-directeur de Thèse

M. Michel Habib, Professeur, Univ. Montpellier II, LIRMM ..... Directeur de Thèse

REMERCIEMENTS

Tout d'abord je voudrais remercier Jean-Pierre Braquelaire, Walter Kropatsch et Annick Montanvert d'avoir accepté de relire cette thèse et d'en être rapporteurs. La version finale de ce mémoire a bénéficié de leur lecture très attentive et de leurs remarques précieuses. Je tiens à remercier Jean-Claude Bajard d'avoir accepté d'être président du jury. Je remercie également tous les membres du jury d'avoir accepté d'assister à la présentation de ce travail, particulièrement Jean-Françon qui s'est déplacé depuis Strasbourg.

Je voudrais remercier tout particulièrement Yves Bertrand qui m'a dirigé tout au long de ces trois années de thèse. Il a toujours été disponible, à l'écoute de mes nombreuses questions, et s'est toujours intéressé à l'avancée de mes travaux. Les nombreuses discussions que nous avons eues ainsi que ses conseils sont pour beaucoup dans le résultat final de ce travail. Sa capacité d'analyse et son enthousiasme m'ont montré que le monde de la recherche pouvait être un univers passionnant. Enfin, ses nombreuses relectures et corrections de cette thèse ont été très appréciables. Cette thèse lui doit beaucoup. Pour tout cela merci.

Je remercie Christophe Fiorio d'avoir co-encadré ce travail de thèse. Il m'a tout d'abord permis d'intégrer l'équipe image en me proposant un sujet très intéressant et m'a laissé la liberté de le re-orienter au cours du déroulement de ma thèse. C'est également grâce à sa collaboration avec le laboratoire SIC que j'ai eu la chance de travailler avec deux équipes de recherche, ce qui c'est avéré une expérience très enrichissante. Au cours de ce travail, il a souvent attiré mon attention sur certains problèmes de conception et m'a consacré beaucoup de temps, entre-autres pour s'occuper de l'organisation du jury de thèse.

Je remercie Pascal Lienhardt avec qui j'ai eu la chance de pouvoir travailler. Sa rigueur, sa capacité d'analyse des problèmes et ses très nombreuses connaissances m'ont permis de progresser et ont répondu à plusieurs de mes préoccupations.

Je remercie tous les thésards de Montpellier pour la bonne ambiance de travail mais également pour les nombreux bons moments passés ensembles. Entre autres Alexis, Éric, Jean-Marc, les Olivier(s), Stephan…

Je remercie également tous les thésards et les autres membres du laboratoire SIC de Poitiers, notamment ceux avec qui j'ai eu l'occasion de travailler et les autres simplement pour les bons moments partagés.

Je voudrais remercier particulièrement Françoise et Sylvie pour leur sympathie et leur efficacité dans l'organisation et la résolution des problèmes administratifs, mais aussi pour les bons moments lors des pauses café.

Enfin, je remercie également toutes les personnes qui m'ont permis d'oublier momentanément le travail dans des soirées, repas, sorties vélo ou autres : Damien, Emma, Éric, Élo, Fab, Françoise, Ludo, Myriam, Niko, Sandrine, Seb, Valérie…

Je tiens à remercier tout particulièrement Vak, Yo et Théo pour leur nombreuses relectures et corrections de mon anglais désastreux. Ils ont toujours répondu présent, même dans les cas les plus désespérés. Merci.

Je remercie également Nathalie qui a pris le temps de relire une grande partie de ce mémoire, et qui s'est arraché les cheveux pour essayer de comprendre le sens souvent tortueux de mes phrases.

Je remercie les stagiaires avec qui j'ai eu la chance de travailler. L'ambiance de travail à été excellente et nous avons pu ainsi allier bons moments et scéances de travail : Cyrius qui a su mettre l'expression *optimal* à la mode ; Fred qui a fait un travail exceptionnel avec Moka et sans qui cette thèse aurait sûrement été beaucoup moins illustrée. J'espère pouvoir continuer à travailler souvent avec lui, pourquoi pas au cours d'une thèse ; Pascal qui a su être patient pour m'expliquer les subtilités des méthodes de Markov ; Patrick avec qui nous avons effectué de nombreuses séances de travail permettant d'éclaircir la partie minimale des cartes topologiques. Sa vision très imagée et enthousiaste nous a permis de travailler toujours dans la bonne humeur.

Enfin, je remercie Christelle qui a su me soutenir, me supporter, m'encourager… pendant toute la durée de ma thèse et plus particulièrement durant les derniers mois de rédaction qui n'ont pas toujours été des plus agréables. Cette thèse et moi te devons beaucoup. Merci.

TABLE DES MATIÈRES

CHAPITRE 1

INTRODUCTION

La segmentation d'images en régions est le point de départ incontournable de tout processus d'analyse d'images. C'est un problème difficile qui est l'objet de nombreux travaux de recherche. Ces travaux portent depuis plusieurs années sur les images en dimension deux. L'avènement d'outils d'acquisition en dimension trois et leur utilisation de plus en plus répandue, principalement dans le domaine de l'imagerie médicale, mais également en géologie ou dans l'industrie, fait apparaître un besoin croissant de segmentation 3d. La segmentation consiste à regrouper les pixels (ou voxels en dimension trois) en régions vérifiant certains critères d'homogénéité. Ces régions doivent représenter les objets contenus dans l'image. La difficulté de la segmentation provient de la qualité des images à segmenter qui varie suivant les techniques d'acquisitions, des problèmes de recouvrement lorsqu'un objet est partiellement caché par un autre, mais surtout de l'absence de critère permettant de juger de la qualité d'une segmentation.

Un autre problème est apparu avec l'avènement des premiers outils de segmentation tri-dimensionnel. La taille des données à traiter est très importante pour cette dimension, ce qui pose un problème d'efficacité des algorithmes de segmentation. Ce problème est moins important en dimension deux, où de ce fait la question de complexité des algorithmes de segmentation est secondaire. À titre d'exemple, une image de taille moyenne en dimension deux comporte 256×256 pixels, ce qui correspond à environ 200 kilo-octets, alors qu'une image de taille moyenne en dimension trois est composée de $256 \times 256 \times 256$ voxels, ce qui correspond à 48 méga-octets. Un algorithme de segmentation ayant une complexité importante pourra être utilisé en dimension deux, mais pas en dimension trois.

Un autre point important pour définir un « bon » algorithme de segmentation a trait aux critères qu'il utilise. La plupart du temps, la seule information utilisée est le niveau de gris ou la couleur des pixels, ce qui est assez limitatif. Il semble en effet naturel d'utiliser plusieurs critères différents, suivant la nature des images et des objets contenus dans ces images. Nous pouvons envisager d'utiliser des critères sur la forme des objets, sur leurs positions relatives, sur le nombre d'objets adjacents… Afin de pouvoir utiliser ces critères, il est nécessaire de représenter et de structurer les informations contenues dans l'image, et notamment les informations d'adjacence, d'incidence, d'inclusion. En fait, coder ces informations revient à rendre compte de la topologie de l'image. De manière intuitive, la topologie décrit les objets de manière « structurelle » : leurs faces, arêtes et sommets, ainsi que la manière dont ces éléments sont positionnés les uns par rapport aux autres. La géométrie décrit la forme de ces objets. Nous nous sommes vite rendu compte du rôle crucial

que la topologie pouvait jouer dans des processus de représentation, d'analyse ou de modification. En effet, il est très difficile (et inefficace) de déduire les propriétés topologiques d'un objet à partir de sa seule géométrie. Or, ces propriétés topologiques permettent de vérifier l'exactitude d'un objet lors de sa construction et d'aider à sa caractérisation, donc à sa reconnaissance. De plus, les informations topologiques structurent les informations géométriques et permettent une manipulation efficace de ces dernières.

Ces considérations sont au cœur de ce travail de thèse. La question est de déterminer comment structurer les informations d'une image pour permettre la définition d'algorithmes de segmentation efficaces. Cette question se pose pour une raison supplémentaire. La plupart des applications nécessitent en effet, après la phase de segmentation, une phase interactive ou semi-interactive au cours de laquelle un expert valide le résultat, et le modifie s'il ne le juge pas satisfaisant. La définition d'une structure de données adéquate permet de répondre également à ce besoin. C'est donc dans cette double optique que nous avons cherché à définir un modèle de représentation d'images 3d segmentées en régions. Remarquons que ce cadre d'utilisation n'est pas limitatif. Il est en effet facile de représenter les images non segmentées en étiquetant chaque composante connexe de pixels de même couleur par un identifiant unique.

Dans un premier temps, nous nous sommes intéressé aux solutions existantes en dimension deux afin d'étudier les possibilités d'extension en dimension supérieure. Deux d'entre elles ont particulièrement retenu notre attention : le graphe topologique des frontières et les cartes discrètes. Ces deux modèles permettent de représenter les images 2d, et utilisent pour cela les cartes combinatoires. Cela nous a amené à nous intéresser à ce modèle combinatoire, et plus largement aux modèles de représentation par les bords (ou B-rep). Ces modèles sont nombreux et utilisés dans le monde de la modélisation géométrique. Ils ont pour avantage de séparer l'aspect topologique de l'aspect géométrique. Cela permet de simplifier les traitements définis sur ce type de modèle. En effet, les opérations de consultation ou modification peuvent se classer en trois catégories distinctes : celles utilisant exclusivement la topologie, celles utilisant exclusivement la géométrie, et celles utilisant les deux. Séparer le modèle topologique du modèle géométrique facilite ces traitements, y compris lorsque les deux modèles sont utilisés.

Un autre avantage des modèles de représentation par les bords, et donc des cartes combinatoires qui en font partie, est que le nombre d'objets à représenter est beaucoup moins important que pour les autres modèles. En effet, intuitivement lorsque nous représentons tous les pixels d'une image, il faut représenter n^2 éléments (où n est la largeur et hauteur de l'image), alors qu'il faut seulement représenter de l'ordre de n informations pour représenter le bord de ces objets. En effet, au lieu de représenter une information de surface (de dimension deux), nous représentons des courbes (de dimension un).

Ces réflexions nous ont convaincu des nombreux avantages des cartes combinatoires et nous ont décidé à les utiliser pour traiter notre problématique. Mais les deux modèles existant en dimension deux basés sur ces cartes combinatoires se sont avérés difficilement extensibles en dimension supérieure. En effet, tout deux se basent sur des algorithmes d'extraction assez techniques. Les définitions des modèles associés sont en partie incomplètes et s'appuient principalement sur ces algorithmes. Nous sommes donc revenu à la dimension deux afin de définir ou redéfinir un modèle permettant de représenter les images 2d utilisant les cartes combinatoires, mais en ayant comme préoccupation principale qu'il soit facilement extensible en dimension supérieure.

Nous avons ainsi défini la *carte topologique 2d* qui est un modèle minimal de représentation d'images 2d segmentées en régions. Le fait que ce modèle soit minimal est important, car il permet d'envisager son extension en dimension supérieure malgré l'augmentation importante de la quantité d'information. De plus, cela permet également des traitements plus efficaces, de par le nombre d'éléments moins élevé à traiter. Ce modèle est équivalent aux deux modèles dont nous nous sommes inspiré, le graphe topologique des frontières et les cartes discrètes. La principale différence avec ces derniers, et l'apport de notre travail, se situe dans la manière dont ce modèle est défini. Nous avons en effet introduit une notion de niveaux de simplification permettant de définir la carte topologique de manière progressive. Chaque niveau se définit simplement à partir du précédent par application d'un type particulier de fusion. Cette définition apporte plusieurs avantages. Tout d'abord elle facilite l'extension en dimension supérieure, ce qui était notre préoccupation principale. De plus, le fait de découper cette définition en plusieurs étapes la simplifie, tout comme sont simplifiées les preuves et l'étude des cas pouvant poser problème. En effet, chaque passage entre deux niveaux consécutifs est simple, étant donné que chaque modification est restreinte. Par exemple l'étude du problème de déconnexion se trouve facilitée, car il suffit de vérifier si chaque passage entre deux niveaux successifs peut conduire à ce type de problème.

Nous avons ensuite étudié les algorithmes permettant d'extraire ce modèle à partir d'une image. Notre définition progressive fournit directement un premier algorithme qui est l'application directe des définitions de chaque niveau de simplification. Puis nous avons défini un deuxième algorithme permettant d'extraire la carte topologique de manière optimale, c'est-à-dire en une seule passe sur l'image et avec un nombre minimal d'opérations. Nous avons pour cela repris le principe de l'algorithme de Christophe Fiorio utilisant la notion de précode, en l'adaptant à notre définition progressive et aux différents niveaux de simplification. Ces niveaux facilitent cette étude étant donné que pour un niveau particulier, il suffit de trouver les nouveaux cas à traiter par rapport au niveau précédent. Cela permet de factoriser les cas pouvant se traiter de manière similaire.

Nous avons ensuite étendu cette définition en dimension trois afin d'obtenir la *carte topologique 3d* qui était notre objectif initial. Cette extension est possible grâce aux différents niveaux de simplification. Les points délicats liés à cette définition sont les problèmes de déconnexion. Leur traitement est primordial afin d'obtenir la représentation minimale. Puis nous avons étendu en 3d les deux algorithmes d'extraction de la carte topologique 2d. Cela ne pose pas de problème pour le premier algorithme, mais le deuxième nécessite un très grand nombre de cas à traiter : 4140. Nous avons pu factoriser un grand nombre de ces cas à l'aide des niveaux de simplification et obtenir finalement seulement 379 cas différents. De plus, nous avons introduit une notion de précode isomorphe par rotation qui nous permet de ramener ce nombre de cas à 129. Même si ce nombre est encore élevé, il est finalement assez faible en regard des 4140 existant au total.

Nous avons donc apporté une solution à notre problématique initiale, la définition d'un modèle permettant de structurer les informations contenues dans une image, en conservant l'efficacité comme préoccupation centrale. Pour l'occupation en espace mémoire, nous proposons la carte topologique qui est la représentation minimale, et pour la complexité en temps nous définissons l'algorithme optimal d'extraction basé sur les précodes. Ces deux problèmes de complexité sont liés. En effet, les algorithmes de traitement seront moins coûteux pour la carte topologique étant donné que le nombre d'éléments à considérer est moins important.

Le plan de cette thèse est le suivant. Le chapitre 2 présente les cartes combinatoires qui sont à la base de ce travail, ainsi que les cartes généralisées qui sont une extension des cartes combina-

toires, et que nous utilisons également. Au chapitre 3, nous étudions quelques modèles existant en dimension deux, principalement les deux qui sont à l'origine de la carte topologique : les cartes discrètes, et le graphe topologique des frontières. Mais nous présentons également quelques autres solutions. Le chapitre 4 présente la carte topologique en dimension deux. Après avoir fixé le cadre de travail, nous définissons les différents niveaux de simplification. Nous présentons ensuite les deux algorithmes d'extraction, et exposons pour le deuxième les cas à traiter pour chaque niveau de simplification. Puis, au chapitre 5, nous étendons cette définition en dimension supérieure. Nous montrons comment résoudre les problèmes de déconnexion, et surtout comment utiliser ces solutions afin de garantir la minimalité de la carte topologique. Nous présentons ensuite les algorithmes d'extraction, en montrant pour l'algorithme optimal comment factoriser les cas se traitant de manière similaire. Le chapitre 6 revient sur l'opération de fusion et montre comment les caractéristiques topologiques d'un objet évoluent suivant ces fusions. Cela nous permet de prouver que chaque niveau intermédiaire de simplification représente les mêmes informations. De plus, cela montre que nous avons traité tous les cas particuliers posant éventuellement problème lors de la définition de la carte topologique. Le chapitre 7 présente quelques travaux auxquels nous avons participé, et directement liés à ce travail de thèse. Certains de ces travaux utilisent et donc valident notre modèle, alors que d'autres ouvrent de nouvelles perspectives de recherche. Enfin le chapitre 8 conclut la thèse, et propose des perspectives.

CHAPITRE 2

PRÉSENTATION DES CARTES

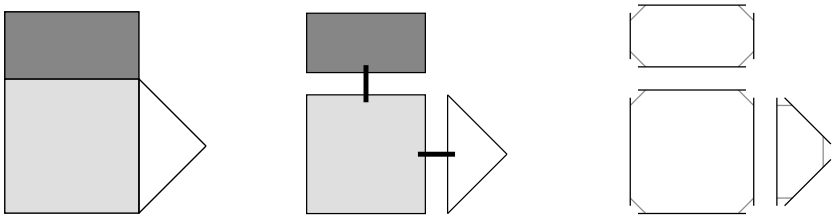
Les cartes combinatoires, qui sont à la base de ce travail, ont été initialement introduites en dimension 2 comme une extension des graphes planaires [Edm60, Tut63, Jac70], et définies ensuite par [Cor73, Cor75]. Puis, elles ont été étendues en dimension 3, sous le nom de pavage [AK88, Lie88, AK89, Spe91], avant d'être définies de manière générale en dimension n [Lie89].

Les cartes combinatoires permettent de représenter les quasi-variétés orientables de dimension n. Elles ont été généralisées [Lie89, Lie91, Lie94] afin de pouvoir représenter les quasi-variétés en dimension n, orientables ou non. Nous présentons section 2.1 nos notations et rappelons quelques notions de base, nécessaires pour la suite de ce travail. Puis nous présentons section 2.2, les cartes combinatoires, et section 2.3 les cartes généralisées, et effectuons une rapide comparaison de ces deux modèles, en montrant sous quelles conditions ils sont équivalents. Nous montrons section 2.4 comment, sous ces conditions, nous pouvons effectuer les conversions entre ces deux modèles, ce qui permet, par la suite, de travailler indifféremment avec l'un ou avec l'autre. Mais les cartes combinatoires aussi bien que les cartes généralisées représentent seulement la topologie des objets. Nous discutons section 2.5 de la géométrie et des différentes possibilités de relier un modèle géométrique aux cartes.

2.1 Rappels et notations

Soit B un ensemble non vide. Nous notons Id_B l'identité sur B. Une *permutation* sur B est une bijection de B dans B. Une *involution* sur B est une permutation f sur B telle que $f = f^{-1}$, ce qui est équivalent à $f \circ f = Id_B$.

Soit $\Phi = \{f_1, \ldots, f_k\}$ des permutations sur un même ensemble B. Nous notons $< \Phi >$ le groupe de permutations engendré par Φ. C'est l'ensemble des permutations qu'il est possible d'obtenir de Φ par application de la composition et de l'inverse. Nous parlons alors de l'*orbite* de $b \in B$ relativement à Φ pour $< \Phi > (b) = \{\phi(b) | \phi \in \Phi\}$, l'ensemble des éléments de B qu'il est possible d'atteindre par application, à partir de b, de n'importe quelle suite de f_i et f_i^{-1}.



a. Un objet 2d. b. Les faces de cet objet. c. Les arêtes de ces faces.

FIG. 2.1 – La décomposition d'un objet afin d'obtenir la carte combinatoire correspondante.

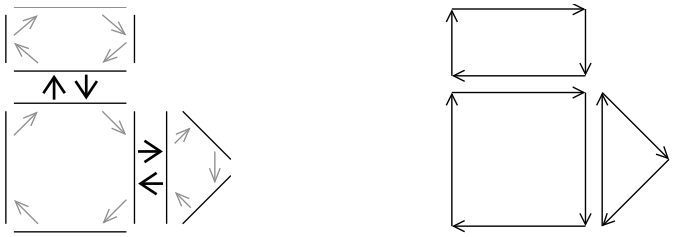
2.2 Les cartes combinatoires

Les cartes combinatoires sont un modèle mathématique de représentation de la topologie des subdivisions de l'espace. Une *subdivision* de l'espace est une partition d'un espace de dimension n en $(n+1)$ sous-ensembles, où chaque élément est appelé *cellule* de dimension 0, 1, 2, 3, ... n (appelés sommets, arêtes, faces et volumes pour les dimensions 0 à 3). Nous parlons de i*-cellule* pour désigner une cellule de dimension i. Des relations de bord sont définies sur ces cellules, où le bord d'une i-cellule est un ensemble de $(j < i)$-cellules. Nous disons que deux cellules sont *incidentes* quand l'une appartient au bord de l'autre, et qu'elles sont *adjacentes* quand elles ont la même dimension i et qu'elles sont incidentes à une même $(i-1)$-cellule.

Les cartes combinatoires permettent de représenter les quasi-variétés orientables. Une *quasi-variété* en dimension n est, de manière intuitive, un objet nd que l'on peut obtenir par assemblage de n-cellules uniquement le long de $(n-1)$-cellules, et tel que chaque $(n-1)$-cellule est incidente à au plus deux n-cellules [Elt94]. Cette notion est différente de la notion de *variété* topologique de dimension n classique [Ago76, Tak91] pour laquelle chaque point doit avoir un voisinage homéomorphe à une sphère de dimension n (ou à une demi-sphère si la variété topologique est ouverte). De ce fait, une quasi-variété n'est pas forcément une variété [Lie93], excepté en dimension 2. Un objet en dimension n est considéré comme orientable si la surface de dimension $(n-1)$ représentant son bord sépare l'espace en deux parties distinctes : l'intérieur et l'extérieur de l'objet.

Les cartes combinatoires permettent de représenter les subdivisions de l'espace ainsi que les relations d'incidence. Elles sont définies de manière formelle en dimension quelconque. Nous parlons de n-carte pour une carte combinatoire de dimension n. Nous pouvons définir, de manière intuitive, les n-cartes, par décompositions successives des différentes cellules d'un objet, comme nous pouvons voir figure 2.1 pour un objet en dimension 2. Nous partons de l'objet à représenter figure 2.1.*a*. Nous commençons par représenter les faces de cet objet de manière disjointe figure 2.1.*b*, la relation d'adjacence entre deux faces étant représentée par un trait noir. Puis nous représentons les arêtes de ces faces de manière disjointe figure 2.1.*c*, en représentant les relations d'adjacences entre ces arêtes par les traits gris.

Nous avons décomposé notre objet initial en un ensemble d'éléments appelés *brins*, qui constituent l'unique élément de la définition des cartes combinatoires. Il faut ensuite reporter les différentes relations d'adjacence sur ces éléments. La relation d'adjacence entre les arêtes est représentée par une permutation, qui pour chaque brin donne le brin suivant de la même face, en respectant l'orientation trigonométrique inverse. Cette relation est notée β_1, car elle met en relation des arêtes



a. Représentation totale. b. Représentation implicite.

FIG. 2.2 – Deux représentations de la même carte combinatoire.

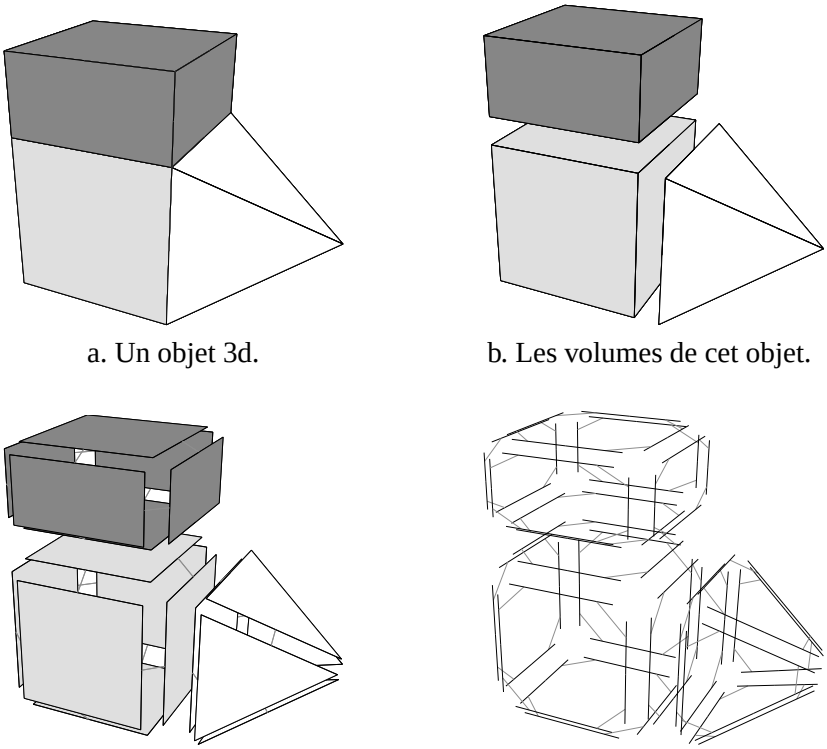
qui sont des cellules de dimension 1. La relation d'adjacence des faces est représentée par une involution et est notée β_2, étant donné qu'elle met en relation des cellules de dimension 2.

Nous pouvons voir figure 2.2.*a* la carte combinatoire de l'objet présenté figure 2.1.*a*. Sur cette figure, chaque brin est représenté par un segment de droite noir, la permutation β_1 est représentée par les flèches grises et l'involution β_2 par les flèches noires. Mais cette représentation peut devenir très vite illisible, lorsque le nombre de brins est important. Afin d'alléger les figures, il est possible de dessiner cette même carte sans représenter explicitement les applications β, comme nous pouvons le voir figure 2.2.*b*. Sur cette figure, nous représentons chaque brin par une flèche, et deux brins en relation par β_1 sont dessinés l'un à la suite de l'autre. Nous pouvons donc retrouver sans difficulté pour chaque brin son image par β_1. De même pour β_2, où deux brins en relation par β_2 seront dessinés parallèles, proches et inversés.

Cette méthode de décomposition d'un objet peut s'utiliser pour n'importe quelle dimension. Nous pouvons voir un second exemple de construction d'une carte combinatoire à partir d'un objet figure 2.3, mais cette fois pour la dimension 3. Nous décomposons l'objet présenté figure 2.3.*a* successivement pour ses volumes figure 2.3.*b*, puis pour les faces de ces volumes figure 2.3.*c* et enfin pour les arêtes de ces faces figure 2.3.*d*. Les éléments obtenus sont les brins de la carte combinatoire représentant notre objet. Il faut maintenant reporter les relations d'adjacence entre chaque cellule sur ces brins. Il existe, comme pour la dimension 2, une permutation β_1 qui met en relation un brin et le brin suivant de la même face, et une involution β_2 qui met en relation deux brins de deux faces adjacentes d'un même volume. En outre, une involution supplémentaire β_3 met en relation deux volumes adjacents.

Nous pouvons voir la carte combinatoire représentée avec l'ensemble des relations entre les brins figure 2.4.*a*. Sur cette figure, nous avons représenté β_1 en vert, β_2 en bleu et β_3 en rouge. Mais comme pour la dimension 2, nous pouvons dessiner cette carte sans représenter explicitement ces relations, comme nous pouvons le voir figure 2.4.*b*, où l'on retrouve sans difficulté chaque relation pour n'importe quel brin de par la manière dont est dessinée cette carte.

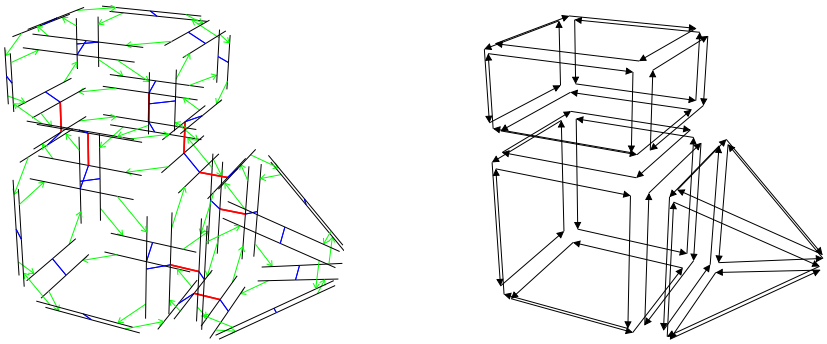
Cette méthode de construction permet d'appréhender les cartes combinatoires de manière intuitive. Mais dans la suite de ce travail nous avons besoin de la définition formelle des cartes combinatoires afin d'utiliser ses propriétés algébriques. Nous trouvons cette définition par exemple dans [Lie91].



a. Un objet 3d. b. Les volumes de cet objet.

c. Les faces de ces volumes. d. Les arêtes de ces faces.

FIG. 2.3 – La décomposition d'un objet 3d afin d'obtenir la carte combinatoire correspondante.



a. Représentation totale. b. Représentation implicite.

FIG. 2.4 – Deux représentations de la même carte combinatoire.

Définition 1 (carte combinatoire) *Soit $n \geq 0$. Une n carte combinatoire , (ou n-carte) est une algèbre $C = (B, \beta_1, \ldots, \beta_n)$ où :*

1. *B est un ensemble fini de brins ;*
2. *β_1 est une* permutation *sur B ;*
3. *$\forall\, 2 \leq i \leq n$, β_i est une* involution *sur B ;*
4. *$\forall\, 1 \leq i \leq n-2, \forall\, i+2 \leq j \leq n$, $\beta_i \circ \beta_j$ est une* involution.

Les brins sont ici une notion abstraite, et servent uniquement de support pour les différentes applications. Seul β_1 est une permutation, les autres β_i sont des involutions. La dernière ligne de cette définition fixe des contraintes sur la manière dont les brins sont mis en relation afin de garantir la validité des objets représentés. Par exemple, en dimension 3, la contrainte ajoutée est que $\beta_1 \circ \beta_3$ doit être une involution, ce qui revient à dire que lorsque nous mettons deux brins de deux faces différentes en relation pour β_3, nous devons obligatoirement mettre tous les autres brins de ces deux faces en relation deux à deux par β_3.

Lorsque deux brins b_1 et b_2 sont tels que $\beta_i(b_1) = b_2$, nous disons que b_1 est *i-cousu* à b_2. Étant donné que les β_i, pour $i \neq 1$, sont des involutions, si b_1 est *i-cousu* à b_2 alors b_2 est *i-cousu* à b_1. Nous parlons de brin *i-libre* pour un brin b tel que $\beta_i(b) = b$. Ceci n'est pas vrai pour β_1, où un brin b tel que $\beta_1(b) = b$ n'est pas considéré comme libre, mais comme une boucle (une arête cousue avec elle-même).

Remarquons que chaque brin possède forcément une image pour chaque β_i, étant donné que ces β_i sont des permutations. L'opération consistant à mettre en relation deux brins pour β_i est appelée *i-couture*. Nous notons β_0 la permutation β_1^{-1}, et β_{ijk} la composition $\beta_k \circ \beta_j \circ \beta_i$. Nous disons qu'une carte est *i-fermée* si aucun de ses brins n'est i-libre. Une carte n'étant pas i-fermée est dite *i-ouverte*. Enfin, nous parlons de carte *fermée* (resp. *ouverte*) pour une carte i-fermée (resp. i-ouverte) pour toutes les dimensions de l'espace.

La notation utilisée ici est celle définie par [Lie91]. Nous la préférons à celle définie initialement par [Cor75] car elle est définie de manière générale en dimension n et car chaque β_i met en relation deux cellules de dimension i. Cette propriété est très intéressante et permet de définir des algorithmes ou des propriétés génériques pour n'importe quelle dimension, comme par exemple la définition des différentes cellules dans les cartes combinatoires.

En effet, nous avons vu, lors de nos exemples de constructions, que les cartes combinatoires représentent les objets par décompositions successives de leurs cellules. Mais la carte obtenue ne représente pas explicitement ces différentes cellules. Nous pouvons les retrouver à l'aide de la notion d'orbite présentée section 2.1.

Définition 2 (i-cellule) *Soit C une n-carte, b un brin de cette carte, et $i \in \{0, \ldots, n\}$. La i-cellule incidente à b est :*

- *Si $i = 0$: $< \beta_{02}, \ldots, \beta_{0n} > (b)$;*
- *Sinon : $< \beta_1, \ldots, \beta_{i-1}, \beta_{i+1}, \ldots, \beta_n > (b)$.*

Nous avons deux cas différents. L'un pour la définition des 0-cellules (les sommets), et l'autre pour les autres cellules. Cela provient du fait que β_1 est une permutation alors que les autres β sont des involutions : les cartes combinatoires n'ont pas une définition homogène. C'est une

des raisons qui a amené Pascal Lienhardt à définir les cartes généralisées, que nous présentons section 2.3. Cette définition des cellules profite pleinement de la définition générique des cartes combinatoires, où n'importe quelle i-cellule est définie simplement en une seule fois, ce qui est moins évident avec les notations de [Cor75].

De manière intuitive, nous pouvons voir une i-cellule comme l'ensemble des brins que nous pouvons atteindre par un parcours en largeur d'origine b, en utilisant les β particuliers ainsi que leurs inverses. Les 0-cellules sont définies ainsi, car nous parcourons seulement un brin sur deux, afin de n'atteindre seulement les brins « sortants » du sommet incident à b. Les autres i-cellules sont simplement l'orbite composée de tous les β sauf β_i. En effet, comme β_i permet de changer de i-cellule, si nous l'interdisons nous restons, au cours de notre parcours, sur les brins de la même i-cellule. Remarquons enfin que chaque ensemble de i-cellule est une partition de l'ensemble des brins de la carte. Chaque brin appartient donc exactement une fois à chaque i-cellule.

Avec cette définition des i-cellules, nous pouvons définir de manière formelle la notion d'incidence dans le cadre des cartes combinatoires, sachant qu'une i-cellule est donc considérée comme un ensemble de brins.

Définition 3 (incidence) *Deux cellules C_1 et C_2 sont* incidentes *si et seulement si elles sont de dimensions différentes, et $C_1 \cap C_2 \neq \emptyset$.*

Avec cette définition de l'incidence, nous pouvons également définir de manière formelle la notion d'adjacence puisque celle-ci s'appuie sur la notion d'incidence.

Définition 4 (adjacence) *Deux cellules C_1 et C_2 sont* adjacentes *si et seulement si elles sont de même dimension i, et s'il existe une cellule c de dimension $i-1$ incidente à C_1 et à C_2.*

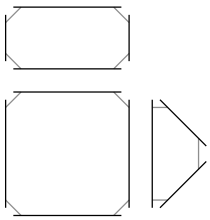
Enfin nous pouvons introduire la notion de degré d'une i-cellule, qui est primordiale pour la suite de ce travail.

Définition 5 (degré d'une i-cellule) *Le* degré d'une i-cellule C*, est le nombre de $(i+1)$-cellules distinctes incidentes à C.*

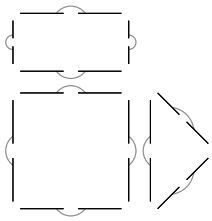
Remarquons que le degré d'une n-cellule dans un espace en dimension n n'est donc pas défini. De plus, le degré d'une i-cellule ne peut jamais être égal à 0, car il existe au moins un brin dans cette cellule qui, s'il n'est cousu à aucun autre brin, est également une $(i+1)$-cellule. Enfin, le degré d'une $(n-1)$-cellule dans un espace en dimension n, est forcément égal à 1 ou 2, étant donné que nous représentons uniquement des quasi-variétés.

2.3 Les cartes généralisées

Les cartes généralisées sont une extension des cartes combinatoires permettant de représenter les quasi-variétés orientables ou non. Leur principal avantage est que leur définition est homogène à toutes les dimensions, contrairement aux cartes combinatoires, ce qui simplifie encore les définitions et l'écriture des algorithmes.

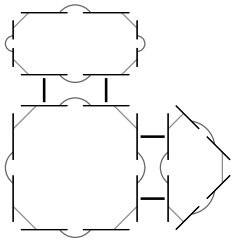


a. Les arêtes obtenues figure 2.1 après décompositions successives.

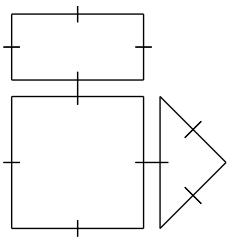


b. Les sommets de ces arêtes.

FIG. 2.5 – La décomposition supplémentaire d'un objet 2d afin d'obtenir une carte généralisée.



a. Représentation totale.



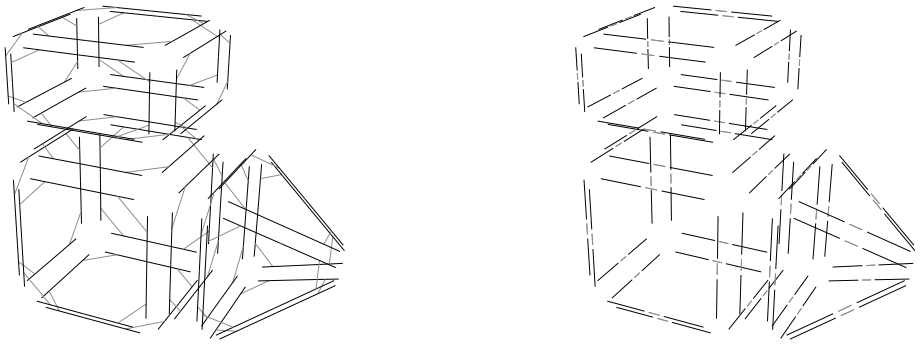
b. Représentation implicite.

FIG. 2.6 – Deux représentations de la même carte généralisée.

Pour définir une carte généralisée de manière intuitive, nous appliquons le même principe de décomposition que celui utilisé pour les cartes combinatoires, mais nous effectuons une décomposition supplémentaire afin de distinguer les sommets. Si nous reprenons le premier exemple utilisé pour les cartes combinatoires présenté figure 2.1, nous avons obtenu au final la décomposition rappelée figure 2.5.*a*. Nous décomposons ensuite les sommets, à partir de cette décomposition en arêtes et obtenons la décomposition présentée figure 2.5.*b*, où les relations d'adjacence entre les sommets d'une arête sont représentées par les arcs de cercle gris. Les éléments obtenus sont les brins de la carte généralisée. Il suffit ensuite, comme pour les cartes combinatoires, de reporter les relations d'adjacence sur ces brins pour obtenir la carte généralisée présentée figure 2.6.*a*.

Étant donné que nous avons, pour les cartes généralisées, également séparé les sommets, nous n'avons plus besoin, comme pour les cartes combinatoires, d'utiliser une permutation pour parcourir les faces. En effet, chaque « côté » d'une arête sera lié avec l'arête suivante de la face pour ce sommet. Il existe donc une involution α_0 qui met en relation les deux brins de la même face et de la même arête (représentée par les arcs de cercle gris), une involution α_1 qui met en relation les deux brins de la même face et du même sommet (représentée par les segments gris clair), et une involution α_2 qui met en relation les deux brins de la même arête et du même sommet.

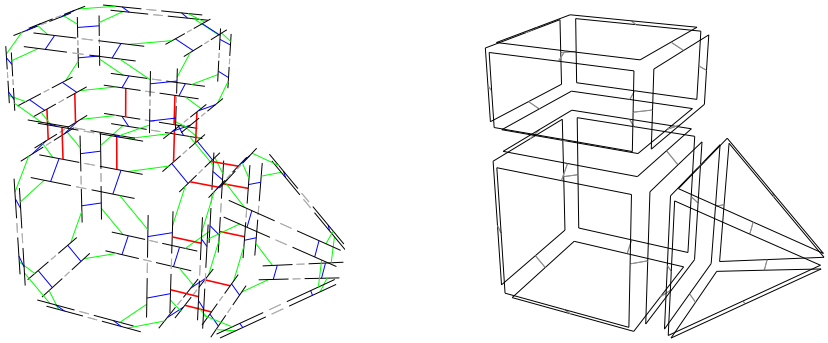
Comme pour les cartes combinatoires, nous ne représentons pas de manière explicite toutes les involutions et utilisons la représentation intuitive présentée figure 2.6.*b*. Deux brins cousus par α_0 sont représentés par un seul segment portant une petite barre en son milieu, deux brins cousus par



a. Les arêtes obtenues figure 2.3 après décompositions successives.

b. Les sommets de ces arêtes.

FIG. 2.7 – La décomposition supplémentaire d'un objet 3d afin d'obtenir la carte généralisée correspondante.



a. Représentation totale.

b. Représentation implicite.

FIG. 2.8 – Deux représentations de la même carte généralisée.

α_1 sont représentés de manière contiguë, et deux brins cousus par α_2 sont représentés parallèles et proches, avec la barre représentant α_0 traversant les deux arêtes.

Nous pouvons voir figure 2.7 le même principe appliqué en dimension 3 à notre exemple de la figure 2.3. Pour les cartes combinatoires, nous avons obtenu au final la décomposition en arêtes rappelée figure 2.7.*a*. La décomposition des sommets des arêtes de cette figure est présentée figure 2.7.*b*, et la carte généralisée correspondante figure 2.8.*a*. Sur cette figure, nous avons représenté α_0 en gris, α_1 en vert, α_2 en bleu et α_3 en rouge. Mais comme pour les cartes combinatoires, nous préférons utiliser la représentation implicite présentée figure 2.8.*b*. En effet, bien que les α ne soient pas représentés explicitement, nous pouvons les retrouver sans difficulté de par la manière dont est dessinée la carte.

[Lie91] définit les cartes généralisées en dimension n de manière simple et générique :

Définition 6 (carte généralisée) *Soit $n \geq -1$. Une n carte généralisée, (ou n-G-carte) est une algèbre $G = (B, \alpha_0, \ldots, \alpha_n)$ où :*

1. *B est un ensemble fini de brins ;*
2. *$\forall\, 0 \leq i \leq n$, α_i est une* involution *sur B ;*
3. *$\forall\, 0 \leq i \leq n-2, \forall\, i+2 \leq j \leq n$, $\alpha_i \circ \alpha_j$ est une* involution.

En comparaison avec les cartes combinatoires, il existe une involution supplémentaire, et il n'y a plus de différence entre les α qui sont tous des involutions. Les n-G-cartes sont définies à partir de $n = -1$, afin de pouvoir définir la G-carte vide, composée uniquement d'un ensemble de brins sans aucune involution. Les G-cartes sont totalement homogènes, ce qui simplifie les définitions et les algorithmes comme nous pouvons le voir, par exemple, pour la définition des cellules.

Définition 7 (i-cellule) *Soit G une n-G-carte, b un brin de cette carte, et $i \in \{0, \ldots, n\}$. La i-cellule incidente à b est $< \alpha_0, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n > (b)$.*

Mais les G-cartes sont deux fois plus coûteuses en espace mémoire que les cartes combinatoires. Ce surcoût en mémoire peut être prohibitif, et c'est pour cette raison que nous avons décidé d'utiliser les cartes combinatoires, dans le cadre de ce travail. Mais nous allons par moment utiliser le formalisme des cartes généralisées pour profiter de leur définition homogène.

En effet, lorsque nous travaillons avec des G-cartes orientables, nous savons effectuer facilement la transformation permettant de passer aux cartes combinatoires, et lorsque nous travaillons avec une carte combinatoire, nous pouvons définir sans problème la carte généralisée correspondante. Nous devons donc être en mesure de savoir si une G-carte est orientable ou non. Nous trouvons la définition suivante dans [Lie91] :

Définition 8 (G-carte orientable) *Soit $G = (B, \alpha_0, \ldots, \alpha_n)$ une n-G-carte fermée. G est* orientable *si et seulement si la n-carte $HG = (B, \alpha_{01}, \ldots, \alpha_{0n})$ a exactement deux composantes connexes distinctes. G est* non orientable *sinon.*

Cette définition permet facilement de tester, étant donné une G-carte, si elle est orientable ou non. De manière intuitive, une G-carte orientable contient les deux orientations possibles de la carte combinatoire correspondante. Cette définition est valable uniquement pour les G-cartes fermées, mais s'étend aisément aux G-cartes ouvertes.

Nous pouvons remarquer que nous n'avons pas besoin de redéfinir les notions d'incidence, d'adjacence, et de degré d'une i-cellule, dans ce cadre particulier des G-cartes. En effet, ces définitions s'appuient uniquement sur la définition d'une i-cellule, définition que nous venons de donner. Elles restent donc valides dans le cadre des G-cartes étant donné qu'elles utilisent seulement les brins appartenant aux différentes cellules.

2.4 Conversions entre carte combinatoire et carte généralisée

La conversion entre une carte combinatoire et une carte généralisée peut s'effectuer sans contrainte, étant donné que le domaine de modélisation des cartes généralisées inclut celui des cartes

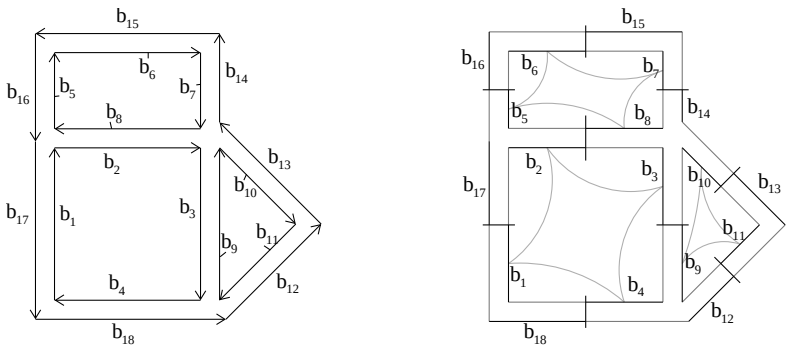


FIG. 2.9 – Un exemple de conversion de carte en G-carte.

combinatoires. Comme pour la notion de G-carte orientable, la conversion de carte en G-carte est définie uniquement pour les cartes fermées. Mais nous pouvons étendre cette définition aux cartes ouvertes, par exemple en commençant par fermer la carte, puis en la convertissant en G-carte, et enfin en supprimant les brins n'appartenant pas à la carte originale avant sa fermeture.

Définition 9 (conversion carte → G-carte) *Soit $C = (B, \beta_1, \ldots, \beta_n)$ une n-carte fermée. Nous notons $B = \{b_1, \ldots, b_k\}$. La n-G-carte représentant la même subdivision que C est $G = (B', \alpha_0, \ldots, \alpha_n)$ où $B' = \{b_1, \ldots, b_k, c_1, \ldots, c_k\}$, où les c_i sont des nouveaux brins. Alors $\forall 1 \leq i \leq k$:*

1. $\alpha_0(b_i) = c_i$ *et* $\alpha_0(c_i) = b_i$ *;*
2. $\alpha_1(b_i) = \alpha_0(\beta_0(b_i))$ *et* $\alpha_1(c_i) = \beta_1(b_i)$ *;*
3. $\forall 2 \leq j \leq n$ *:* $\alpha_j(b_i) = \alpha_0(\beta_j(b_i))$ *et* $\alpha_j(c_i) = \beta_j(b_i)$.

De manière informelle, pour convertir une carte en G-carte, nous « coupons » chaque brin b_i de C en deux brins b_i et c_i. Nous pouvons voir figure 2.9 un exemple de conversion d'une carte en G-carte. Chaque brin b_i de la carte reste, dans la G-carte, le brin incident à la même face, à la même arête et au même sommet. Le brin de la même face, de la même arête et du sommet opposé est c_i, qui est donc α_0-cousu à b_i. La définition des involutions de la G-carte se fait ensuite sans difficulté, en différenciant à chaque fois deux cas, suivant si le brin concerné est un brin de la carte, donc possédant des β coutures, ou un nouveau brin. De plus, il faut différencier la définition de α_1 de la définition des autres α, du fait que les cartes ne sont pas homogènes et qu'il nous faut alors considérer différemment β_1 des autres β.

La transformation inverse se fait avec encore moins de difficultés, étant donné que nous pouvons définir chaque β comme une composition de certains α. Mais pour cela, il faut nécessairement que la G-carte soit orientable. En effet, si elle ne l'est pas, il n'existe pas de carte représentant la même subdivision de l'espace. De plus, cette définition est, comme la précédente, valable uniquement pour les G-cartes fermées.

Définition 10 (conversion G-carte → carte) *Soit $G = (B, \alpha_0, \ldots, \alpha_n)$ une n-G-carte orientable et fermée, et b un brin de G. La n-carte représentant la même subdivision que G, et contenant b, est $C = (B', \beta_1, \ldots, \beta_n)$ où*

– $B' =< \alpha_{01}, \ldots, \alpha_{0n} > (b)$;
– $\forall 1 \leq i \leq n, \forall c \in B', \beta_i(c) = \alpha_{0i}(c)$

La carte C est définie en conservant un brin sur deux de la G-carte. En effet, comme nous savons que la G-carte est orientable et fermée, l'orbite $< \alpha_{01}, \ldots, \alpha_{0n} > (b)$ va contenir un brin sur deux de la G-carte, ce qui n'est pas le cas sinon. La définition des différentes applications β sur cet ensemble de brins se fait sans problème, chaque β_i étant simplement la composition de α_0 avec α_i. Remarquons que la carte ainsi définie représente une orientation de la G-carte. Il est possible de définir la carte représentant l'autre orientation possible, en prenant comme brin de départ pour la définition de B', un brin n'appartenant pas à la première carte.

Ces méthodes de conversion nous permettent, dans la suite de cette présentation, de travailler indifféremment avec une carte combinatoire ou une carte généralisée orientable. Cela nous permettra, par exemple, de travailler avec des G-cartes pour définir formellement l'opération de fusion en dimension quelconque, afin de profiter de leur homogénéité, puis de travailler avec des cartes combinatoires pour définir nos algorithmes d'extractions, qui ont besoin d'être implantés avec le modèle des cartes combinatoires pour des contraintes d'espace mémoire.

2.5 Plongement des cartes

Les cartes combinatoires et les cartes généralisées, représentent seulement la topologie des objets. Mais la plupart des applications ont besoin également de représenter la géométrie de ces objets, par exemple pour les afficher, pour calculer des caractéristiques de forme, de taille… Il nous faut donc associer un modèle géométrique aux cartes. Pour cela, nous pouvons associer un élément géométrique de dimension i à chaque i-cellule de la carte. Nous appelons *plonger* l'opération qui consiste à associer un modèle géométrique à un modèle topologique, et nous parlons du *plongement* d'un modèle topologique pour désigner le modèle géométrique associé.

Par exemple, en dimension 2, nous pouvons associer à chaque sommet topologique (0-cellule) les coordonnées d'un sommet géométrique. Ce plongement peut suffire pour représenter totalement la géométrie des objets modélisés, si chaque arête de la carte est plongée par un segment de droite.

Mais dans le cas général, en dimension n, lorsqu'il n'y a pas de lien entre le plongement des cellules de dimension i et le plongement des cellules de dimension $i-1$, nous devons alors plonger chaque i-cellule topologique par un objet géométrique ouvert de dimension i. Cet objet doit être ouvert, car le plongement du bord de la i-cellule topologique est représenté par les plongements de ses $(i-1)$-cellules topologiques incidentes. Nous parlons plus en détail, dans la suite de cette présentation, du plongement dans le cadre plus précis de ce travail et présentons quelques modèles géométriques pouvant s'associer à notre utilisation particulière des cartes.

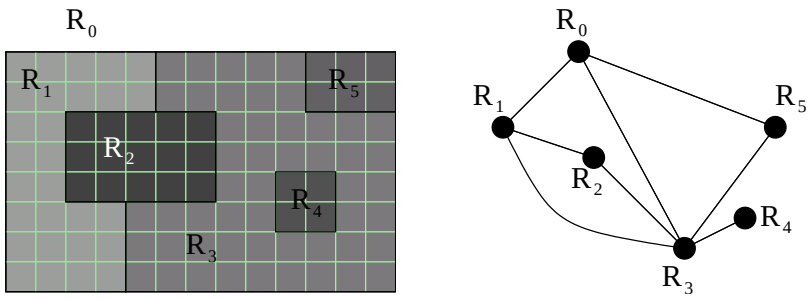
CHAPITRE 3

QUELQUES MODÈLES EN DIMENSION 2

Le but principal de notre travail est de définir un modèle permettant la représentation d'image 3d segmentées en régions. Ce problème a été beaucoup étudié en dimension 2, et de nombreuses solutions ont été proposées. Nous avons donc commencé, dans un premier temps, à étudier ces modèles afin de connaître les solutions envisagées, mais également afin de retenir celles qui étaient les plus susceptibles d'être étendues en dimension supérieure.

Nous présentons ici quelques-uns de ces modèles. Dans un premier temps, section 3.1, nous présentons le Region Adjacency Graph (RAG) [Ros74]. C'est le premier modèle de représentation d'images 2d segmentées en régions à avoir été défini, et également la source d'inspiration de nombreux modèles différents. Nous présentons ensuite, section 3.2, les graphes duaux [Kro95, MK95, KM95, Wil95] qui sont une évolution du RAG palliant plusieurs de ses problèmes. Ensuite, section 3.3, nous présentons les cartes discrètes qui sont un modèle de représentation d'images 2d segmentées en régions, utilisant les cartes combinatoires et les frontières interpixel [Dom92, Bru96, BD97, BDB97, BB98, BD99]. Ce modèle est très proche de celui que nous présentons section 3.4, qui est le graphe topologique des frontières (TGF) [Fio95, Fio96, AFG99] également basé sur les cartes combinatoires. Ce sont ces deux derniers modèles qui ont été à la base de notre travail, même si nous avons dû redéfinir une variante de ces modèles en dimension 2, afin de pouvoir ensuite l'étendre plus facilement en dimension 3.

Nous présentons ici uniquement les quelques modèles qui sont, soit très proches de nos préoccupations, soit les principaux acteurs qui ont fait évoluer le monde de la modélisation et de la représentation d'images. Mais il existe de nombreux autres modèles, proposant pour certains des solutions qui peuvent être vues comme des variantes des quatre modèles présentés ici comme [PJ97], et d'autres qui proposent des solutions différentes. Par exemple le *medial axis transform* (MAT) [RP68, RK82] qui consiste à représenter une image par un ensemble de blocs de valeurs uniformes. Mais ce modèle ne représente pas les frontières des régions de manière explicite. De nombreuses autres solutions ont défini des modèles hiérarchiques, comme par exemple les pyramides [Bro82, PRW82, Kro95], les quadtrees [Sam80, DRS80, Sam84], ou plus récemment les pyramides combinatoires [BK99, BK00, BK01]. Ces modèles permettent la représentation d'une image à plusieurs niveaux, ce qui est intéressant notamment pour une représentation multi-échelle. Mais la plupart de ces modèles ne permettent pas des mises à jour efficaces lors d'opérations de



a. Une image segmentée en régions. b. Le RAG correspondant.

FIG. 3.1 – Une image 2d segmentée en régions et le RAG correspondant.

modifications de l'image, et nécessitent parfois de reparcourir l'ensemble du modèle afin d'accéder ou de modifier une information. Ces modèles sont plus éloignés de nos préoccupations car ils ne représentent pas de manière explicite le bord des régions de l'image.

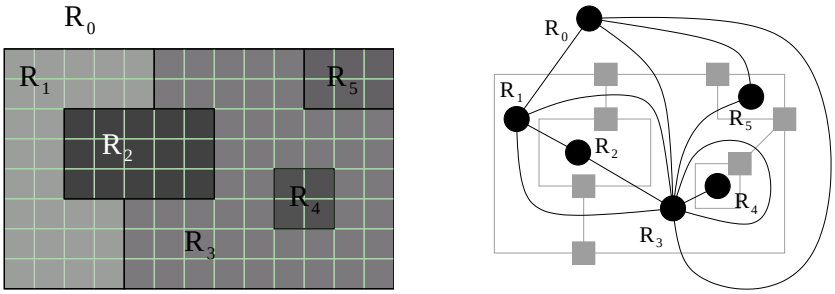
De nombreux autres modèles de représentation par bords ont été définis, mais du point de vue topologique et non plus dans le cadre précis de la représentation d'images segmentées en régions. En dimension 2 sont définis par exemple les *winged-edge* ou arêtes ailées [Bau75, Wei85] et une de ses extensions, le graphe d'adjacence des faces [ADFF85] ou encore les *quad-edge* [GS85]. En dimension 3, nous pouvons trouver les *radial-edge* [Wei88] ou une extension des quad-edge, les *facet-edge* [DL87]. Enfin en dimension n, nous pouvons trouver les *incidence posets* [Bri89], le graphe d'incidence [Ede87] et les *selective geometric complexes* [RO89]. [Lie91] montre que ces modèles sont divisés principalement en deux classes : les graphes d'incidences et les modèles topologiques ordonnés. De plus, il effectue une comparaison entre quelques-uns de ces modèles, et montre qu'ils sont globalement équivalents en regard de la classe des objets qu'ils peuvent modéliser.

Ces quelques références montrent que le nombre de modèles existants est très important. Mais nous nous intéressons à un cadre beaucoup plus restreint que le domaine de modélisation de plusieurs de ces modèles. C'est pour cette raison que nous présentons uniquement les quatre modèles représentant les images segmentées en régions que nous avons déjà introduits en début de ce chapitre : le RAG, les graphes duaux, les cartes discrètes et le TGF.

3.1 Le RAG

Le RAG est une des premières structures permettant de représenter les images 2d segmentées en régions, en rendant compte des relations d'adjacence entre les différentes régions de l'image. Il a comme avantage principal d'être défini très simplement. En effet, c'est un graphe où chaque sommet représente une région de l'image, et une arête est présente entre deux sommets si les deux régions correspondantes sont adjacentes. La figure 3.1 présente une image segmentée en régions et le RAG correspondant.

Nous pouvons voir sur cet exemple, l'image segmentée en régions figure 3.1.*a*, image que nous utilisons tout au long de ce chapitre pour comparer les différents modèles, et le RAG correspondant



a. Une image segmentée en régions. b. Le graphe dual correspondant.

FIG. 3.2 – Une image 2d segmentée en régions et le graphe dual correspondant.

à cette image figure 3.1.*b*. Nous considérons que l'image est entourée par une région infinie, R_0, contenant l'ensemble des pixels n'appartenant pas à l'image. Le RAG représente donc les relations d'adjacences des régions de l'image. Ses principaux avantages sont qu'il :

- est simple à définir ;
- peut s'étendre sans problème en dimension quelconque ;
- peut être facilement mis à jour, lorsque des régions de l'image sont fusionnées.

Mais il est facile de voir sur notre exemple que ce graphe possède également plusieurs inconvénients :

- il ne différencie pas la relation d'adjacence et d'inclusion ;
- il ne représente pas l'adjacence multiple. Sur notre exemple, la région R_1 est adjacente deux fois à la région R_3, information qui n'est pas représentée dans le RAG ;
- il n'est pas représentatif de l'image : deux images topologiquement différentes pourront avoir deux RAG identiques.

Ces inconvénients ont amené [Kro95, MK95, KM95, Wil95] à proposer une évolution du RAG, les graphes duaux.

3.2 Les graphes duaux

L'idée principale des graphes duaux est de conserver deux graphes en parallèle, une extension du RAG d'un côté, et son graphe dual de l'autre. Le premier graphe est une simple extension du RAG, qui est maintenant un multi-graphe, rendant donc compte des adjacences multiples, avec également des boucles lorsqu'une région contient des régions incluses. Ces boucles proviennent du fait que le graphe primal et son dual doivent tous deux être connexes. De ce fait, dans le dual, une arête fictive permet à la région incluse d'être reliée à la région la contenant. Cette arête devient, dans le primal, une boucle autour de la région contenant l'autre région.

Nous pouvons voir figure 3.2 un exemple de graphe dual de notre image d'exemple. Sur cet exemple, figure 3.2.*b*, nous avons représenté en noir le graphe primal, et en gris le graphe dual. Il est facile de voir que le graphe primal est l'extension du RAG tenant compte des adjacences multiples, il y a maintenant deux arêtes entre R_1 et R_3, et représentant correctement la relation d'inclusion au moyen de boucles, la région R_3 qui contient la région R_4 possède une boucle sur son sommet qui entoure le sommet de R_4.

Sur le graphe gris, nous pouvons noter la présence d'une *arête fictive* qui relie le bord de la région R_4 avec le bord de la région R_3. Cette arête est fictive car elle ne représente pas, contrairement aux autres arêtes de ce graphe, une frontière entre deux régions distinctes. Elle est nécessaire afin de conserver le graphe gris connexe. Nous pouvons noter que dans le graphe primal, représenté en noir, cette arête fictive devient la boucle autour de R_3. Cette boucle nous permet de différencier la relation d'inclusion et la relation d'adjacence. Mais pour cela, les deux graphes doivent être planaires, et il faut pouvoir reconnaître une boucle qui entoure géométriquement une autre région.

Ce modèle répond correctement aux problèmes du RAG. Ses principaux avantages sont qu'il :
- code correctement la relation d'inclusion ;
- représente toute la topologie des objets, y compris les adjacences multiples ;
- est donc caractéristique d'une image.

Mais il est facile de voir sur notre exemple que ce graphe possède également plusieurs inconvénients :
- il faut maintenir en parallèle deux graphes, ce qui complique et multiplie par deux les mises à jour ;
- il faut analyser la géométrie des arêtes afin de reconnaître la relation d'inclusion, c'est-à-dire trouver si une boucle entoure géométriquement un sommet du graphe ;
- la présence d'arêtes fictives et de boucles n'est pas totalement satisfaisante. En effet, ces arêtes obligent à avoir des traitements différents pour les arêtes « normales » et pour les arêtes fictives lors des différentes consultations ou mises à jour de ce modèle.

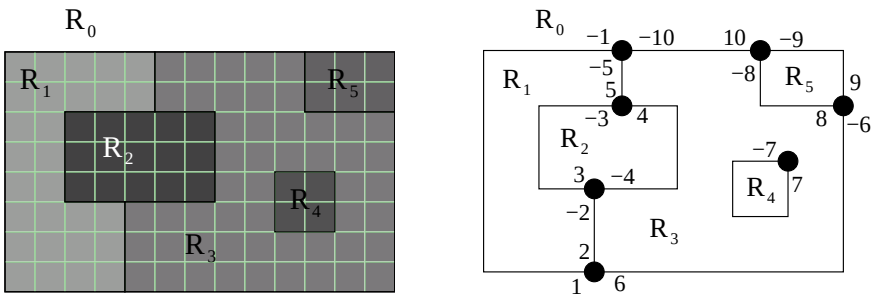
Pour ces raisons, les graphes duaux, même s'ils apportent une solution aux problèmes posés par le RAG, ne sont pas totalement satisfaisants. De plus, un autre problème, et non des moindres, peut être imputé aux graphes duaux : ils ne sont pas extensibles facilement en dimension supérieure, ce qui est le but principal de ce travail. Ce sont ces inconvénients qui ont fait que de nombreux travaux se sont poursuivis dans ce domaine de recherche afin de définir un modèle résolvant ces problèmes.

3.3 Les cartes discrètes

Ce modèle est le résultat du travail de thèse de Jean-Philippe Domenger [Dom92], qui a utilisé les cartes combinatoires afin de représenter les images 2d segmentées en régions. Les cartes combinatoires permettent de rendre compte de la topologie, mais ce modèle représente également la géométrie, en représentant les frontières interpixel de l'image.

Nous pouvons voir un exemple de carte discrète représentant les frontières interpixel de notre image segmentée en régions figure 3.3. La notation utilisée ici est celle définie par [Cor73, Cor75]. En dimension 2, une carte combinatoire est définie avec un ensemble de brins, une permutation σ et une involution α. Cette dernière involution est représentée en numérotant les brins par des entiers positifs et négatifs, tel que pour tout brin b, $\alpha(b) = -b$. La permutation σ représente les sommets de la carte topologique, orientés dans le sens trigonométrique inverse. Figure 3.3.*b* nous avons par exemple $\sigma(-1) = -5$, $\sigma(-5) = -10$ et $\sigma(-10) = -1$.

Ces notations permettent de représenter les cartes combinatoires en dimension 2, de manière duale de notre définition présentée section précédente. En effet, ici la permutation représente les sommets, alors que pour notre notation elle représente les faces. Mais la conversion entre les deux notations se fait sans aucun problème. Lorsque nous avons une 2-carte $C = (B, \beta_1, \beta_2)$, la



a. Une image segmentée en régions. b. La carte discrète correspondante.

FIG. 3.3 – Une image 2d segmentée en régions et la carte discrète correspondante.

carte combinatoire utilisant les notations de Cori est simplement définie par $C' = (B, \sigma, \alpha)$, avec $\sigma = \beta_2 \circ \beta_1$ et $\alpha = \beta_2$. La transformation inverse se définit également sans aucun problème. Lorsque nous avons une carte $C = (B, \sigma, \alpha)$, la 2-carte correspondante pour nos notations est simplement $C' = (B, \beta_1, \beta_2)$ avec $\beta_1 = \alpha \circ \sigma$ et $\beta_2 = \alpha$. Cette carte est notée $C = (B, \varphi, \alpha)$ avec les notations de Cori, où $\varphi = \beta_1$ et $\alpha = \beta_2$.

Cette carte combinatoire représente les frontières interpixel de l'image 2d segmentée en régions, comme nous pouvons le vérifier sur notre exemple. Chaque arête de la carte représente une frontière interpixel entre deux régions adjacentes, et les adjacentes multiples sont correctement représentées. Ce modèle étant basé sur les cartes combinatoires, il représente donc totalement la topologie des objets de l'image. En effet, nous pouvons retrouver facilement toutes les relations d'adjacence ainsi que toutes les relations d'incidence entre n'importe quelles cellules de l'image.

Le seul problème subsistant avec les cartes combinatoires est la représentation de la relation d'inclusion. En effet, nous pouvons voir figure 3.3.*b* que nous avons deux composantes connexes distinctes dans la carte combinatoire, étant donné que nous avons la région R_4 qui est totalement incluse dans la région R_3. Afin de conserver cette information, un arbre d'inclusion est ajouté à la carte combinatoire, permettant de rendre compte de toutes les relations d'inclusion, et de positionner les différentes composantes connexes de la carte les unes par rapport aux autres.

Ce modèle répond, comme les graphes duaux, à tous les problèmes du RAG. Mais il présente des avantages supplémentaires qui résolvent également les problèmes de ces derniers. En effet, les cartes combinatoires ont l'avantage de conserver les mêmes informations que les graphes duaux, mais avec une seule et même structure. Nous n'avons donc ici plus besoin de conserver deux structures en parallèles, ce qui évite des doubles mises à jour. De plus, les cartes combinatoires ajoutent la notion de face, notion qui n'apparaît pas pour les graphes duaux. Il n'y a pas d'élément fictif, et donc les traitements en sont plus homogènes. La relation d'inclusion est codée correctement, et même si c'est dans une structure annexe, l'arbre d'inclusion, cela facilite les parcours et les mises à jour en évitant d'avoir, pour chaque traitement, des cas particuliers afin de gérer les éventuels éléments fictifs. Enfin, il est facile d'associer à ce type particulier de carte, un modèle géométrique rendant compte de la forme des régions de l'image. Pour cela, il est possible d'associer à chaque arête de la carte une suite de 1-cellules représentant la courbe de dimension 1 correspondant à la frontière entre les deux régions. Mais d'autres solutions sont envisageables, comme nous pourrons le voir dans la suite de ce travail.

3.4 Le TGF

Le graphe topologique des frontières (TGF) est un des résultats du travail de thèse de Christophe Fiorio [Fio95]. Comme pour le modèle précédent, ce modèle est basé sur les cartes combinatoires, et représente les frontières interpixel des régions de l'image.

Ce TGF est en fait une carte combinatoire avec un arbre d'inclusion des régions, de manière totalement similaire au modèle présenté à la section précédente. Il est très intéressant de constater que les deux approches de Jean-Philippe Domenger et de Christophe Fiorio, les ont amenés à la définition de structures totalement équivalentes. Il existe seulement deux petites différences entre ces deux structures :

- l'arbre d'inclusion du TGF est représenté de manière implicite, chaque région conservant son bord extérieur et une liste de ses éventuels bords intérieurs, ces listes étant simplement une manière particulière de représenter un arbre ;
- pour représenter le TGF, au lieu de dessiner les brins d'une région sur les frontières interpixel correspondantes, ils sont dessinés de manière similaire au RAG, entre les sommets représentants les régions de l'image.

Mais ces deux différences sont minimes. En effet, le fait de coder l'arbre d'inclusion avec des listes, ou de manière plus classique n'a pas de conséquences sur l'utilisation de la structure. La deuxième différence porte simplement sur la manière habituelle de représenter les deux modèles, et n'a donc clairement aucune conséquence sur leur définitions. Le TGF est défini par exemple dans [Fio95], par :

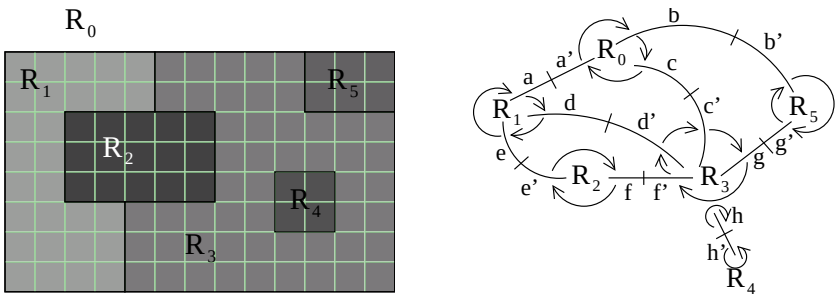
Définition 11 (TGF) *Le* graphe topologique des frontières *est le quadruplet* $G = (V, D, \alpha, \sigma)$ *avec :*

- V *est l'ensemble des régions de l'image segmentée, plus une région infinie ;*
- D *est l'ensemble des brins de la carte combinatoire ;*
- α *est une involution sur* D *;*
- σ *est une permutation sur* D*.*

Chaque brin connaît sa région d'appartenance, et chaque région conserve un brin de son bord extérieur, et une liste de brin, un par bord intérieur. Ces informations permettent de basculer entre la carte combinatoire proprement dite, et l'arbre d'inclusion afin de retrouver, par exemple, l'ensemble des régions incluses dans une région particulière. L'arbre d'inclusion est donc représenté uniquement par les différentes listes de brins représentant les éventuels bords intérieurs.

Lorsque nous avons un graphe topologique des frontières, définir la carte combinatoire correspondante, avec nos notations, se fait sans problème. En effet, contrairement au modèle précédent, nous avons simplement $\beta_1 = \sigma$ et $\beta_2 = \alpha$. Le TGF conserve la même permutation et la même involution que notre notation des cartes combinatoires, l'unique différence se situant au niveau de la représentation habituelle et non pas de la structure.

Nous pouvons voir un exemple de graphe topologique des frontières représentant la topologie de notre image segmentée en régions figure 3.4. L'involution α est représentée de manière induite, deux brins cousus par cette involution étant dessinés bout à bout, avec une petite barre en travers de l'arête. La permutation σ est représentée par les arcs de cercles orientés autour des régions. Nous pouvons voir que la région R_3 possède deux permutations distinctes, l'une pour représenter son bord extérieur : (f', d', c', g), et l'autre pour représenter son bord intérieur, composé uniquement



a. Une image segmentée en régions. b. Le TGF correspondant.

FIG. 3.4 – Une image 2d segmentée en régions et le graphe topologique des frontières correspondant.

d'un seul brin : (h), qui est donc cousu sur lui-même pour σ. En effet, il existe une seule et unique frontière entre R_3 et R_4, ce qui se traduit donc dans la carte combinatoire par un brin cousu sur lui-même. Si la région R_3 incluait d'autres régions, elle aurait alors autant de permutations distinctes que de composantes connexes totalement incluses, c'est-à-dire une par bord intérieur.

3.5 Conclusion

L'étude de ces quelques modèles nous a permis de mieux comprendre les différents problèmes que pose la définition d'un modèle de représentation d'image, et de voir quelles étaient les solutions habituellement utilisées afin de les résoudre. Nous en avons conclu que les modèles basés sur les cartes combinatoires semblaient être les plus intéressants à étudier afin d'en envisager l'extension en dimension supérieure.

En effet, les cartes combinatoires sont définies de manière formelle en dimension n, et sont utilisées depuis de nombreuses années en modélisation géométrique [Ber92, BDFL92, Lie97]. Même si les problématiques ne sont pas identiques, de nombreux points communs apparaissent et des thématiques connexes indiquent que nous avons tout intérêt à étudier les liens existants entre ces différents travaux.

De plus, les avantages des cartes combinatoires ou généralisées sont nombreux. Leur définition est simple et générique. Différents travaux ont défini des algorithmes de traitements efficaces et ont montré que ces algorithmes pouvaient de surcroît être écrits simplement. Enfin, bien que ces cartes soient uniquement un modèle topologique, il est facile de leur associer un modèle géométrique, point très important pour notre travail étant donné que nous voulons absolument traiter les deux modèles parallèlement.

Ces nombreux avantages nous ont conduit à nous intéresser en priorité aux deux derniers modèles présentés : les cartes discrètes et le TGF, qui sont deux modèles de représentation d'images 2d segmentées en régions basés sur les cartes combinatoires. Mais cette étude nous a amené à constater que la définition de ces deux modèles ainsi que les algorithmes permettant leurs extractions à partir d'une image étaient très complexes et techniques, et rendaient très difficile leurs extensions en dimension supérieure. Nous avons donc commencé à étudier ce problème de la dé-

finition d'un modèle de représentation d'images segmentées tout d'abord en dimension 2, afin de définir un nouveau modèle, ou de redéfinir de manière différente un modèle existant, en ayant comme préoccupation principale de pouvoir l'étendre en dimension supérieure.

CHAPITRE 4

LA CARTE TOPOLOGIQUE EN DIMENSION 2

Après avoir étudié plusieurs modèles de représentation d'images 2d segmentées en régions et en avoir déduit que les modèles présentant le plus d'avantages et étant le mieux adaptés à nos problématiques étaient le TGF ou son équivalent, les cartes discrètes, nous avons ensuite cherché à les étendre en dimension supérieure. Les principales difficultés que nous avons rencontrées proviennent du fait que les définitions du TGF et des cartes discrètes sont basées sur la notion de frontière, qui est définie en interpixel comme nous verrons section 4.1. La définition de frontière en dimension 2 ne pose pas de problème, car une frontière entre deux régions est simplement une courbe 1d que nous pouvons définir assez simplement.

Le problème est autrement plus complexe en dimension 3, où la frontière entre deux régions est une surface, i.e. un objet de dimension 2. Cela complique les différentes définitions, car il peut y avoir de nombreuses configurations différentes difficiles à prendre en compte. De plus, les cartes combinatoires représentent les objets par leurs bords. Il faut donc définir le bord des surfaces 2d afin d'avoir des courbes en dimension 1, qui peuvent être représentées avec des brins.

Une autre difficulté provient de la définition des différents algorithmes d'extraction. En effet, pour le TGF, Christophe Fiorio a défini un algorithme optimal d'extraction qui calcule le TGF d'une image de dimension 2 en une seule passe de l'image. Mais cet algorithme est complexe et difficilement extensible en dimension supérieure [BFP99]. Il en est de même pour le modèle de Jean-Philippe Domenger, qui est défini à partir des frontières interpixel, à l'aide d'un algorithme de suivi de contour. Ce suivi est facile à effectuer en dimension 2, mais beaucoup plus complexe en dimension 3, étant donné qu'il faut parcourir des surfaces afin de trouver leurs bords.

Pour définir la carte topologique en dimension 3, nous avons d'abord défini la carte topologique en dimension 2 de sorte que la définition soit simple et aisément extensible en dimension supérieure. De plus, nous avons conservé l'objectif principal : obtenir une structure minimale, caractéristique des objets qu'elle représente, et invariante par rotation, translation et homothétie. Nous cherchons également à définir un algorithme d'extraction facilement extensible en dimension supérieure.

Cette démarche nous a amené à définir la *carte topologique*, nom que nous donnons à la carte combinatoire représentant les frontières interpixel d'une image segmentée en régions, et

donc équivalente au TGF et aux cartes discrètes présentés au chapitre précédent. Pour cela, nous introduisons une notion de niveau de simplification qui permet de définir la carte topologique de manière progressive. Chaque niveau s'obtient à partir du niveau précédent par application d'un type particulier de fusion. Ce type de définition est aisément extensible en dimension n. De plus, cette démarche fournit un premier algorithme d'extraction de notre modèle étroitement lié à la définition de la carte topologique.

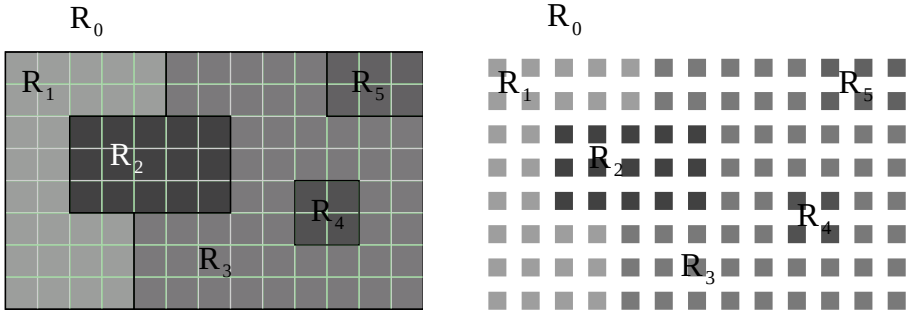
Puis nous avons cherché à optimiser cet algorithme « naïf » qui, bien que linéaire en le nombre de brins de la carte, effectue plusieurs passes sur celle-ci avant d'obtenir la carte topologique. De plus, cet algorithme a comme principal inconvénient de créer un grand nombre de brins, avant d'en détruire une bonne partie au cours des différentes passes de simplification. Nous avons pour cela, repris l'idée de l'algorithme optimal de Christophe Fiorio [Fio95, Fio96], qui consiste à balayer l'image de haut en bas et de gauche à droite avec une fenêtre de 2×2 pixels, en exécutant un traitement dépendant de la configuration locale des pixels de cette fenêtre. Nous avons adapté cette idée à nos divers niveaux de carte, et avons étudié les configurations à traiter pour chacun de ces niveaux.

L'idée générale de notre démarche est de calculer la carte de manière incrémentale et directe, au moyen d'un seul balayage de l'image. De plus, les traitements à effectuer dépendent uniquement de la configuration locale des pixels, et créent exactement le nombre de brins nécessaires. Les niveaux de simplification offrent ici un avantage important, car ils permettent de factoriser les cas pouvant se traiter de manière identique. En effet, nous étudions pour chaque niveau, les cas supplémentaires à traiter par rapport au niveau précédent. Au final, nous obtenons le nombre minimal de cas différents à traiter afin d'extraire la carte topologique. Ces regroupements ne sont pas très importants en dimension 2, étant donné qu'il existe seulement 15 cas différents au total. Mais nous verrons au chapitre 5 que ces regroupements sont cruciaux en dimension supérieure et permettent de définir l'algorithme optimal.

Nous commençons section 4.1 par présenter le cadre de ce travail, en rappelant les notions de segmentation en régions, d'interpixel, et celle de frontière interpixel. Section 4.2 nous introduisons la notion de niveau de simplification et définissons formellement la carte topologique au moyen de ces niveaux. Mais les cartes combinatoires ne représentent que la topologie des objets, et il faut leur adjoindre un modèle géométrique afin de coder toutes les informations des objets modélisés. La présentation de divers modèles géométriques et la manière de les associer aux différents niveaux de carte est l'objet de la section 4.3. Nous étudions ensuite section 4.4 le premier algorithme naïf d'extraction de nos diverses structures. Puis section 4.5 nous présentons l'algorithme optimal d'extraction. Section 4.6 nous présentons quelques résultats et effectuons une comparaison des diverses cartes obtenues avant de conclure ce chapitre section 4.7.

4.1 Images, segmentation en régions et interpixel

Nous considérons, pour notre processus d'extraction, que l'image a préalablement fait l'objet d'une segmentation en régions. Nous rappelons section 4.1.1 cette notion qui permet de préciser le cadre de ce travail. Nous rappelons également brièvement la notion d'*interpixel* section 4.1.2, ce qui permet de définir la notion de frontière, centrale dans ce travail.



a. Plongée dans le plan Euclidien. b. Représentée sous forme discrète.

FIG. 4.1 – Une image 2d segmentée en régions.

4.1.1 La segmentation en régions

Nous donnons ici quelques notations utilisées dans la suite de ce travail. Un *pixel* est un point coloré du plan discret $\mathbb{Z}^2$, et une *image* de dimension 2 est un ensemble fini de pixels. Nous utilisons les notions classiques de 4-*voisinage*, de chemin et de 4-*connexité*. Nous travaillons en 4-connexité car les cartes combinatoires ne permettent pas de représenter les non variétés, et donc le 8-voisinage.

La *segmentation en régions* d'une image I de dimension 2 est une partition de I en un ensemble de régions $\{R_1, R_2, \dots R_k\}$ telles que chaque région R_i est une composante 4-connexe.

Rappelons qu'un ensemble $\{R_1, R_2, \dots R_k\}$ est une *partition* d'un ensemble I s'il vérifie :
– $\cup_{i=1}^{k} R_i = I$;
– $\forall i, 1 \leq i \leq k, \forall j, 1 \leq j \leq k, i \neq j \Rightarrow R_i \cap R_j = \emptyset$.

Afin d'éviter un traitement particulier des pixels du bord de l'image, nous considérons que l'image est incluse dans une *région infinie* R_0 contenant l'ensemble des pixels n'appartenant pas à l'image. Ainsi nous avons des propriétés homogènes pour tous les pixels de l'image. La propriété principale, que nous utilisons par la suite, est que chaque pixel possède 4 voisins pour la 4-connexité. Cette région infinie permet également de traiter n'importe quel type d'image, et pas uniquement celles qui sont rectangulaires et sans trous.

La figure 4.1 montre un exemple d'image en dimension 2 segmentée en régions, tout d'abord plongée dans le plan Euclidien (figure 4.1.*a*), ou sous forme discrète (figure 4.1.*b*). Elle comporte cinq régions distinctes, plus la région infinie. Nous représentons sur nos figures chaque région par une couleur unique. Une notion importante et utile pour la suite de ce travail est l'inclusion de régions que nous définissons ci-après.

Définition 12 (inclusion) *Une région R_i est* incluse *dans une région R_j si et seulement si tout chemin 4-connexe allant d'un pixel de R_i vers un pixel de R_0 (la région infinie) possède au moins un pixel appartenant à la région R_j.*

Informellement, cette notion d'inclusion correspond au fait qu'une région en « encercle » complètement une autre, sans aucune contrainte sur la manière dont est effectué cet encerclement.

Cette définition, appliquée à l'exemple figure 4.1, indique que la région R_4 est incluse dans deux régions : R_3 et R_0. Toutes les autres régions sont, sans distinction, incluses uniquement dans la région infinie. Remarquons que chaque région est au moins incluse dans la région infinie. Cela est dû au fait que la relation d'inclusion est une relation d'ordre. En effet, elle est :

- **Réflexive** : chaque région r est incluse dans elle-même car tout chemin allant d'un pixel de r à un autre pixel de r possède forcément au moins un pixel de r ;
- **Transitive** : si $R_i \subseteq R_j$ et $R_j \subseteq R_k$, alors $R_i \subseteq R_k$. Cela peut se prouver en étudiant les chemins de R_i à R_0. Ils passent forcément par la région R_j, et les chemins de R_j à R_0 passent forcément par R_k, par définition de l'inclusion. Donc tous les chemins allant de R_i à R_0 traversent bien R_k ;
- **Antisymétrique** : si $R_i \subseteq R_j$, alors nous n'avons pas $R_j \subseteq R_i$ (pour $R_i \neq R_j$). Cela peut se prouver par l'absurde car si c'était le cas, tout chemin de R_i à R_0 passerait par R_j et tout chemin de R_j à R_0 passerait par R_i. Nous pouvons prouver dans ce cas que R_i n'est pas 4-connexe, ce qui contredit la définition de la segmentation en régions.

Cette notion d'inclusion sera utile afin de conserver la relation correspondante sur les régions, comme nous le verrons section 4.4.3. Pour cela, il n'est pas nécessaire de conserver toutes les relations d'inclusion. En effet, il suffit de conserver la réduction transitive de la relation d'ordre, les relations de transitivité pouvant alors aisément se retrouver à partir de cette nouvelle relation. Nous appelons cette relation l'*inclusion directe*.

Définition 13 (inclusion directe) *Une région R_i est* directement incluse *dans une région R_j si et seulement si $R_i \subseteq R_j$, et $\nexists R \neq R_j$ tel que $R_i \subseteq R$ et $R \subseteq R_j$.*

Sur notre exemple figure 4.1, la seule différence par rapport à la notion d'inclusion est pour la région R_4 qui est directement incluse uniquement dans R_3. Les autres régions étant incluses uniquement dans R_0, elles sont donc également directement incluses dans R_0.

Lorsqu'une image a été segmentée en régions, nous nous intéressons aux contours de celles-ci. Ces contours vont permettre de structurer les diverses relations topologiques entre les régions et d'obtenir ainsi le maximum d'informations utilisables pour des processus d'analyse ou de traitement. Ces relations topologiques se calculent en étudiant les contours des régions segmentées, et la manière dont ils se positionnent les uns par rapport aux autres.

Plusieurs travaux ont été réalisés autour de la notion de contours dans une image discrète. Il apparaît qu'utiliser une topologie basée sur la notion d'*interpixel* permet de définir correctement les contours de sorte qu'ils vérifient les propriétés topologiques habituelles (comme par exemple le théorème de Jordan [Ale61]). Nous présentons brièvement cette notion. Pour une définition complète on se réfèrera à [Kov89, KR89, KKM90, KKM91, Fio95].

4.1.2 Interpixel et frontières

La notion d'*interpixel* consiste à considérer une image non pas uniquement comme une matrice de pixels, mais comme une subdivision de l'espace (ici de dimension 2) en un ensemble de cellules de dimension inférieure ou égale à celle de l'espace. Les différentes cellules obtenues sont les *pixels*, cellules de dimension 2, les *lignels*, cellules de dimension 1 « entre » deux pixels, et les *pointels*, cellules de dimension 0 « entre » deux lignels. Nous appelons *i-cellule* une cellule de dimension i. Ces différentes cellules sont présentées figure 4.2.

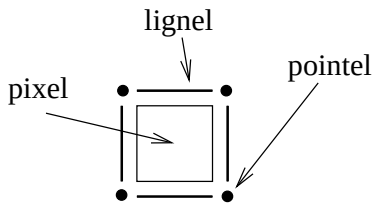


FIG. 4.2 – Les divers éléments interpixel en dimension 2.

La relation d'*incidence* permet de lier ces diverses cellules. Deux cellules sont dites *incidentes* si l'une appartient au *bord* de l'autre, et *adjacentes* si elles ont même dimension et sont incidentes à la même $(i-1)$-cellule. Remarquons que nous pouvons étendre cette notion d'adjacence en définissant que deux i-cellules sont j-adjacentes ($j < i$) si elles sont incidentes à la même j-cellule. Mais comme nous travaillons uniquement, pour un espace de dimension n, en $2n$ connexité, nous n'utilisons pas cette adjacence « étendue » et nous limitons donc à la première notion d'adjacence. Remarquons que ces notions sont les mêmes que celles présentées au chapitre 2, lors du rappel sur les cartes combinatoires. En effet, les cartes combinatoires représentent les subdivisions de l'espace, ce qui est également le cas d'une topologie interpixel. C'est pour cette raison qu'il est naturel de représenter les contours interpixel d'une image avec une carte combinatoire.

À l'aide de cette notion d'interpixel, nous pouvons définir de manière formelle la notion de courbe puis celle de frontière.

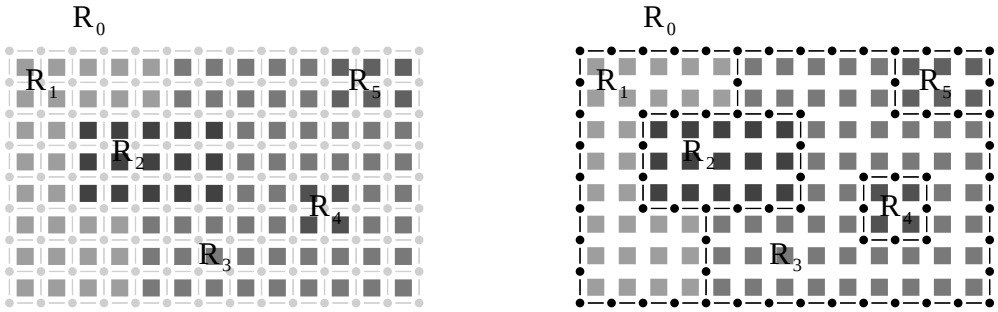
Définition 14 (courbe) *Une* courbe *est une suite de pointels et de lignels $c_0, \ldots, c_{2k}$ vérifiant :*
- $\forall i \in \{0, \ldots, k\}$, c_{2i} *est un pointel ;*
- $\forall i \in \{0, \ldots, k-1\}$, c_{2i+1} *est un lignel ;*
- $\forall i \in \{0, \ldots, 2k-1\}$, c_i *et* c_{i+1} *sont incidents.*

Une courbe est dite *simple* si toutes ses cellules sont distinctes[1]. Elle est dite *fermée* si $c_0 = c_{2k}$. Une *frontière* interpixel entre deux régions R_i et R_j est un ensemble de courbes simples disjointes deux à deux, tel que chaque lignel de ces courbes est incident à exactement un pixel de R_i et un pixel de R_j. Nous appelons *courbe frontière* une courbe appartenant à l'une des frontières de l'image. Ces courbes sont maximales : tout lignel de l'image incident à un pixel de R_i et un pixel de R_j appartient forcément à une courbe frontière, et deux lignels adjacents appartenant à une frontière appartiennent forcément à la même courbe frontière (ils sont séparés dans cette courbe par le pointel incident à ces deux lignels).

Figure 4.3.*b* est présenté l'ensemble des frontières interpixel de l'image figure 4.1. Nous y voyons que la frontière entre les régions R_1 et R_3 est composée de deux courbes distinctes, que celle entre R_3 et R_4 est composée d'une courbe fermée, et que celle entre R_1 et R_4 est vide car ces deux régions ne sont pas adjacentes.

Nous notons $frontière(R_i, R_j) = \{c_1, \ldots, c_l\}$ la frontière entre les régions R_i et R_j composée des courbes $c_1 \ldots c_l$. Nous considérons que $frontière(R_i, R_i) = \emptyset$. En effet, une région ne possède pas de frontière avec elle-même. Enfin, nous notons $Frontières(R_i)$ pour l'ensemble

[1]Une courbe est *simple* quand elle ne s'auto-intersecte pas.



a. L'image présentée figure 4.1 avec tous les éléments interpixel.

b. Les frontières interpixel des régions de cette image.

FIG. 4.3 – Frontières interpixel.

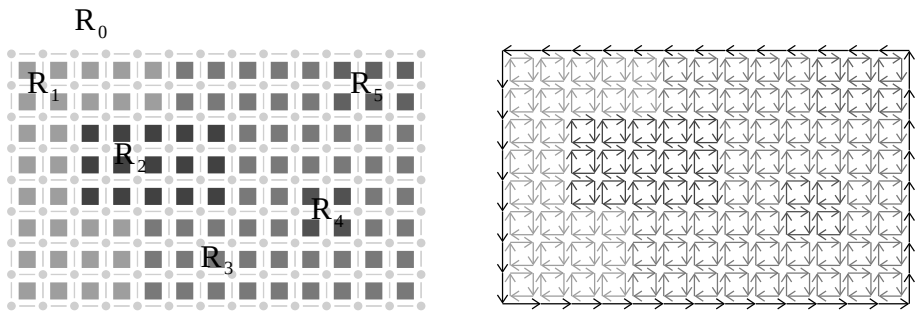
des frontières de la région R_i : $\cup_{j=1}^{k} frontière(R_i, R_j)$. Nous appelons respectivement *pointel frontière* et *lignel frontière* un pointel et un lignel appartenant à une frontière de l'image. Remarquons que cette définition des frontières reste valable pour les régions du bord de l'image grâce à la région infinie.

4.2 Les niveaux de simplification

Dans la section 4.1.2, nous avons introduit la notion de frontière interpixel. Il faut maintenant définir une structure de données représentant ces frontières ainsi que les diverses relations d'incidences entre les éléments de celles-ci. Pour cela, nous utilisons les cartes combinatoires qui sont bien adaptées au codage des frontières interpixel, car elles représentent les subdivisions de l'espace ainsi que leurs relations. De plus, elles définissent totalement la topologie des objets qu'elles représentent. C'est pour cette raison qu'elles ont été utilisées dans de nombreux travaux. Mais il existe plusieurs cartes combinatoires différentes représentant le même objet, et ces travaux ont utilisé diverses cartes suivant leurs besoins particuliers. De plus, définir formellement une carte combinatoire spécifique représentant les frontières interpixel d'une image est difficile et de ce fait souvent éludé.

Nous introduisons ici une notion de niveau de simplification qui permet de rapprocher ces divers travaux. Ces niveaux peuvent de plus se définir formellement à l'aide de définitions algorithmiques simples et progressives. Nous en présentons dans un premier temps seulement l'aspect topologique sans nous préoccuper du lien avec un modèle géométrique. Le lien avec un modèle géométrique est l'objet de la section 4.3.

L'idée de notre démarche est de construire la carte combinatoire complète, et de la simplifier petit à petit, tant que cela n'entraîne pas de perte d'information topologique. Ceci permet de définir progressivement la carte représentant les frontières interpixel de l'image, et ainsi d'en simplifier la définition.



a. L'image et ses éléments interpixel. b. La carte de niveau 0.

FIG. 4.4 – Le niveau 0 d'une image 2d.

4.2.1 Le niveau 0 : la carte complète

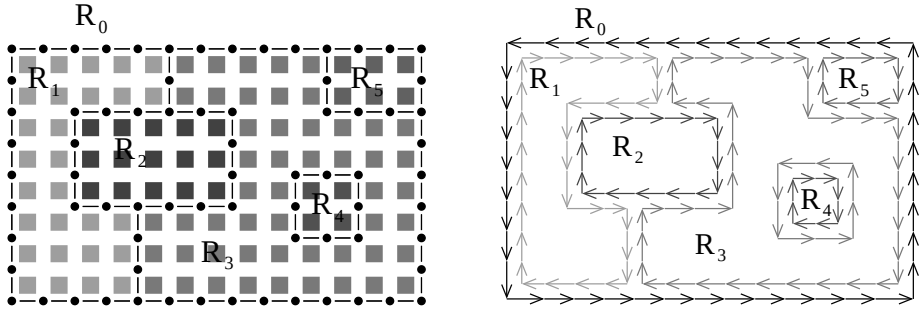
Cette première carte est seulement le point de départ de notre processus. Elle ne représente pas les frontières interpixel de l'image, et c'est pour cette raison que nous l'appelons le niveau 0.

Définition 15 (carte de niveau 0) *La* carte de niveau 0 *correspondant à une image de $n \times m$ pixels, est la carte combinatoire ayant $n \times m$ faces carrées, β_2-cousues entre elles lorsqu'elles sont adjacentes, chacune de ces faces représentant un pixel de l'image, et une autre face « entourant » ces faces représentant la région infinie.*

Cette carte représente tous les éléments interpixel de l'image à traiter. Nous voyons figure 4.4.*b* la carte de niveau 0 de l'image présentée figure 4.4.*a*. Cette carte est composée, pour une image de taille $n \times m$, de $(n \times m)+1$ faces. $n \times m$ faces carrées composées de 4 brins, chacune représentant un pixel de l'image. La face supplémentaire représente la région infinie et est composée de $2 \times (n+m)$ brins. Deux brins sont cousus par β_1 lorsqu'ils représentent deux lignels consécutifs de la même face, et sont cousus par β_2 lorsqu'ils représentent le même lignel de l'image. Remarquons que chaque face est orientée selon le sens trigonométrique inverse, à l'exception de la face infinie. En effet, les brins associés à cette face représentent un bord intérieur qui est donc orienté dans le sens trigonométrique. Nous représentons sur nos figures chaque brin de la couleur de la région auquel il appartient. Notons que cette carte est fermée, chaque brin est cousu par β_1 et β_2.

4.2.2 Le niveau 1 : la carte lignel

Afin d'obtenir une carte combinatoire codant les frontières interpixel de l'image, il faut « supprimer » toutes les arêtes de la carte de niveau 0 représentant des lignels n'appartenant pas à une courbe frontière. Cette opération s'effectue à l'aide de l'opération de fusion, qui modifie deux i-cellules adjacentes pour n'en obtenir plus qu'une. C'est cette opération de fusion qui permet de modifier la carte de niveau 0 afin d'obtenir la carte de niveau 1. Cette opération sera détaillée section 4.4 lors de la présentation de l'algorithme naïf d'extraction de la carte topologique, mais est également l'objet du chapitre 6 permettant de valider notre démarche.



a. L'image et ses frontières interpixel. b. La carte de niveau 1.

FIG. 4.5 – La carte de niveau 1 d'une image.

Définition 16 (carte de niveau 1) *La* carte de niveau 1 *est la carte obtenue à partir de la carte de niveau* 0 *en fusionnant chaque couple de faces adjacentes appartenant à la même région.*

Nous voyons figure 4.5.*a* l'image de notre exemple, représentée avec ses frontières interpixel, et figure 4.5.*b* la carte de niveau 1 correspondante. Nous pouvons facilement vérifier que cette carte représente bien les frontières interpixel de l'image.

Propriété 1 *Toute arête de la carte de niveau* 1 *correspond exactement à un lignel d'une courbe frontière, et tout lignel d'une courbe frontière est représenté dans la carte de niveau* 1 *par une arête.*

Preuve de la propriété 1

1. *Toute arête de la carte de niveau* 1 *correspond exactement à un lignel d'une courbe frontière :*
   Soit e *une arête de la carte de niveau* 1. e *est forcément composée de deux brins appartenant à deux régions différentes, sinon les deux faces incidentes auraient été fusionnées et* e *aurait été détruite lors de cette fusion. Cette arête est de longueur un, car toutes les arêtes de la carte de niveau* 0 *le sont, et nous n'avons pas modifié la longueur des arêtes entre ces deux niveaux.*
2. *Tout lignel d'une courbe frontière est représenté dans la carte de niveau* 1 *par une arête :*
   Soit l *un lignel d'une courbe frontière. Par définition des frontières,* l *est incident à deux pixels appartenant à deux régions différentes. Alors les deux faces correspondant à ces deux pixels ne sont pas fusionnées lors du passage du niveau* 0 *au niveau* 1 *et nous avons donc dans la carte de niveau* 1 *une arête représentant ce lignel.*

□

C'est cette propriété qui nous a amené à appeler cette carte de niveau 1 la *carte lignel*. De plus, nous appelons cette carte le niveau 1, car c'est la première carte, et la plus grande en termes de nombre de brins, représentant les frontières interpixel de l'image. En effet, pour ajouter des brins à cette carte, en représentant uniquement les frontières interpixel, il faut représenter un même lignel par plusieurs arêtes. C'est bien entendu réalisable théoriquement (et dans ce cas il n'y a pas de plus grande carte, nous pouvons ajouter un nombre infini d'arêtes) mais peu intéressant en pratique.

Propriété 2 *Soit C une carte lignel codant une image I segmentée en régions. Chaque région (exceptée la région infinie) est représentée dans C par un contour extérieur, et éventuellement par un ou plusieurs contours intérieurs.*

Chaque région possède un contour extérieur, exceptée la région infinie R_0 qui possède uniquement un contour intérieur. Les éventuels contours intérieurs surviennent lorsqu'une région est incluse dans une autre. Nous pouvons voir sur l'exemple figure 4.5.*b*, que seule la région R_3 possède un contour intérieur (si l'on excepte la région infinie). Nous avons alors déconnecté la carte en plusieurs composantes connexes (ici deux), et nous avons perdu l'information reliant les deux bords maintenant déconnectés. Il faut conserver cette information permettant de relier le bord extérieur d'une région et ses éventuels bords intérieurs, car c'est une information topologique importante, permettant de représenter les « trous » dans une région. Pour cela, nous introduisons un arbre d'inclusion des régions. Cet arbre possède un nœud par région de l'image. Sa racine est la région infinie, et chaque nœud possède comme fils les régions directement incluses dans la région correspondant à ce nœud. À l'aide de cet arbre, nous pouvons retrouver les régions incluses dans n'importe quelle région et donc reconstruire, ou parcourir correctement son bord.

La carte lignel, et plus précisément les contours des régions dans cette carte, vérifie plusieurs propriétés intéressantes :

1. la carte est fermée ;
2. la région infinie est représentée par un unique contour intérieur ;
3. les contours extérieurs sont orientés dans le sens trigonométrique inverse ;
4. les contours intérieurs sont orientés dans le sens trigonométrique ;

Les propriétés 1 à 3 peuvent se démontrer facilement, car elle sont vraies par construction pour le niveau 0, et nous pouvons prouver qu'elles restent vraies après une fusion de faces. Elle sont dont également vraies pour la carte lignel, de par sa construction. La propriété 3 revient à dire que lorsque nous parcourons un contour extérieur dans le sens trigonométrique inverse, la région correspondante se trouve tout le temps à droite du contour. Par contre, c'est l'inverse pour les contours intérieurs : si nous parcourons ces contours dans le sens trigonométrique inverse, la région correspondante se trouve à gauche. C'est pour cette raison que les contours intérieurs sont orientés en sens inverse des contours extérieurs, donc dans le sens trigonométrique. Avec cette orientation, pour n'importe quel contour, la région se trouve tout le temps à droite. Cela explique la propriété 4 et donne une idée de preuve. Remarquons que le sens des contours extérieurs et intérieurs a été choisi par convention, fixé lors de la construction de la carte de niveau 0.

Cette carte lignel dotée d'un arbre d'inclusion contient toutes les informations topologiques de l'image qu'elle représente. Nous venons de voir comment est représentée la relation d'inclusion, au moyen de l'arbre d'inclusion des régions. Cette carte code toutes les 1-cellules et 0-cellules des frontières des régions de l'image. Les 2-cellules ne sont plus représentées, mais nous représentons maintenant les régions (qui sont des ensembles de 2-cellules) au moyen de leurs frontières. Les relations d'adjacences et d'incidences entre n'importe quelles cellules sont toutes représentées dans la carte, de manière explicite ou implicite, de par la définition des cartes combinatoires. Nous nous appuyons sur cette remarque, afin de conserver toutes les informations topologiques au cours de nos différentes simplification. Pour cela, il suffit de montrer, lorsque nous effectuons une simplification, que nous ne perdons pas d'information topologique.

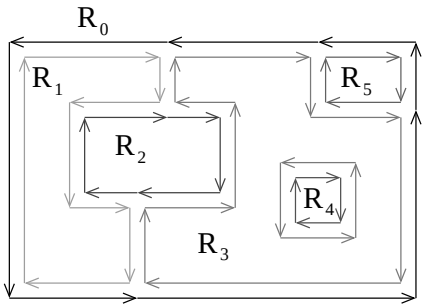


FIG. 4.6 – La carte de niveau 2 de notre exemple.

Cette carte lignel est donc la première carte représentant les frontières interpixel d'une image segmentée en régions. Mais cette représentation n'est optimale ni en espace mémoire ni en temps d'exécution. En effet, de par le grand nombre de brins à parcourir, par exemple pour retrouver toutes les régions adjacentes à une région, les opérations sur cette carte sont coûteuses en temps d'exécution. De plus cette carte n'est pas caractéristique de la topologie des images qu'elle représente, n'est pas stable par rotation, translation et homothétie, et n'est pas minimale en nombre de brins. C'est pour ces raisons que nous allons simplifier cette carte et définir la carte de niveau 2.

4.2.3 Le niveau 2 : la carte des frontières

Cette *carte des frontières* a été présentée dans [BFP99]. C'est la carte la plus intuitive représentant les frontières interpixel d'une image. Dans cette carte, chaque arête représente une « portion » rectiligne de frontière. La définition formelle de cette carte s'obtient aisément à partir de la carte lignel qui est le niveau précédent. Nous appelons indifféremment cette carte la carte de niveau 2, car c'est la deuxième carte que nous définissons qui représente les frontières de l'image, ou la carte des frontières qui est le nom original donné dans [BFP99].

Définition 17 (carte de niveau 2) *La* carte de niveau 2 *est la carte obtenue à partir de la carte de niveau* 1*, en fusionnant chaque couple d'arêtes adjacentes, alignées et incidentes à un sommet de degré deux.*

La carte des frontières de l'image figure 4.5.*a* est présentée figure 4.6. La propriété principale de cette carte est que chaque arête représente un segment de droite des frontières interpixel de l'image. Ces arêtes sont désormais de longueur quelconque, contrairement à la carte lignel.

Propriété 3 *Chaque arête d'une carte des frontières correspond à une suite maximale de lignels alignés d'une frontière.*

Cette propriété est évidente, de par la définition de la carte des frontières. En effet, dans la carte de niveau 1, chaque arête correspond à un lignel, et donc lorsque deux lignels sont alignés, et qu'ils sont incidents à un sommet de degré deux, les arêtes correspondantes sont fusionnées

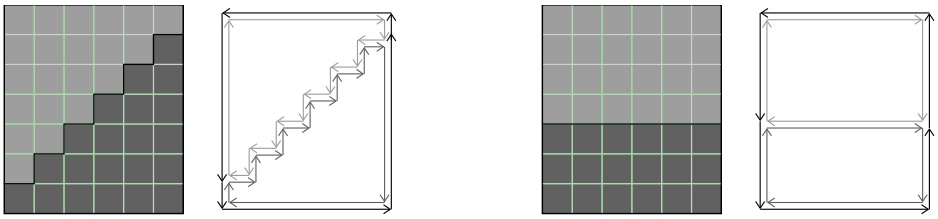


FIG. 4.7 – Deux images topologiquement équivalentes et leurs cartes des frontières.

pour obtenir la carte de niveau 2. Cette carte ne peut donc pas avoir deux arêtes représentant deux lignels adjacents alignés étant au milieu d'une frontière (ce qui revient à « incident à un sommet de degré deux »). De manière symétrique, si deux arêtes sont adjacentes dans la carte des frontières, alors soit elles sont incidentes à un sommet de degré supérieur à deux, soit elles représentent deux portions de frontières non alignées.

Mais le point le plus important, afin de valider notre démarche, est de garantir l'absence de perte d'information topologiques par rapport à la carte lignel.

Propriété 4 *La carte des frontières représente exactement les mêmes relations topologiques que la carte lignel.*

Cela peut se prouver facilement. La seule modification entre ces deux niveaux, est que nous avons fusionné chaque couple d'arêtes alignées incidentes à des sommets de degré deux. Lorsque deux arêtes sont incidentes à un sommet de degré deux, elles sont alors forcément toutes les deux entre les deux mêmes régions R_i et R_j. Dans ce cas, les deux arêtes représentent la même information topologique d'adjacence entre ces deux régions, et le fait de n'en conserver qu'une seule n'entraîne aucune perte d'information. La preuve formelle de la validité des réécritures entre les différents niveaux est l'objet du chapitre 6 dans le cadre plus général de la dimension 3.

En comparant, sur notre exemple, la carte lignel avec la carte des frontières, nous observons que le nombre de brins a considérablement diminué, en passant de 276 brins à 44. Cette baisse importante entraîne évidemment un gain important en espace mémoire, mais également en temps d'exécution, pour les raisons évoquées lors de la présentation de la carte lignel.

Mais ce niveau n'est pas totalement satisfaisant. En effet, le nombre de brins nécessaires pour représenter une frontière entre deux régions dépend de la géométrie de celle-ci. Dans le pire des cas, si une courbe frontière ne possède aucun lignel consécutif aligné (cas de « l'escalier » figure 4.7), le nombre de brins représentant cette frontière sera égal à deux fois le nombre de lignels de celle-ci. Cela a pour conséquence que deux images topologiquement équivalentes peuvent avoir deux cartes des frontières totalement différentes.

Notre but étant d'extraire et de structurer les informations pertinentes des images, il nous semble primordial d'avoir, pour deux images topologiquement équivalentes, deux cartes combinatoires isomorphes. De plus, dans la carte des frontières, il subsiste des arêtes adjacentes incidentes à des sommets de degré deux. Comme pour la carte lignel, ces arêtes représentent la même information topologique, et peuvent donc être simplifiées. La carte des frontières n'est donc pas

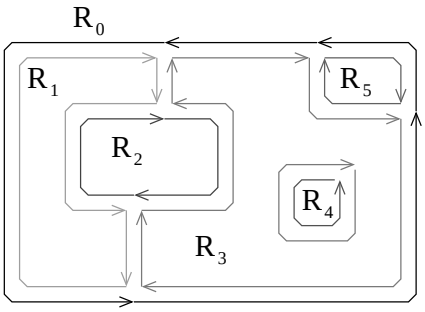


FIG. 4.8 – La carte de niveau 3 de notre exemple.

minimale. Nous définissons donc la carte topologique, qui est la carte de niveau 3, afin d'avoir une structure complètement homogène mais également moins coûteuse en espace mémoire.

4.2.4 Le niveau 3 : la carte topologique

Afin de définir ce dernier niveau de carte, nous conservons notre principe de simplification. Nous partons de la carte des frontières et fusionnons les derniers éléments représentant des informations topologiques de manière redondante : les arêtes adjacentes incidentes à un sommet de degré deux.

Définition 18 (carte de niveau 3) *La* carte de niveau 3 *est la carte obtenue à partir de la carte de niveau* 2 *en fusionnant chaque couple d'arêtes adjacentes incidentes à un sommet de degré deux.*

Nous voyons figure 4.8 la carte de niveau 3 correspondant à notre image d'exemple. La propriété principale de la carte de niveau 3 peut se vérifier sur cet exemple.

Propriété 5 *Chaque arête d'une carte de niveau* 3 *correspond exactement à une courbe frontière de l'image correspondante.*

C'est cette propriété qui nous a amené à appeler cette carte *carte topologique*. En effet, cette carte est minimale, nous ne pouvons plus simplifier quoi que ce soit sans entraîner une perte d'information topologique. C'est bien notre but final : la carte combinatoire minimale représentant les frontières interpixel d'une image. Cette propriété peut se reformuler en : *chaque arête de la carte topologique est incidente à deux sommets de degré strictement supérieur à deux (à l'exception des boucles)*. Cela peut se prouver facilement à l'aide de la définition de cette carte. En effet, toutes les arêtes adjacentes incidentes à des sommets de degré deux ont été fusionnées, et il n'en reste donc plus. Cette propriété nous permet de calculer le nombre de brins de cette carte : il est égal à deux fois le nombre de courbes frontière de l'image correspondante. Nous pouvons observer sur la figure 4.8 que la frontière entre les régions R_3 et R_4 est représentée par une boucle. En effet, la région R_4 est incluse dans R_3 et ne possède pas d'autre région adjacente. Le sommet incident à cette boucle est de degré 1, ce qui permet de conserver la boucle dans la carte de niveau 3 étant donné que nous supprimons uniquement les sommets de degré 2.

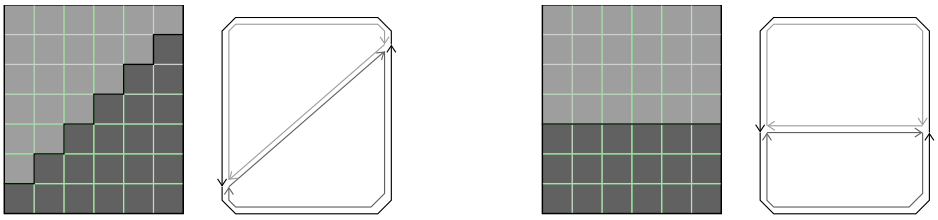


FIG. 4.9 – Deux images topologiquement équivalentes et leurs cartes topologiques qui sont isomorphes.

Pour ce dernier niveau, nous associons à chaque courbe frontière exactement une arête de la carte. La carte obtenue est alors totalement homogène et deux images topologiquement équivalentes seront représentées par deux cartes isomorphes, comme nous pouvons le voir figure 4.9. De plus, cette carte est moins coûteuse en espace mémoire que les précédentes, tout en représentant exactement les mêmes informations topologiques.

Propriété 6 *La carte des frontières représente exactement les mêmes relations topologiques que la carte lignel.*

Pour nous assurer de cette propriété, nous utilisons le même principe que pour la carte des frontières, en montrant que deux arêtes incidentes à un sommet de degré deux représentent la même information topologique. Nous pouvons donc ne conserver qu'une des deux arêtes sans entraîner de perte d'information. Nous voyons sur l'exemple figure 4.8, qu'il ne reste plus que 20 brins qui représentent les 10 courbes frontières de l'image. C'est peu en comparaison des 276 brins de la carte lignel de la même image.

Avec la carte topologique, nous avons codé chaque courbe frontière de manière minimale (une arête) et nous ne pouvons plus « simplifier » la carte obtenue sans perte d'information topologique. Cette carte est donc minimale, stable par rotation, translation et homothétie, et représente bien toutes les informations topologiques de l'image.

Remarquons enfin que la définition de la carte topologique n'utilise pas la notion de frontière. Ce point est très important, car c'est ce qui rend cette définition simple. Le seul lien entre la carte combinatoire et l'image segmentée en régions que nous voulons représenter, est effectué lors de la construction du niveau 1 à partir du niveau 0. En effet, pour construire ce niveau, nous fusionnons les faces adjacentes appartenant à la même région, information que nous trouvons dans l'image segmentée. La construction des autres niveaux n'est plus du tout liée à l'image, mais uniquement à la carte et aux différentes cellules de celle-ci. De ce fait, les notions de frontières interpixel et de courbes frontières ne sont pas nécessaires pour définir la carte topologique, mais uniquement pour prouver qu'elle représente correctement les informations de l'image. Ceci est important pour l'extension de notre modèle en dimension supérieure, et permettra de définir la carte topologique sans se préoccuper de la définition des surfaces frontières dans un espace de dimension 3.

4.3 Modèles de plongement et niveaux de simplification

Les cartes combinatoires codent la topologie des objets qu'elles représentent par subdivision de l'espace qu'ils occupent. Mais dans la plupart des applications, représenter uniquement la topologie ne suffit pas, il faut également représenter la géométrie des objets modélisés. Afin d'associer un modèle géométrique à notre modèle topologique, il est possible d'associer à chaque cellule topologique un objet géométrique de même dimension.

Mais dans la plupart du temps, pour des applications spécifiques, l'existence de liens entre le modèle topologique et le modèle géométrique rend inutile le plongement de chaque cellule. Nous étudions ici quelques modèles de plongements, pour chaque niveau de carte. Ces plongements sont les plus couramment utilisés, mais des applications spécifiques nécessiteront certainement un plongement particulier, qui pourrait alors s'inspirer ou être une composition de certains plongements présentés ici.

4.3.1 La carte lignel

Dans cette carte, chaque brin représente un lignel des frontières interpixel de l'image. Un premier plongement, qui découle immédiatement de cette propriété, consiste à associer à chaque brin le code de Crack[2] [BF70] correspondant. Il suffit alors de conserver les coordonnées du pointel en haut à gauche de chaque composante connexe de la carte pour pouvoir retrouver les coordonnées de chaque pointel, et pouvoir ainsi reconstruire la géométrie des frontières. Le principal avantage de ce plongement est qu'il est très peu coûteux en espace. Il nécessite seulement 8 octets par composante connexe de la carte (pour les coordonnées du pointel en haut à gauche), ce qui est négligeable, plus 2 bits par brin pour le code de Crack. En effet, les lignels étant forcément horizontaux ou verticaux, 2 bits suffisent pour ce codage. L'inconvénient majeur de ce plongement est que pour récupérer les coordonnées d'un pointel particulier, nous devons parcourir, au pire des cas, toute la composante connexe afin de trouver les coordonnées du point de départ. Pour les opérations agissant sur toute la carte (comme l'affichage complet) ce n'est pas coûteux car nous pouvons commencer le parcours par ces points particuliers. Par contre pour des opérations locales (comme le calcul du périmètre d'une région) ce plongement entraîne un surcoût en temps rédhibitoire. En effet, dans ce cas il faut parcourir toute la composante connexe avant de pouvoir calculer ce périmètre.

Un autre plongement, qui contrairement au précédent privilégie le temps d'exécution au détriment de l'espace, consiste à associer à chaque sommet topologique de la carte lignel les coordonnées du pointel correspondant. Ce plongement est beaucoup plus coûteux en espace mémoire que le précédent, mais les opérations pourront s'effectuer de manière minimale en nombre de brins à parcourir. Suivant nos besoins particuliers, nous choisirons l'un ou l'autre plongement. Il est également envisageable d'utiliser un plongement hybride, où certains sommets topologiques seraient associés aux coordonnées des pointels correspondant, et chaque brin serait associé au code de Crack du lignel correspondant. Cette solution permettrait de diminuer la complexité moyenne des opérations, en augmentant légèrement l'espace nécessaire.

[2] Le code de Crack est la restriction du code de Freeman à seulement quatre directions afin de représenter les contours interpixel d'une région.

4.3.2 La carte des frontières

Pour cette carte, nous utilisons sa principale propriété afin de définir un plongement efficace, comme pour la carte lignel. Nous avons vu à la section précédente que chaque arête de cette carte correspond à la suite maximale de lignels alignés d'une frontière. Cela permet de ne pas plonger les arêtes. En effet, pour chaque arête de la carte, nous pouvons retrouver le plongement de ses deux sommets extrémités, et reconstruire le plongement de cette arête qui est simplement le segment de droite défini par ces deux sommets. Pour ce premier plongement, nous associons simplement à chaque sommet de la carte des frontières les coordonnées du pointel correspondant. Ce plongement est peu coûteux en espace mémoire en comparaison à l'espace utilisé pour le codage de la carte elle-même. En effet le nombre de sommets est en moyenne égal à un tiers du nombre de brins [BF97], et le codage des coordonnées d'un sommet demande moins d'espace que le codage d'un brin. De plus, il est simple à mettre en œuvre étant donné que nous avons un seul type de cellule à plonger. Enfin, retrouver le plongement des arêtes ou des faces se fait sans aucune difficulté ni surcoût en complexité, en parcourant les brins et en récupérant les coordonnées de chaque sommet rencontré.

La simplicité de ce plongement, et le fait qu'il est naturel[3] font qu'il est souvent utilisé. Un autre plongement moins coûteux en espace consisterait à conserver les coordonnées du pointel en haut à gauche de chaque composante connexe, et à associer à chaque brin une direction et une longueur. Mais le gain en espace mémoire n'est pas très important, alors que la complexité des opérations augmente, comme nous l'avons vu pour la carte lignel.

4.3.3 La carte topologique

Pour ce dernier niveau, nous ne pouvons plus retrouver le plongement d'une arête à partir du plongement de ses sommets extrémités. En effet, les arêtes de cette carte topologique peuvent maintenant avoir un plongement quelconque et ne sont plus uniquement des segments de droite.

Un premier plongement consiste à associer à chaque arête la courbe de dimension 1 représentant la frontière interpixel correspondante. Ce plongement a comme principal avantage d'être simple car, comme pour la carte des frontières, un seul type de cellule est plongé. Nous avons représenté figure 4.10 la carte topologique de notre exemple plongée de cette manière. Sur cette figure, nous avons numéroté les brins de 1 à 20. Nous associons à chaque arête une courbe de dimension 1 qui représente le plongement de la frontière correspondante. Nous avons représenté ces courbes par une liste de sommets, qui peuvent être codés de plusieurs manières différentes. Une solution consiste à les représenter par des 1-cartes, la structure obtenue est alors hiérarchique. Dans ce cas, ces courbes de dimension 1 doivent être orientées dans le même sens que les brins qui les désignent, afin de pouvoir, par exemple, reconstruire correctement le plongement d'une face en récupérant la suite des plongements des arêtes qui la compose.

Pour représenter l'image de la figure 4.3, chaque sommet doit être associé aux coordonnées du pointel correspondant dans l'image. Mais plusieurs sommets ont les mêmes coordonnées. Ce sont les sommets géométriques associés au même sommet topologique de la carte. Par exemple, les brins $\{2, 9, 19\}$ représentent un sommet topologique de la carte. Les courbes 1d associées à

[3] Quelqu'un ne connaissant pas les cartes combinatoires dessinera plus naturellement les arêtes comme des segments de droite que comme des courbes quelconques.

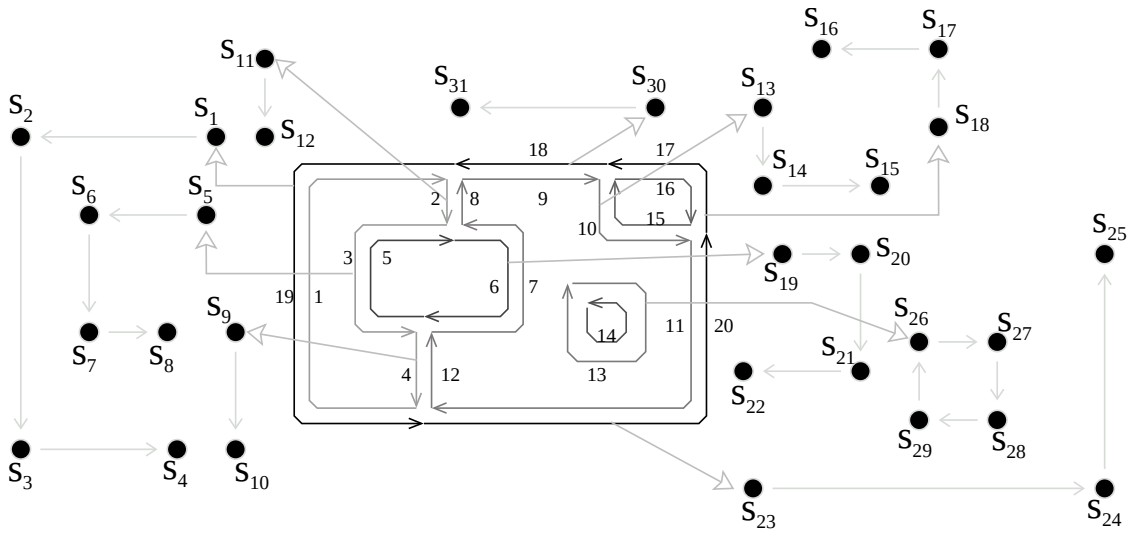


FIG. 4.10 – La carte topologique et son plongement arête.

chaque arête incidente à un de ces brins auront toutes les coordonnées de ce sommet en commun. Cela se vérifie sur notre exemple, figure 4.10, où les sommets s_1, s_{11} et s_{31} ont tous les mêmes coordonnées puisqu'ils représentent le même pointel.

De manière générale, les sommets extrémités de chaque courbe frontière sont dupliqués autant de fois qu'il y a d'arêtes incidentes à ce sommet dans la carte topologique. Cette redondance est non seulement coûteuse en espace, mais également en temps d'exécution lors des mises à jour des plongements. En effet, si nous voulons modifier les coordonnées d'un sommet, il faut le faire pour tous les plongements des arêtes incidentes à ce sommet.

Afin d'éviter cette duplication, nous définissons un deuxième type de plongement pour lequel nous plongeons les sommets ainsi que les arêtes de la carte topologique (cf. figure 4.11). Chaque sommet sera plongé par les coordonnées du pointel correspondant, et chaque arête par la courbe de dimension 1 correspondant à la frontière interpixel associée, moins ses extrémités. Nous parlons abusivement de *courbe ouverte* bien que ce ne soit pas exactement une courbe ouverte au sens topologique du terme. L'avantage de ce plongement est de ne pas coder de manière redondante les coordonnées des sommets extrémités des courbes frontières. Son inconvénient est que lorsque nous voulons récupérer le plongement d'une courbe frontière, il faut le reconstruire en utilisant les plongements des sommets incidents à l'arête correspondant à cette courbe frontière et le plongement de l'arête. Mais cette reconstruction peut s'effectuer en temps linéaire et n'est donc pas pénalisante par rapport à la carte des frontières.

Ce plongement est présenté figure 4.11 sur notre image d'exemple. Cette carte a sept sommets topologiques (plongés par les sommets s_1, s_4, s_5, s_8, s_{11}, s_{13} et s_{15}), et dix arêtes topologiques dont trois n'ont pas de plongement. En effet, lorsque le plongement d'une arête est un segment de droite, comme c'est le cas sur notre exemple pour les trois arêtes $\{2, 8\}$, $\{4, 12\}$ et $\{9, 18\}$, la courbe 1d correspondante privée de ses extrémités ne contient plus aucun sommet. Pour reconstruire le plongement d'une arête incidente à un brin b, il suffit de récupérer, dans un premier temps,

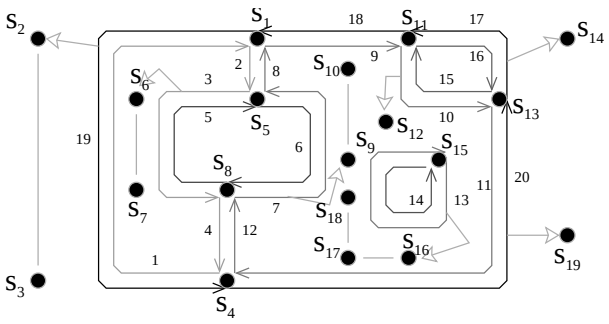


FIG. 4.11 – La carte topologique et son plongement sommet et arête « ouverte ».

le plongement du sommet incident à ce brin, puis le plongement de l'arête, et enfin le plongement du sommet incident au brin $\beta_2(b)$. La courbe 1d constituée de la concaténation, dans cet ordre, des sommets obtenus est le plongement de l'arête.

Sur notre exemple, afin de récupérer le plongement de l'arête incidente au brin 19, nous récupérons le plongement du sommet incident à ce brin, c'est s_1, puis les sommets du plongement arête, s_2 et s_3, et enfin le plongement du sommet incident au brin 1 (qui est $\beta_2(19)$), c'est s_4. Le plongement de l'arête est donc la courbe 1d constituée des sommets $[s_1, s_2, s_3, s_4]$. Comme pour le plongement précédent, les courbes de dimension 1 doivent être orientées correctement, c'est-à-dire dans le même sens que le brin « pointant » sur cette courbe, afin que la reconstruction soit valide. Ce plongement est moins coûteux en espace mémoire que le précédent, et la complexité en temps des opérations est identique, voire moins coûteuse. Par exemple l'opération de modification des coordonnées d'un sommet de la carte est, pour ce plongement, moins coûteuse que pour le plongement précédent. C'est ce plongement que nous utilisons par la suite, dans les différents algorithmes travaillant sur la carte topologique.

4.4 Un premier algorithme d'extraction

Après avoir défini les trois niveaux de carte combinatoire et les plongements qui leur sont associés, nous allons nous intéresser à la manière de construire ces diverses cartes à partir d'une image segmentée en régions.

Nous présentons ici un premier algorithme « naïf » d'extraction. Cet algorithme a comme principal avantage d'être simple, puisqu'il découle immédiatement des définitions de nos différents niveaux de carte. En effet, ces définitions sont algorithmiques, et définissent chaque niveau de carte à partir du niveau précédent et d'un type particulier de fusion. Pour définir totalement cet algorithme, nous devons donc nous intéresser aux fusions dans les cartes combinatoires.

4.4.1 Les fusions dans les 2-cartes

Une fusion en dimension i consiste à modifier deux objets adjacents de dimension i pour en obtenir un seul constitué de l'union des deux objets. Nous pouvons voir deux exemples de fusion

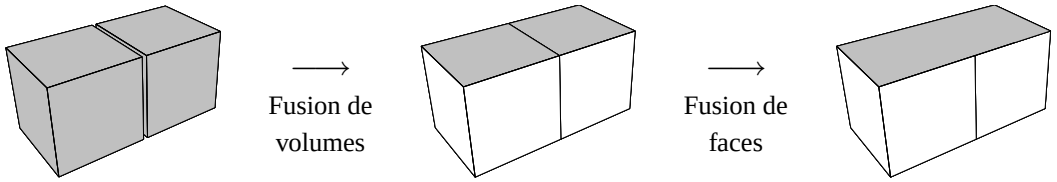
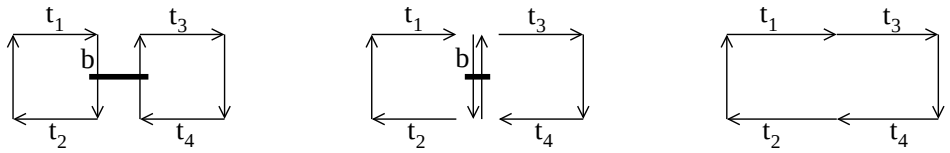


FIG. 4.12 – Deux exemples de fusions.



a. Avant la fusion.

b. β_0 et β_1 décousures des brins centraux.

c. β_1 couture des brins décousus.

FIG. 4.13 – Fusion de faces en 2d.

figure 4.12. Pour la dimension n, il existe exactement n fusions possibles qui sont les fusions des dimensions de $1 \dots n$. Ces opérations de fusions sont simples à écrire dans le cadre des cartes combinatoires, et ce pour n'importe quelle dimension. Afin que cette opération de fusion puisse s'effectuer correctement, la configuration des cellules que nous voulons fusionner est contrainte.

Lorsque nous voulons fusionner deux i-cellules adjacentes, C_1 et C_2, nous parlons de fusion le long de la $(i-1)$-cellule incidente à C_1 et C_2 que nous appelons ici C. Pour que la fusion puisse être appliquée, C doit être de degré 1 ou 2. Intuitivement, la fusion consiste à supprimer cette cellule. Lors de cette suppression, les deux cellules C_1 et C_2 sont modifiées pour n'en faire plus qu'une, mais s'il existe d'autres cellules incidentes à C, leurs relations d'incidences ne seront alors plus valides. Le cas où C est de degré 1, revient en dimension 2 à effectuer la fusion d'une face le long d'un isthme (figure 4.14).

Remarquons que nous considérons ici la fusion de « base », consistant à fusionner deux cellules C_1 et C_2 uniquement le long d'une seule cellule. Il existe des fusions plus évoluées, permettant de fusionner deux cellules qui possèdent des multi-adjacences en une seule opération. Mais cette fusion de haut niveau peut être réalisée par un ensemble de fusions simples, et n'est pas utile ici.

En dimension 2, il existe deux types différents de fusions qui sont la fusion de faces (dimension 2) et la fusion d'arêtes (dimension 1). La figure 4.13 illustre le déroulement de l'algorithme 1 effectuant la fusion de deux faces.

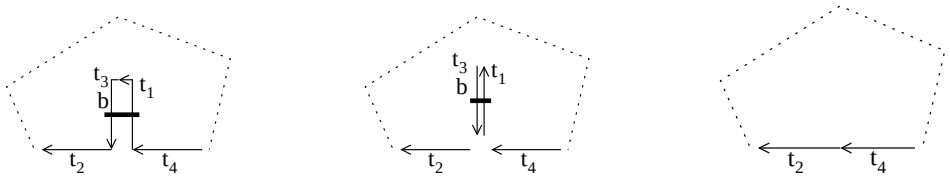
Cet algorithme prend en entrée un brin b, et fusionne la face incidente à ce brin avec la face incidente au brin $\beta_2(b)$ (c'est pour cette raison que b doit nécessairement être cousu par β_2). Il découd l'arête incidente au brin b par β_0 et β_1 (figure 4.13.*b*), puis coud les brins initialement cousus à cette arête (figure 4.13.*c*). Remarquons qu'aucun traitement ne met à jour les éventuels plongements car les sommets topologiques conservent les mêmes coordonnées avant et après la fusion, et les éventuels plongements arêtes ne sont pas modifiés.

Algorithme 1 Fusion de faces en 2d

Entrée : Un brin b β_2 cousu

Résultat : Les deux faces incidentes à b et à $\beta_2(b)$ sont fusionnées.

$t_1 \leftarrow \beta_0(b)$; $t_2 \leftarrow \beta_1(b)$;
$t_3 \leftarrow \beta_{21}(b)$; $t_4 \leftarrow \beta_{20}(b)$;
β_0-découdre(b) ; β_1-découdre(b);
β_0-découdre($\beta_2(b)$) ; β_1-découdre($\beta_2(b)$);
β_1-coudre(t_1, t_3) ; β_1-coudre(t_4, t_2);



a. Avant la fusion. b. β_0 et β_1 décousures des brins centraux. c. β_1 couture des brins décousus.

FIG. 4.14 – La fusion de faces en 2d pour un cas dégénéré.

La seule modification éventuelle du plongement est réalisée de manière transparente, lors des coutures et décousures. En effet, chaque sommet topologique doit être associé à exactement un sommet géométrique. L'opération de décousure duplique un sommet géométrique si elle entraîne la déconnexion d'un sommet topologique, et réciproquement l'opération de couture détruit un sommet géométrique si elle regroupe deux sommets topologiques distincts. Dans ce dernier cas, afin de pouvoir contrôler quel est le sommet détruit, nous fixons par convention que c'est toujours le sommet appartenant au sommet topologique du deuxième brin passé en paramètre de la fonction de couture.

Nous pouvons remarquer que cet algorithme effectue correctement la fusion de faces quelles que soient leurs configurations, même dans les cas « dégénérés », par exemple pour la configuration présentée figure 4.14. Cet exemple montre la fusion de faces le long d'une arête de degré un (car $\beta_0(b) = \beta_2(b)$). Nous pouvons vérifier que même dans ce cas la fusion de faces par l'algorithme 1 ne pose aucun problème. En effet, les coutures et décousures s'effectuent uniquement de manière locale, sans aucune contrainte sur la manière dont les brins doivent être cousus, à condition que les brins b et $\beta_2(b)$ soient différents.

Comme pour la fusion de faces, la fusion d'arêtes incidentes à un sommet de degré deux se fait de manière simple comme nous pouvons le voir sur l'algorithme 2.

La figure 4.15.*a* présente un exemple de carte avant la fusion d'arêtes, et le résultat de cette fusion figure 4.15.*b*. Comme pour la fusion de faces, l'algorithme de fusion d'arêtes fonctionne quelle que soit la configuration des brins, et quelle que soit la géométrie des arêtes.

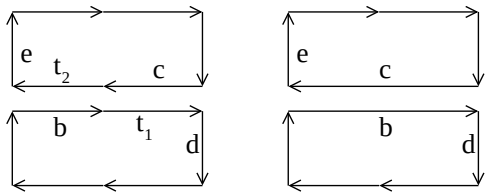
Le fait de conserver le brin b ainsi que le brin $c = \beta_{20}(b)$ pour la nouvelle arête, évite toute modification de plongement sommet. En effet, chaque sommet topologique conserve le même plongement avant et après la fusion car nous ne modifions pas l'orbite des deux sommets incidents

Algorithme 2 Fusion d'arêtes en 2d

Entrée : Un brin b β_1 cousu

Résultat : Les deux arêtes incidentes à b et à $\beta_1(b)$ sont fusionnées.

$t_1 \leftarrow \beta_1(b)$; $t_2 \leftarrow \beta_2(b)$;
$c \leftarrow \beta_2(t_1)$; $d \leftarrow \beta_1(t_1)$; $e \leftarrow \beta_1(t_2)$;
β_0-découdre(t_1) ; β_1-découdre(t_1) ; β_2-découdre(t_1);
β_0-découdre(t_2) ; β_1-découdre(t_2) ; β_2-découdre(t_2);
β_1-coudre(b, d) ; β_1-coudre(c, e) ; β_2-coudre(b, c);



a. Avant la fusion. b. Après la fusion.

FIG. 4.15 – Fusion d'arêtes en 2d.

à ces deux brins, nous supprimons seulement l'orbite du sommet à fusionner. Dans le cas où nous avons un plongement arête, nous devons distinguer deux cas différents. Si les arêtes sont alignées, il n'y a pas de modification à faire, car le plongement de l'arête est simplement le segment de droite reliant les deux plongements sommets extrémités de l'arête. Par contre, si les deux arêtes ne sont pas alignées, nous devons modifier le plongement arête afin de conserver toute la géométrie de la frontière. Dans ce cas, il suffit d'ajouter les coordonnées du sommet supprimé en fin du plongement arête du brin b. De plus, l'opération de couture par β_2 devra gérer le plongement arête, afin de fusionner éventuellement les deux plongements des deux brins cousus. Ces opérations de mise à jour du plongement se définissent simplement, et permettent ensuite d'oublier totalement le plongement qui sera mis à jour de manière transparente par les opérations de base sur la carte.

4.4.2 L'algorithme d'extraction naïf

Les définitions des différents niveaux de carte, présentées section 4.2, nous donnent immédiatement l'algorithme 3 d'extraction. Cet algorithme prend en entrée une image segmentée en région, et le niveau de simplification souhaité (un entier entre 1 et 3), et retourne la carte correspondante.

La première ligne de cet algorithme construit la carte de niveau 0. Pour obtenir la carte de niveau 1, nous devons, d'après la définition 16, fusionner chaque couple de faces adjacentes et appartenant à la même région. Nous utilisons pour cela l'opération de fusion présentée section 4.4.1. Cette fusion de faces s'effectue uniquement le long d'une arête. Nous devons éventuellement fusionner plusieurs fois le même couple de faces, mais le long d'arêtes différentes. De plus, nous devons également fusionner une face avec elle-même, afin de supprimer les éventuels isthmes. Mais cette opération s'effectue, malgré cela, en complexité linéaire en le nombre de brins de la carte. En effet, nous parcourons l'ensemble des arêtes de la carte, et pour chacune testons si

Algorithme 3 Extraction « naïve » de la carte de niveau n en 2d

Entrée : Une image I segmentée en régions de $n_1 \times n_2$ pixels
n le niveau de la carte que nous voulons extraire
Sortie : La carte de niveau n de l'image I.

1 Construire la carte de niveau 0 correspondant à I;
2 Fusionner tout couple de faces adjacentes appartenant à la même région;
si $n \geq 2$ **alors**
3 Fusionner tout couple d'arêtes alignées incidentes à un sommet de degré deux;
si $n = 3$ **alors**
4 Fusionner tout couple d'arêtes incidentes à un sommet de degré deux;
5 Calculer l'arbre d'inclusion des régions;

les deux faces qui lui sont incidentes sont de même région. Si c'est le cas nous effectuons leur fusion, et sinon nous savons que cette fusion de faces ne sera jamais réalisée. Cette technique nous permet de fusionner, lorsque cela est nécessaire, plusieurs fois les mêmes faces le long de différentes arêtes, mais également de fusionner une face avec elle-même. Avec cette technique, nous parcourons exactement une fois chaque arête de la carte, d'où la complexité linéaire en le nombre de brins (car le nombre d'arêtes dans une 2-carte fermée est égal à deux fois le nombre de brins).

Ensuite, si le niveau désiré est le niveau 2 ou 3, nous devons d'après la définition 17, fusionner chaque couple d'arêtes alignées incidentes à un sommet de degré deux pour obtenir la carte de niveau 2. Enfin, si la carte demandée est celle de niveau 3, nous fusionnons chaque couple d'arêtes incidentes à un sommet de degré deux d'après la définition 18. Ces deux types de fusions d'arêtes ne peuvent pas être effectués entre deux fois la même arête, contrairement à la fusion de faces. En effet, dans ce cas le sommet est de degré un, et nous ne devons alors pas effectuer de fusion. Afin de réaliser ces deux étapes, nous utilisons le même principe que pour la fusion de faces. Nous parcourons l'ensemble des sommets de la carte, testons pour chacun s'il est de degré deux, et effectuons la fusion si c'est le cas. La complexité est donc à nouveau linéaire en le nombre de brins de la carte.

Nous pouvons remarquer que pour l'extraction de la carte de niveau 3, l'étape 3 pourrait être supprimée, car les arêtes fusionnées lors de cette étape le seraient lors de l'étape 4. Mais conserver cette étape permet d'avoir un parallèle total entre l'algorithme et les divers niveaux de simplification. La carte de niveau 0 est construite au début de l'algorithme, puis chaque nouvelle étape construit le niveau suivant de simplification.

Cet algorithme ne présente pas les opérations concernant les modifications du plongement, car elles sont faites de manière transparente lors de l'opération de fusion, de la manière déjà présentée section précédente. Sa complexité globale est linéaire en le nombre de brins de la carte, car nous venons de voir que c'est le cas pour le calcul de chacun des niveaux, et nous verrons section 4.4.3 que c'est également le cas de l'algorithme de calcul de l'arbre d'inclusion.

4.4.3 Calcul de l'arbre d'inclusion

Le calcul de l'arbre d'inclusion s'effectue en dernière étape de notre algorithme d'extraction, après avoir extrait la carte. Cet arbre est nécessaire afin de conserver les liens entre les diverses composantes connexes de la carte, et pour les positionner les unes par rapport aux autres. Cet algorithme est générique et fonctionne sur les 3 niveaux de carte.

La notion d'inclusion utilisée ici est celle présentée définition 12 section 4.1.1. D'après cette définition, une région est incluse dans une ou plusieurs régions. Mais il n'est pas nécessaire de coder, pour une région, l'ensemble des régions dans laquelle elle est incluse car ces régions se retrouvent sans difficulté grâce à la propriété de transitivité de la relation d'inclusion.

Nous représentons donc uniquement la relation d'inclusion directe (définition 13) au moyen d'un *arbre d'inclusion*, un nœud associé à une région R_i est fils d'un nœud associé à une région R_j si et seulement si R_i est directement incluse dans R_j. Cet arbre aura toujours pour racine la région infinie. Si nous voulons retrouver, pour une région R_i, toutes les régions dans laquelle R_i est incluse, il suffit de parcourir l'arbre d'inclusion en partant du nœud associé à R_i et de remonter jusqu'à la racine. Chaque nœud rencontré lors de ce parcours est associé à une région dans laquelle R_i est incluse.

Pour construire cet arbre d'inclusion, nous parcourons l'ensemble des brins de la carte, composante connexe par composante connexe, ce qui permet de retrouver les diverses inclusions. Mais pour cela, nous avons besoin d'un certain nombre d'informations que nous utilisons lors de cette étape :

- chaque brin doit connaître sa région d'appartenance ;
- chaque région r connaît un des brins de la carte associée à cette région. Nous appelons un tel brin le *représentant* de sa région. Ce brin est soit le brin ayant comme plongement sommet le pointel le plus en haut et à gauche de r, soit si un tel brin n'existe pas, le brin ayant ce même pointel dans son plongement arête ;
- nous connaissons la liste de toutes les régions de l'image, sans la région infinie. Cette liste doit être ordonnée de sorte qu'une région R_i est inférieure à une autre région R_j si et seulement si R_i se rencontre avant R_j lorsque nous parcourons l'image de haut en bas et de gauche à droite.

Ces trois propriétés sont assurées par nos algorithmes d'extraction. La première lors de la construction de la carte de niveau 0, la deuxième également lors de cette construction, puis en nous assurant de la conserver lors de chaque fusion, et enfin la dernière par un parcours préalable de l'image, en classant les régions rencontrées selon cet ordre. Ce sont ces propriétés qui nous assurent que la région contenant la composante connexe en cours de traitement est bien la région *parent*.

L'algorithme 4 calcule l'arbre d'inclusion d'une carte représentant les frontières d'une image segmentée en régions. Il prend comme paramètres d'entrée une carte, qui peut être de n'importe quel niveau, et la liste des régions de l'image, triée suivant l'ordre présenté ci-dessus.

Cet algorithme se décompose en deux boucles. La première parcourt l'ensemble des régions de la liste L non marquées. En effet, quand une région est marquée, c'est qu'elle a déjà été traitée, ainsi que toute la composante connexe incidente à cette région. Pour une région non marquée, nous initialisons le brin de départ du parcours de la deuxième boucle comme étant le représentant de cette région. La région dans laquelle la composante connexe incidente à ce brin est incluse est forcément la région du brin cousu par β_2 au brin représentant de sa région. Cela est dû au fait

Algorithme 4 Calcul de l'arbre d'inclusion en 2d

Entrée : La carte C d'une image I segmentée en régions
La liste L des régions de l'image I
Sortie : L'arbre d'inclusion $\mathcal{A}$ correspondant.

Démarquer chaque région de L;
Ajouter un nœud associé à R_0 dans $\mathcal{A}$;
1 **pour chaque** *région R_i de L non marquée* **faire**
  $b_{init} \leftarrow représentant(R_i)$;
  parent $\leftarrow$ le nœud associé à $région(\beta_2(b_{init}))$;
2 **pour chaque** *brin b de la composante connexe incidente à b_{init}* **faire**
    si $région(b)$ *non marquée* **alors**
      Ajouter un nœud associé à $région(b)$ dans $\mathcal{A}$;
      Ajouter ce nœud comme fils de *parent* dans $\mathcal{A}$;
      Marquer $région(b)$;
retourner $\mathcal{A}$

que la liste des régions L est ordonnée de telle manière qu'une région R_i apparaît avant une autre région R_j si et seulement si R_i se rencontre avant R_j lorsque nous parcourons l'image de haut en bas et de gauche à droite, mais également à la contrainte fixée sur les brins représentants.

De par notre ordre de parcours particulier de la liste L, nous savons, lorsque nous atteignons une région non marquée R_i, qu'elle est forcément incluse dans une autre région déjà traitée, et qu'à sa gauche et en-dessus de cette région R_i se trouve cette région dans laquelle elle est incluse. Supposons que R_i ne soit pas incluse dans une région R_j déjà traitée. Cela signifierait que R_j ne possède aucun pixel à gauche et au-dessus de R_i, ce qui est en contradiction avec la définition de l'inclusion. Pour cette même raison, nous sommes certains qu'à gauche et au-dessus de R_i se trouvent uniquement des pixels de R_j. En effet, si nous trouvons à gauche ou en-dessus de R_i des pixels appartenant à une autre région que R_j, cela voudrait dire que cette région serait avant R_i dans la liste. Comme cette région appartiendrait à la même composante connexe que R_i, elle ne serait donc pas traitée, sinon nous aurions également traité R_i. Cela est en contradiction avec le fait que nous parcourons la liste L en respectant son ordre, et en traitant les premières régions rencontrées.

La boucle 2 consiste à parcourir chaque brin de la composante connexe (dans la carte C) incidente à un brin de départ particulier b_{init}, qui est le représentant de la région en cours de traitement. Avant cette boucle, nous avons donc initialisé ce brin de départ, ainsi que le nœud associé à la région dans laquelle la composante connexe incidente à ce brin est incluse : c'est *parent*. Au cours de ce parcours, chaque nouvelle région rencontrée est une région directement incluse dans la région associée à *parent*. Nous associons donc à cette région un nœud dans l'arbre d'inclusion qui est un fils du nœud *parent*. Comme nous parcourons tous les brins de la composante connexe incidente à b_{init}, nous traitons donc toutes les régions incidentes (mais pas incluses) à la région contenant b_{init}.

La complexité de cet algorithme est linéaire en le nombre de brins de la carte et en le nombre de régions de l'image (en pratique, le nombre de régions est très inférieur au nombre de brins). En effet, nous parcourons chaque brin exactement une fois, car nous parcourons les composantes

connexes une par une, deux composantes connexes différentes étant forcément disjointes. Nous parcourons également chaque région une seule fois car nous parcourons la liste L. Cet algorithme de calcul fonctionne pour n'importe quel niveau de carte. De plus, son exécution est plus rapide lorsque le niveau augmente, même si la complexité ne change pas, car le nombre de brins diminue de plus en plus.

L'algorithme naïf d'extraction d'une carte à partir d'une image est l'application directe de la définition des différents niveaux de carte. Il est simple à comprendre, et facilement implantable. Mais il a comme inconvénient principal de créer au départ beaucoup de brins, pour en détruire ensuite un grand nombre lors des différentes fusions. Cela pose deux problèmes majeurs. Tout d'abord le temps nécessaire à la création et à la destruction de nombreux brins est important et inutile. Ensuite, l'espace nécessaire à la construction du niveau 0 peut être trop important pour pouvoir construire cette carte, alors que la carte topologique demande un espace mémoire beaucoup moins important et devrait donc pouvoir être calculée aisément. De plus, après avoir créé la carte de niveau 0, il faut parcourir chaque brin n fois pour obtenir la carte de niveau n. Bien que cela ne modifie pas l'ordre de complexité, cela entraîne quand même une perte de temps d'exécution. Nous présentons donc une amélioration de cet algorithme naïf, utilisant la notion de précode déjà utilisée par [Cha95, Fio95, Fio96, BFP99, BDF00]. Nous allons montrer qu'il y a un parallèle entre le niveau de simplification que nous voulons extraire et le nombre de précodes à traiter. De plus, notre approche particulière, basée sur nos différents niveaux de simplification, nous permet de factoriser au maximum les précodes pouvant se traiter de manière identique.

4.5 Les précodes pour définir un algorithme optimal d'extraction

L'idée de Christophe Fiorio [Fio95, Fio96] était de ne pas construire, comme pour notre algorithme d'extraction naïf, toute la carte de niveau 0 pour la simplifier ensuite. Son algorithme d'extraction effectue un balayage de l'image de haut en bas et de gauche à droite, avec une fenêtre de 2×2 pixels, en exécutant un traitement dépendant de la configuration locale des pixels de cette fenêtre.

Nous avons étendu cette idée, pour l'adapter à nos divers niveaux de carte, et nous avons étudié précisément les configurations à traiter pour chacun de ces niveaux. L'idée générale de notre démarche est de calculer la carte de manière incrémentale et directe, au moyen d'un seul balayage de l'image. De plus, les traitements à effectuer vont dépendre uniquement de la configuration locale des pixels, et vont créer exactement le nombre de brins nécessaires. L'originalité de ce travail est principalement la factorisation des précodes pouvant se traiter de manière similaire. Cette factorisation est facilitée par l'introduction des niveaux de simplification. Nous obtenons alors des « classes » de précodes en fonction du niveau de simplification souhaité. Mais avant de définir ces classes et de donner notre algorithme d'extraction, nous devons dans un premier temps définir la notion de précode.

4.5.1 Les précodes

Nous balayons l'image de haut en bas et de gauche à droite. Au cours de ce balayage, nous appelons *pixel courant* le pixel en cours de traitement, donc le dernier rencontré au cours de notre balayage, et *précode* une configuration locale de pixels.

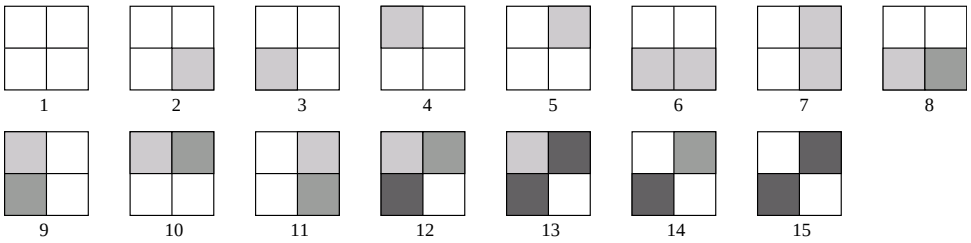


FIG. 4.16 – Les 15 précodes de la dimension 2.

Définition 19 (précode) *Un* précode *en dimension n est une partition de l'ensemble des 2^n n-cellules de l'hypercube de longueur* 2*.*

Un précode est donc une partition $\{p_1, \ldots, p_k\}$ d'un ensemble de cellules. Chaque p_i est l'ensemble des cellules appartenant à la même région. Lorsque $p_i \neq p_j$, alors les cellules de p_i appartiennent à une région différente des cellules de p_j.

Pour la dimension 2, un précode est une fenêtre de 2×2 pixels, le pixel courant étant celui en bas à droite de cette fenêtre. Nous pouvons voir figure 4.16 les précodes de dimension 2. Nous représentons un précode en affectant une couleur différente à chaque élément de la partition. Deux pixels de même couleur indiquent qu'ils appartiennent à la même région, et réciproquement.

Mais il faut effectuer le lien entre les précodes et les images segmentées en régions. Pour cela, étant donné un précode, nous associons une région différente à chaque p_i de la partition de ce précode. Nous obtenons alors une *configuration*, que nous dirons *couverte* par ce précode. Un précode couvre plusieurs configurations. En effet, pour un précode $\{p_1, \ldots, p_k\}$, si nous avons r régions distinctes dans l'image, nous pouvons affecter r régions à p_1, $r-1$ régions à p_2, ..., $r-k+1$ à p_k. Le nombre de configurations couvertes par un précode est donc $A_r^k = \frac{r!}{(r-k)!}$.

Un précode en dimension n étant une partition d'un ensemble de 2^n n-cellules, nous pouvons donc calculer le nombre de précodes distincts qui est le nombre de manières de partitionner les 2^n cellules. Soit N_n le nombre de précodes à la dimension n, nous pouvons calculer N_n grâce aux nombres S_l^k, appelés nombres de Stirling de première espèce. S_l^k donne le nombre de surjections d'un ensemble de l éléments dans un ensemble à k éléments. $\frac{S_l^k}{k!}$ donne alors le nombre de partitions en k sous-ensembles d'un ensemble à l éléments. Dans notre cas, $l = 2^n$. Il suffit alors d'additionner les partitions à un ensemble, deux ensembles, ..., à 2^n ensembles pour obtenir le nombre de précodes. Nous avons donc : $N_n = \sum_{k=1}^{2^n} \frac{S_{2^n}^k}{k!}$. Sachant que $B_l = \sum_{k=1}^{l} \frac{S_l^k}{k!}$ où B_l sont les nombres de Bell donnés par la formule de récurrence suivante : $B_l = \sum_{i=0}^{l-1} \frac{B_i . l!}{i!(l-i)!}$ avec $B_0 = B_1 = 1$, nous pouvons alors calculer le nombre de précodes N_n pour la dimension n qui est B_{2^n}. La formule précédente permet de calculer qu'il y a 15 précodes différents pour la dimension 2, déjà présentés figure 4.16.

Définition 20 (précode non variété) *On dit qu'un précode de dimension n est* non variété *si et seulement s'il existe dans ce précode une partition n'étant pas $2n$-connexe. Nous appelons précode* variété *un précode n'étant pas non variété.*

Pour la dimension 2, nous pouvons voir sur la figure 4.16 que les trois derniers précodes sont non variétés. La notion de précode est utile afin de définir les actions à effectuer lors de notre balayage de l'image. Mais nous avons besoin d'une autre notion permettant de regrouper plusieurs précodes : c'est la notion de *précode partiel*.

Définition 21 (précode partiel) *Un* précode partiel *en dimension n est une partition d'un sous-ensemble des cellules d'un précode, chaque ensemble de la partition devant être un ensemble $2n$-connexe.*

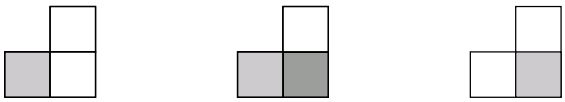


FIG. 4.17 – Deux exemples de précodes partiels (a et b) et un contre-exemple (c).

Nous pouvons voir figure 4.17 deux exemples de précodes partiels en dimension 2 (*a* et *b*) ainsi qu'un contre-exemple de précode partiel (*c*). Les pixels non représentés peuvent appartenir à n'importe quelle région. La partition associée au précode présenté figure 4.17.*c* n'est pas un précode partiel car un ensemble de cellules de la même région n'est pas 4-connexe.

L'interprétation d'un précode partiel n'est pas totalement identique à celle d'un précode. En effet, si nous appelons $\{p_1, \ldots, p_k\}$ la partition du précode partiel, chaque p_i contient les cellules étant de la même région. Mais contrairement aux précodes, deux p_i différents ne sont pas obligatoirement de deux régions différentes. En fait il y a deux cas distincts. Étant donné $p_i \neq p_j$:

- $p_i \cup p_j$ est $2n$-connexe. Alors la région associée à p_i est forcément différente de la région associée à p_j ;
- $p_i \cup p_j$ n'est pas $2n$-connexe. Alors la région associée à p_i peut être égale ou différente de la région associée à p_j.

De manière intuitive, cela veut dire que le précode partiel fixe l'appartenance ou non à la même région uniquement pour les régions voisines. Sur notre exemple, figure 4.17, le précode partiel *b* indique uniquement que le pixel inférieur droit n'est pas de la même région que son voisin de gauche, et pas de la même région que son voisin du haut. Mais il ne fixe aucune contrainte sur l'appartenance à la même région des pixels en bas à gauche et en haut à droite, car leur union n'est pas un ensemble 4-connexe. Par contre, les deux ensembles de pixels du précode partiel *a* ne peuvent pas être de la même région car leur union est un ensemble 4-connexe.

Les cellules n'étant dans aucun élément de la partition d'un précode partiel peuvent être de n'importe quelle région, sans aucune contrainte. Nous les appelons les *cellules libres*. Un précode partiel *couvre* plusieurs précodes : ce sont les précodes que l'on peut obtenir à partir du précode partiel en affectant les cellules libres dans des autres partitions ou dans des nouvelles partitions, et en faisant éventuellement des fusions respectant les contraintes déjà présentées.

Sur la figure 4.17, le précode partiel *a* couvre les précodes 3, 7 et 9 de la figure 4.16. Le précode partiel *b* couvre les précodes 2, 8, 11, 12, 13 et 14. Nous pouvons remarquer que ce précode couvre par exemple le précode 2 où les pixels en bas à gauche et en haut à droite sont de la même région, et couvre également le précode 12 où ces deux pixels sont dans des régions différentes.

4.5.2 Algorithme optimal et général d'extraction

L'algorithme général d'extraction d'une carte en une seule passe de l'image est simple puisque, comme nous pouvons voir sur l'algorithme 5, il se résume par un balayage de l'image en exécutant le code associé à chaque précode rencontré. Cet algorithme fonctionne pour tout type d'image : segmentées ou non, étiquetage fort ou faible[4]. En effet, la seule information utilisée ici est la notion de précode basée uniquement sur les configurations locales de pixels.

Algorithme 5 Extraction optimale de la carte de niveau n en 2d

Entrée : Une image I segmentée en régions de $n_1 \times n_2$ pixels
n le niveau de la carte que l'on veut extraire

Sortie : La carte de niveau n de l'image I.

```
1 last ← Construire le bord supérieur de la carte;
2 pour j = 1 à n2 + 1 faire
      pour i = 1 à n1 + 1 faire
          last ← Exécuter le code associé au précode (i, j) du niveau n;
3 Calculer l'arbre d'inclusion des régions;
```

Nous considérons ici que l'image est constituée des pixels dont les coordonnées vont de $1 \dots n_1$ en x, et de $1 \dots n_2$ en y, et qu'elle est entourée d'une région infinie. Le précode (i, j) est le précode ayant comme pixel courant (c'est le pixel en bas à droite de la fenêtre 2×2) le pixel (i, j).

La première étape de cet algorithme consiste à construire le bord supérieur de la carte. En effet au cours de cet algorithme, l'invariant est que la carte correspondant aux pixels plus anciens (pour notre ordre de parcours) a déjà été construite. Cet invariant garantit, à tout moment, qu'il existe un brin à gauche et au-dessus du pixel courant que l'on appelle respectivement *last* et *up* (cf. figure 4.18).

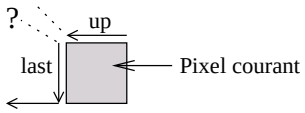


FIG. 4.18 – Les brins *last* et *up* par rapport au pixel courant.

Avant de commencer le parcours de l'image, le pixel courant est le pixel $(1, 1)$ et pour que cet invariant soit vérifié, nous devons avoir construit la carte pour les pixels de la région infinie à gauche ou au-dessus de ce pixel. C'est cette carte qui va être créée au cours de cette première étape détaillée plus précisément section 4.5.6.

Au cours de notre algorithme d'extraction, nous conservons le brin *last* qui permet de « raccrocher » le bout de la nouvelle carte à l'ancienne. Il nous suffit de conserver ce brin, car le brin *up* peut se calculer facilement à partir de ce brin, à l'aide de l'algorithme 6.

[4] On parle d'étiquetage fort si chaque région possède une étiquette unique, et d'étiquetage faible si deux régions non connexes peuvent avoir une même étiquette.

Algorithme 6 Calcul du brin *up* en 2d

Entrée : *last* le brin à gauche du pixel courant

Sortie : *up* le brin au-dessus du pixel courant.

$up \leftarrow \beta_0(last)$;
tant que up *est* β_2*-cousu* **faire**
  $up \leftarrow \beta_{20}(up)$)
retourner up;

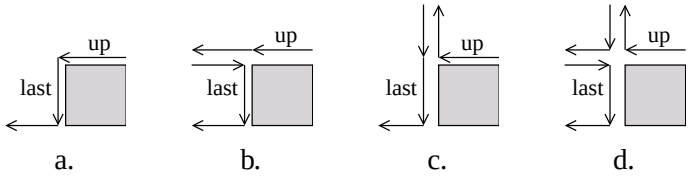


FIG. 4.19 – Les quatre configurations possible des brins *last* et *up*.

Pour cet algorithme, nous partons du brin *last* et nous « tournons » autour du sommet incident à ce brin jusqu'à trouver un brin qui n'est pas β_2-cousu. Ce brin est alors le brin *up*, car tous les autres brins incidents à ce sommet sont forcément β_2-cousus. Nous pouvons voir figure 4.19 les quatre configurations possibles des brins *last* et *up*. Sur cette figure, nous avons représenté le pixel courant, et les différentes possibilités des cartes déjà construites. Ce sont les seules car la carte que nous construisons est associée aux frontières interpixel, ce qui limite les configurations possibles. Nous voyons sur cette figure que l'algorithme de calcul du brin *up* à partir du brin *last* effectue, dans le cas *d* qui est le pire des cas, deux tours dans la boucle « tant que ». Nous notons, dans les prochaines figures, *l* le brin *last* et *u* le brin *up* afin de condenser la notation.

Après avoir créé le bord supérieur de l'image dans l'algorithme 5, le brin *last* est initialisé comme étant le brin vertical en haut à gauche de ce bord. L'algorithme d'extraction effectue ensuite un parcours de l'image de haut en bas et de gauche à droite, en exécutant chaque traitement associé au précode courant. Chacun de ces précodes modifie la carte localement, et retourne le brin vertical en bas à droite de ce précode, qui est alors affecté à *last* pour le prochain précode. Étant donné que l'image est plaquée sur un cylindre, ce traitement est valable pour tous les précodes, même ceux du bord. Nous consacrons la section 4.5.6 à la gestion du bord de l'image et à la manière dont il est utilisé lors de l'algorithme d'extraction.

Enfin, après avoir extrait la carte, il faut, comme pour l'algorithme naïf, construire l'arbre d'inclusion qui permet de faire le lien entre les éventuelles composantes connexes de la carte. Cet algorithme est l'algorithme 4 présenté section 4.4.3. Pour ce second algorithme d'extraction, le parcours préalable de l'image afin d'obtenir la liste des régions triées selon leur ordre d'apparitions n'est plus nécessaire. En effet, cette liste peut maintenant être créée durant le même parcours que celui construisant la carte.

Cet algorithme optimal d'extraction est générique. En effet, pour extraire un niveau particulier de carte, il faut définir quels précodes nous devons traiter, et comment nous allons les traiter. C'est

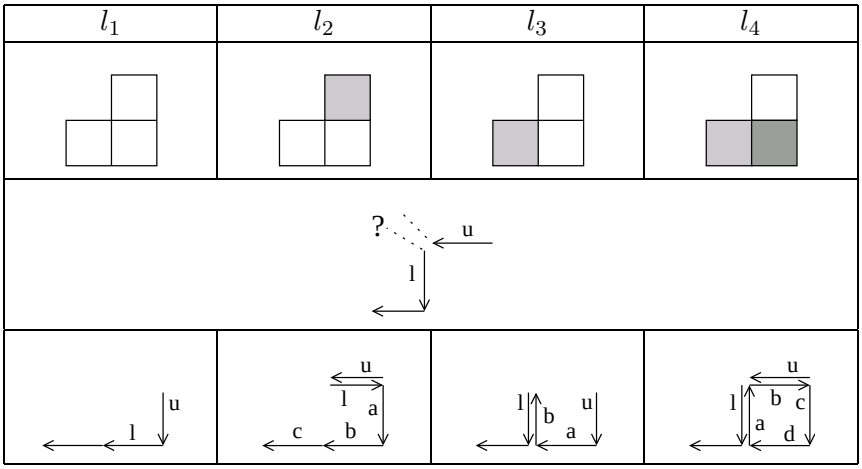


FIG. 4.20 – Les quatre précodes pour obtenir la carte lignel.

simplement pour ces définitions qu'il y aura une distinction entre les différents niveaux. Nous étudions, dans un premier temps, les précodes nécessaires à l'extraction de la carte lignel.

4.5.3 Les précodes pour la carte lignel

Pour trouver les précodes à traiter afin d'extraire une carte spécifique, nous devons regarder quelles sont les fusions à effectuer afin d'obtenir cette carte, et étudier l'implication de ces fusions sur le traitement du précode local. En effet, étant donné que nous avons déjà construit la carte des pixels plus anciens, nous devons uniquement étudier le traitement à réaliser pour le pixel courant.

Pour la carte lignel, la seule fusion à effectuer est celle de faces adjacentes appartenant à la même région (cf. définition 16). Nous considérons donc uniquement les deux voisins à gauche et au-dessus du pixel courant. En effet, ce sont les deux seules fusions de faces possibles avec le pixel courant. Il faut étudier toutes les combinaisons de fusion ou non du pixel courant avec ces deux voisins. Nous obtenons alors 4 possibilités qui sont les 4 précodes présentés figure 4.20.

Sur cette figure, nous avons représenté sur la première ligne les précodes (que nous appelons par la suite *précodes lignels*), sur la deuxième ligne l'état de la carte correspondante avant de traiter le pixel courant, et sur la dernière ligne la carte à obtenir après avoir traité ce précode. Nous ne connaissons pas l'ensemble des brins présents dans la carte étant donné que nous ne savons pas la région d'appartenance du pixel libre dans les précodes partiels. Mais il est suffisant de connaître les brins *last* et *up* car les modifications à effectuer porteront uniquement sur ces brins. De même, les cartes obtenues après traitement des précodes sont des représentations partielles où nous montrons uniquement les brins modifiés.

Pour trouver le code associé à chaque précode, il suffit de regarder la carte avant le traitement de ce précode, de la comparer à la carte que nous voulons obtenir, et d'écrire la suite d'opérations permettant de transformer la première carte en la seconde. Nous pouvons voir sur les algorithmes 7, 8, 9 et 10 que ce code est simple à écrire. De plus, il est facile de vérifier sur la figure 4.20 que ces

quatre algorithmes effectuent correctement les modifications des cartes de départ pour arriver aux cartes finales.

Algorithme 7 Code associé au précode l_1 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(up)$;
β_0-découdre$(last)$; β_1-découdre(up);
β_1-coudre$(up, last)$; β_1-coudre(a, b);
Le plongement du sommet incident à $last \leftarrow (x, y)$;
retourner up;

Algorithme 8 Code associé au précode l_2 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow$ un nouveau brin ; $b \leftarrow$ un nouveau brin;
$c \leftarrow \beta_1(last)$; β_1-découdre$(last)$;
β_1-coudre(a, b) ; β_1-coudre(b, c);
β_1-coudre$(last, a)$; β_2-coudre$(up, last)$;
Le plongement du sommet incident à $b \leftarrow (x, y)$;
retourner a;

Algorithme 9 Code associé au précode l_3 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow$ un nouveau brin ; $b \leftarrow$ un nouveau brin;
$c \leftarrow \beta_1(up)$; β_1-découdre(up);
β_1-coudre(a, b) ; β_1-coudre(b, c);
β_1-coudre(up, a) ; β_2-coudre$(last, b)$;
Le plongement du sommet incident à $a \leftarrow (x, y)$;
retourner up;

Algorithme 10 Code associé au précode l_4 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow$ un nouveau brin ; $b \leftarrow$ un nouveau brin;
$c \leftarrow$ un nouveau brin ; $d \leftarrow$ un nouveau brin;
β_1-coudre(a, b) ; β_1-coudre(b, c);
β_1-coudre(c, d) ; β_1-coudre(d, a);
β_2-coudre$(last, a)$; β_2-coudre(up, b);
Le plongement du sommet incident à $d \leftarrow (x, y)$;
retourner c;

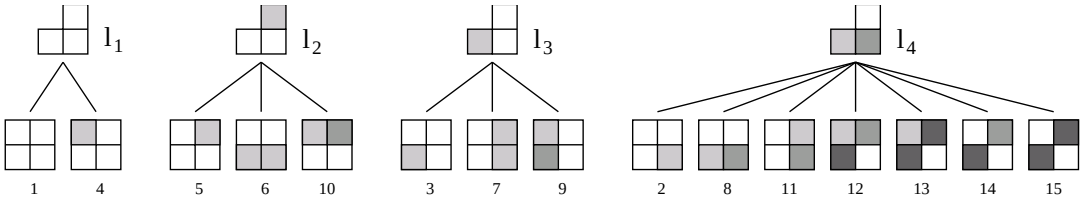


FIG. 4.21 – Les arbres de couverture des quatre précodes lignels.

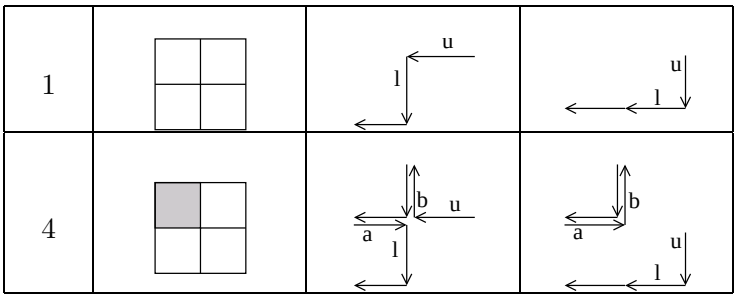


FIG. 4.22 – Les deux précodes couverts par le précode l_1 en 2d.

Pour ces algorithmes, ainsi que pour tous les suivants, nous considérons qu'à chaque sommet de la carte sont associées les coordonnées du pointel correspondant. Les fonctions de coutures et de décousures dupliquent ou fusionnent donc les plongements lorsque c'est nécessaire, comme nous l'avons présenté section 4.4. Le traitement d'un précode peut s'écrire de plusieurs manières. Nous avons essayé d'être minimal en nombre d'opérations afin d'être optimal en temps d'exécution. Nous avons également organisé les coutures et décousures de façon à minimiser les modifications de plongements.

Nous pouvons vérifier que les quatre précodes lignels couvrent bien toutes les configurations que nous pouvons rencontrer dans l'image. En effet, les 15 précodes les couvrent bien toutes par définition, et les quatre précodes lignels couvrent bien ces 15 précodes comme nous pouvons le voir figure 4.21. Sur cette figure, un précode p_1 est fils d'un précode partiel p_2 lorsque p_1 est couvert par p_2. À l'aide de ces arbres, nous pouvons également vérifier que les traitements à effectuer pour deux précodes couverts par le même précode partiel sont identiques. C'est pour cette raison que nous n'avons pas besoin de définir 15 traitements différents mais simplement 4.

Il est intéressant d'étudier plus précisément le précode l_1. En effet, il couvre les précodes 1 et 4, qui semblent assez différents, mais que nous traitons pourtant avec le même algorithme. Nous pouvons voir ces deux précodes figure 4.22, ainsi que la carte avant et après leurs traitements. La seule différence entre ces deux précodes, porte sur les brins appelés a et b pour le précode 4. En effet, dans ce cas, il faut β_1-coudre a à b, alors que cette opération n'est pas utile pour le précode 1. Mais cette opération, bien qu'inutile, n'est pas source d'erreur. En effet, pour ce précode 1, nous avons alors $a = u$ et $b = l$, et β_1-coudre a à b revient à « recoudre » u à l, qui étaient déjà β_1-cousus. Nous pouvons donc effectuer cette opération de couture tant pour le précode 1 que pour le

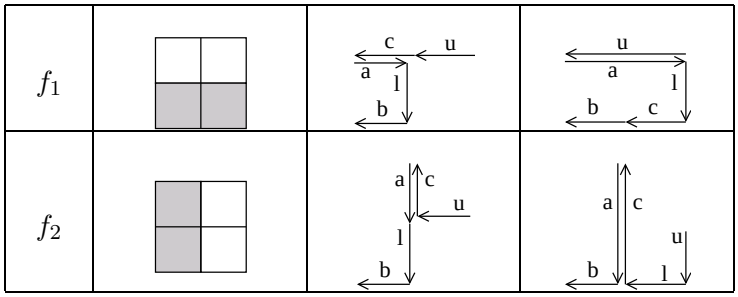


FIG. 4.23 – Les deux précodes pour obtenir la carte des frontières en 2d.

précode 4, ce qui permet d'obtenir un seul traitement et évite d'avoir à tester précisément quel est le précode courant.

La définition des quatre précodes lignels et de leur traitement définit complètement l'algorithme optimal d'extraction de la carte lignel.

4.5.4 Les précodes pour la carte des frontières

L'extraction de la carte des frontières nécessite de fusionner chaque couple d'arêtes alignées incidentes à un sommet de degré deux (cf. définition 17). Étant donné que nous effectuons la construction de cette carte en une seule passe, nous devons fusionner également les faces adjacentes de même région (fusion nécessaire pour obtenir la carte de niveau 1). Il est facile de voir qu'il existe seulement deux cas pour lesquels deux arêtes alignées sont incidentes à un sommet de degré deux : ce sont les précodes présentés figure 4.23. C'est donc seulement ces deux précodes qu'il va falloir traiter (en plus des quatre de la carte lignel), et c'est uniquement dans ces deux cas que nous allons fusionner deux arêtes. Nous devons conserver les quatre précodes lignel, car nous devons, lorsque nous ne sommes pas dans un de ces deux précodes frontières, traiter la configuration courante par l'un des quatre précodes lignels afin de réaliser les éventuelles fusions de faces nécessaires.

Sur chaque ligne de la figure 4.23, nous avons représenté le précode, puis la carte avant son traitement et enfin celle obtenue après traitement. Contrairement aux quatre précodes lignels, nous connaissons ici tous les brins présents dans la carte avant le traitement du précode. En effet, ces deux précodes ne sont pas partiels, et nous connaissons donc les régions des quatre pixels. Comme pour la carte lignel, l'écriture des algorithmes de traitement de ces deux précodes se fait de manière simple comme le montrent les algorithmes 11 et 12.

Pour ces deux précodes, nous ne faisons pas de modification de plongement, excepté pour le « nouveau sommet ». En effet, par exemple pour le précode f_1, nous conservons les brins a et u car ce sont ceux qui ont les plongements sommets que nous voulons obtenir dans la carte résultat. Pour le précode f_2, par exemple lors de la β_0-couture de b avec a, il y avait avant deux sommets topologiques distincts (incidents aux brins b et c) et après la couture il n'y en a plus qu'un. Dans ce cas, l'ordre de la couture importe : c'est toujours le sommet ayant un rapport avec le premier brin passé en paramètre qui est conservé. Lors du traitement du précode f_2, nous conservons donc bien le plongement du sommet incident à b.

Algorithme 11 Code associé au précode f_1 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(last)$;
$c \leftarrow \beta_1(up)$; $d \leftarrow \beta_1(c)$;
β_1-découdre$(last)$; β_0-découdre(c);
β_1-découdre(c) ; β_2-découdre(c);
β_1-coudre$(last, c)$; β_1-coudre(c, b);
β_1-coudre(up, d) ; β_2-coudre(up, a);
Le plongement du sommet incident à $c \leftarrow (x, y)$;
retourner last;

Algorithme 12 Code associé au précode f_2 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(last)$; $c \leftarrow \beta_1(up)$;
β_0-découdre$(last)$; β_1-découdre$(last)$; β_1-découdre(up);
β_1-coudre$(up, last)$; β_1-coudre$(last, c)$; β_0-coudre(b, a);
Le plongement du sommet incident à $last \leftarrow (x, y)$;
retourner up;

Remarquons également que pour ces deux précodes, nous conservons un brin particulier, c pour le précode f_1 et l pour f_2, qui est d'abord totalement décousu avant d'être ensuite utilisé à un autre endroit de la carte. Nous aurions pu choisir de détruire ce brin, et ensuite de créer un nouveau brin. Mais notre solution évite une destruction suivie d'une création, et permet donc un gain de temps d'exécution.

Pour extraire directement la carte des frontières à partir d'une image, nous appliquons l'algorithme 5 en cherchant à chaque étape de la boucle quel est le précode courant, parmi les 6 précodes possibles (les 2 précodes f_1 et f_2 plus les 4 précodes lignels) et en exécutant le code correspondant. Nous pouvons vérifier que ces six précodes couvrent bien toutes les configurations que nous pouvons rencontrer dans l'image sur les arbres de couverture présentés figure 4.24. Nous voyons sur cette figure que les deux précodes f_1 et f_2 couvrent uniquement un seul précode étant donné que ce ne sont pas des précodes partiels.

4.5.5 Les précodes pour la carte topologique

Pour ce niveau, nous devons fusionner chaque couple d'arêtes incidentes à des sommets de degré deux (cf. définition 18). Il existe seulement quatre cas différents pour lesquels cette fusion doit être réalisée. Ce sont les quatre précodes présentés figure 4.25. Cette figure montre sur la première ligne ces précodes, sur la deuxième la configuration de la carte avant de traiter le précode correspondant, et sur la dernière ligne la carte à obtenir après ce traitement. Nous n'avons pas représenté le plongement des arêtes sur les cartes de cette figure, mais il faut bien entendu le modifier. Nous supposons que le plongement de la carte est celui décrit section 4.3 au moyen d'un

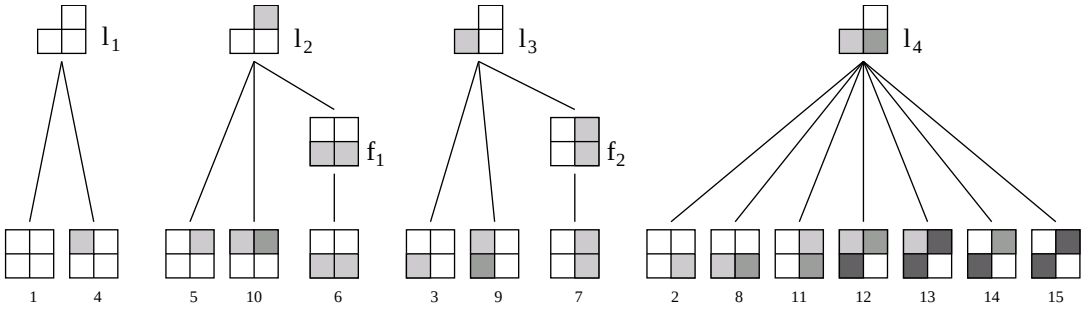


FIG. 4.24 – Les arbres de couverture pour la carte des frontières en 2d.

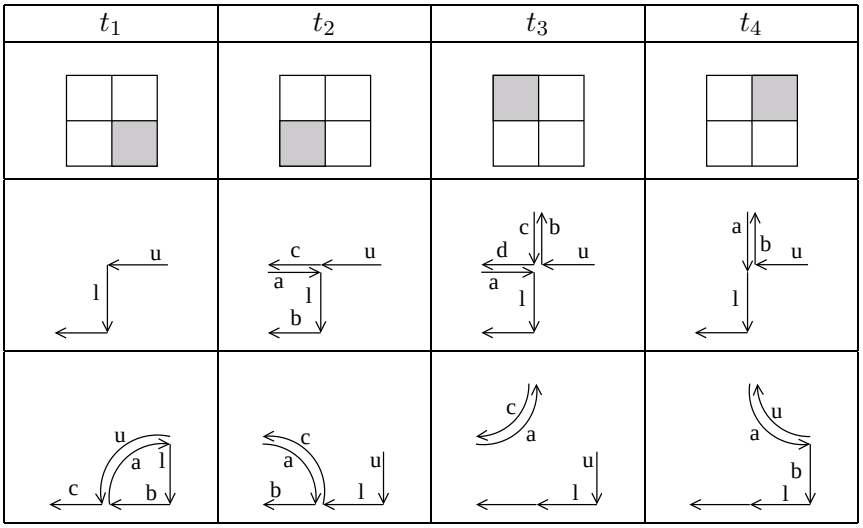


FIG. 4.25 – Les quatre précodes pour obtenir la carte topologique en 2d.

plongement sommet et d'un plongement arêtes ouvertes. Les algorithmes 13, 14, 15 et 16 donnent le code associé à ces quatre précodes.

Algorithme 13 Code associé au précode t_1 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow$ un nouveau brin ; $b \leftarrow$ un nouveau brin;
$c \leftarrow \beta_1(last)$; β_2-coudre(up, a);
Ajouter le plongement sommet de *last* en fin du plongement arête de a;
β_0-découdre$(last)$; β_1-découdre$(last)$;
β_1-coudre$(a, last)$; β_1-coudre$(last, b)$;
β_1-coudre(b, a) ; β_0-coudre(c, up);
Le plongement du sommet incident à $b \leftarrow (x, y)$;
retourner *last*;

Pour l'algorithme 13, nous savons que le brin *up* ne possède pas de plongement arête car il n'est pas β_2-cousu. Au cours du traitement de ce précode, il est β_2-cousu au brin a et forme donc désormais une arête. Afin de conserver le plongement, nous devons initialiser le plongement de cette arête comme étant une courbe réduite à un seul sommet : l'ancien sommet incident au brin *last*. Cette initialisation est réalisée au moyen d'une méthode générique qui ajoute les coordonnées d'un point en fin d'un plongement arête. Dans ce cas, étant donné qu'il n'y a pas de plongement arête, cela revient à initialiser ce plongement. Les plongements sommets n'ont pas besoin d'être mis à jour, excepté le nouveau sommet, car les méthodes de coutures se chargent de modifier correctement les plongements suivant les évolutions des orbites sommets. Par exemple pour la dernière couture de l'algorithme 13, il y a deux sommets topologiques incidents à c et à a. Le plongement sommet conservé est celui du sommet ayant un lien avec le premier paramètre, c'est-à-dire le sommet incident à c.

Algorithme 14 Code associé au précode t_2 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(last)$; $c \leftarrow \beta_1(up)$;
Ajouter le plongement sommet de *last* en fin du plongement arête de a;
β_0-découdre$(last)$; β_1-découdre$(last)$; β_1-découdre(up);
β_0-coudre(b, a) ; β_1-coudre$(up, last)$; β_1-coudre$(last, c)$;
Le plongement du sommet incident à $last \leftarrow (x, y)$;
retourner up;

Pour l'algorithme 14 traitant le précode t_2, l'arête incidente au brin a possède peut-être un plongement. Mais l'ajout en fin des coordonnées d'un sommet fonctionne correctement, que l'arête ait un plongement ou non. Pour les plongements sommets, il faut initialiser correctement celui du sommet incident au brin *last*, mais les autres n'ont pas besoin de mise à jour, toujours grâce aux coutures qui vont éventuellement les modifier.

Algorithme 15 Code associé au précode t_3 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(up)$; $c \leftarrow \beta_2(b)$;
$d \leftarrow \beta_1(c)$; $t_1 \leftarrow \beta_1(b)$; $t_2 \leftarrow \beta_1(d)$;
Ajouter le plongement sommet de *last* en fin du plongement arête de a;
Fusionner le plongement arête de b en fin du plongement arête de a;
β_0-découdre(b) ; β_1-découdre(b) ; β_2-découdre(b);
β_0-découdre(d) ; β_1-découdre(d) ; β_2-découdre(d);
β_0-découdre$(last)$; β_1-coudre(a, t_1) ; β_1-coudre(c, t_2);
β_1-coudre$(up, last)$;
β_2-coudre(a, c);
détruire les brins b et d;
Le plongement du sommet incident à $last \leftarrow (x, y)$;
retourner up;

Pour le traitement du précode t_3, nous pouvons remarquer sur l'algorithme 15 qu'il est un peu plus complexe que les précédents. En effet, il est possible que les arêtes incidentes au brin a et au brin b possèdent toutes deux un plongement arête. C'est pour cette raison que nous devons « fusionner » ces deux plongements, en insérant au milieu les coordonnées du sommet incident au brin *last*. La méthode se chargeant de cette fusion doit fonctionner sur des plongements vides, pour traiter les cas où une ou les deux arêtes n'ont pas de plongement arête. De plus, il faudra faire attention à l'ordre des plongements lors de cette fusion. Suivant la manière dont les courbes de dimension 1 sont implantées, nous pourrons avoir besoin de retourner une des deux courbes afin d'effectuer correctement la fusion. Lors de la β_2-couture de a avec c, si les deux brins ont tous deux un plongement arête, seul celui de a sera conservé, l'autre sera alors détruit.

De plus, cet algorithme nécessite un pré-traitement afin de gérer correctement le cas particulier où une région est incluse dans une autre sans avoir d'autre région adjacente (comme pour la région R_4 de notre image d'exemple). Dans ce cas, la frontière entre les deux régions est composée d'une boucle. Avant d'exécuter le traitement du précode t_3, il est donc nécessaire de tester si l'arête incidente au brin $\beta_0(last)$ est la même que celle incidente au brin $\beta_1(up)$. Lorsque c'est le cas, il n'y a pas de fusion d'arêtes à exécuter étant donné que le sommet est de degré un. Il suffit alors d'exécuter le traitement du précode l_1 à la place de celui de t_3 (l_1 est le précode père de t_3 dans la hiérarchie présentée figure 4.26). Le traitement de l_1 va effectuer les mêmes fusions de faces que celui de t_3, mais ne va pas fusionner les arêtes. Après ce traitement, nous obtenons bien la boucle représentant la frontière entre la région incluse et la région qui l'entoure.

Algorithme 16 Code associé au précode t_4 en 2d

Entrée : *last* et *up*
(x, y) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(up)$; $c \leftarrow \beta_1(b)$;
Ajouter le plongement sommet de *last* en fin du plongement arête de a;
β_0-découdre$(last)$; β_0-découdre(b);
β_1-découdre(b) ; β_2-découdre(b);
β_1-coudre(up, c) ; β_2-coudre(up, a);
β_1-coudre(a, b) ; β_1-coudre$(b, last)$;
Le plongement du sommet incident à $last \leftarrow (x, y)$;
retourner b;

L'algorithme 16 est plus simple car nous sommes sûrs que le brin *up* n'a pas de plongement arête, étant donné qu'il n'est pas β_2-cousu. Son traitement est similaire à ceux déjà présentés des précodes t_1 et t_2.

Maintenant que nous disposons des quatre algorithmes associés aux quatre précodes topologiques, nous pouvons extraire la carte topologique en une seule passe de l'image au moyen de l'algorithme optimal (algorithme 5) qui teste quel est le précode courant, parmi les 10 possibles (les 4 lignels plus les 2 frontières plus les 4 topologiques). Nous pouvons voir que cet algorithme, bien qu'un peu long à décrire en détail, est simple, et facile à implanter. De plus, nous avons un algorithme optimal, qui ne fait aucune opération inutile, qui crée directement le nombre de brins nécessaires tout en conservant le plongement sous forme de courbes de dimension 1 combinées avec le plongement des sommets.

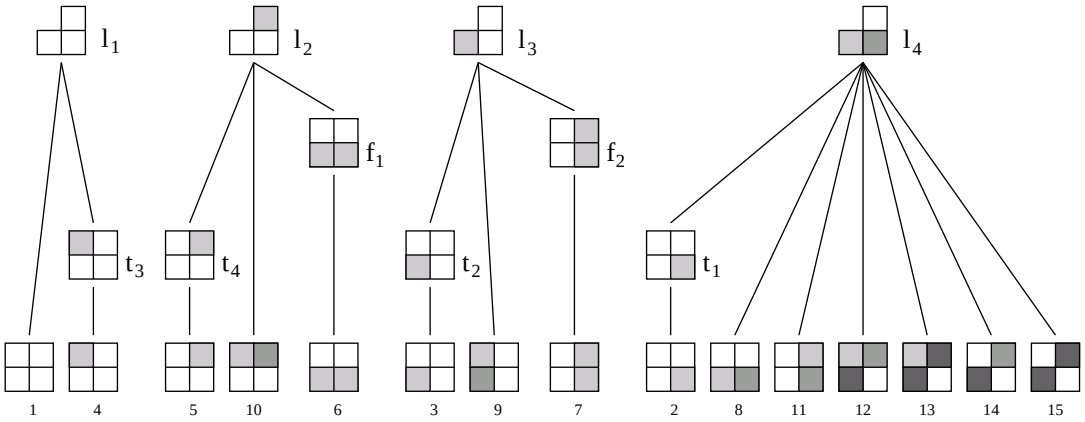


FIG. 4.26 – Les arbres de couverture pour la carte topologique en 2d.

Nous vérifions sur la figure 4.26 que les dix précodes couvrent bien toutes les configurations que nous pouvons rencontrer dans l'image. Nous voyons que le seul précode couvrant encore plusieurs cas est le précode l_4. Ce cas regroupe tous les précodes pour lesquels il n'y a aucune fusion à faire. Ces cas sont seulement au nombre de 6, mais nous verrons qu'ils sont beaucoup plus nombreux en dimension 3. Nous voyons que les précodes non variétés sont tous couverts par le précode l_4. En effet, nous représentons uniquement la 4-connexité entre les régions, et de ce point de vue les précodes non variétés 13, 14 et 15 sont équivalents au précode 12 pour lequel aucune fusion n'est nécessaire. Il est possible d'envisager une extension de nos travaux afin de représenter la 8-connexité, et dans ce cas ces précodes devraient alors être traités de manière différente.

Maintenant que nous avons défini totalement les algorithmes d'extraction des trois niveaux de carte possibles, nous allons revenir sur la gestion du bord de l'image. En effet, c'est grâce à sa définition adéquate que l'algorithme d'extraction reste simple, sans traiter un grand nombre de cas particuliers.

4.5.6 Le bord de l'image

Pour traiter les bords de l'image, une première solution consiste à traiter différemment les premières et dernières lignes et colonnes. Mais cela oblige à définir le traitement de chaque précode de plusieurs manières différentes, avec de légères variations à chaque fois. Cette multitude de méthodes risque d'être une source d'erreurs, et rend beaucoup plus difficile les évolutions ou modifications des précodes. De plus, l'algorithme d'extraction perd en lisibilité, car il faut alors effectuer le parcours de la première et de la dernière ligne dans une boucle spécifique, et pour chaque ligne traiter différemment la première et dernière colonne.

Afin de compter le nombre de méthodes supplémentaires que cette solution oblige à traiter, nous schématisons le bord d'une image au moyen des précodes figure 4.27. Cette figure montre quels précodes peuvent être rencontrés suivant la position du pixel courant dans l'image. De manière évidente, les quatre angles de l'image sont toujours fixes car un seul pixel appartient à l'image, les trois autres sont des pixels appartenant à la région infinie donc tous de la même ré-

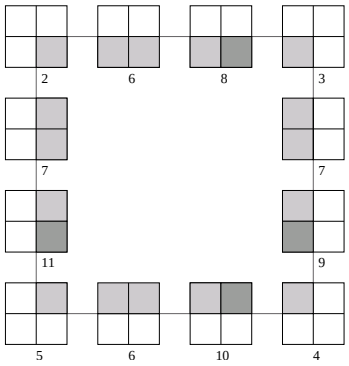
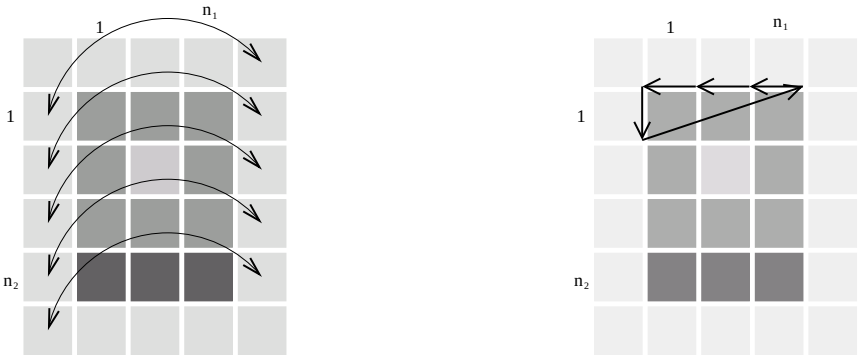


FIG. 4.27 – Les précodes possibles en fonction du bord de l'image.



a. L'image sur un cylindre : les pixels reliés par une flèche sont identifiés.

b. La carte pour le bord supérieur de l'image.

FIG. 4.28 – Gestion du bord en 2d.

gion. Pour les quatre bords (privés des angles), nous pouvons avoir à chaque fois deux précodes différents qui sont donnés sur la figure. Par exemple, sur le bord gauche de l'image, nous pouvons avoir uniquement le précode 7 ou le précode 11. Chacun de ces précodes doit avoir un traitement particulier suivant sa position dans l'image. Par exemple le précode 7 doit avoir un traitement différent s'il se trouve sur le bord gauche, le bord droit ou au milieu de l'image. Nous devons donc définir 12 traitements particuliers afin de gérer correctement le bord de l'image.

Cette solution étant trop lourde à mettre en œuvre, nous avons décidé de gérer le bord en considérant que l'image est plaquée sur un cylindre (cf. figure 4.28.*a*). Cette solution permet de ne pas se préoccuper de la position du pixel courant dans l'image : le traitement du précode courant est identique dans tous les cas. Pour mettre en œuvre ce traitement, il suffit de créer le bord supérieur de l'image de la manière présentée sur l'exemple figure 4.28.*b*. Ce bord étant fermé pour β_1, il simule le fait que l'image est sur un cylindre.

La figure 4.28.*a* présente une image de $n_1 \times n_2$ pixels. Le fait de considérer qu'elle est entourée par la région infinie permet d'avoir des pixels tout autour de l'image (aux coordonnées $(0, y)$,

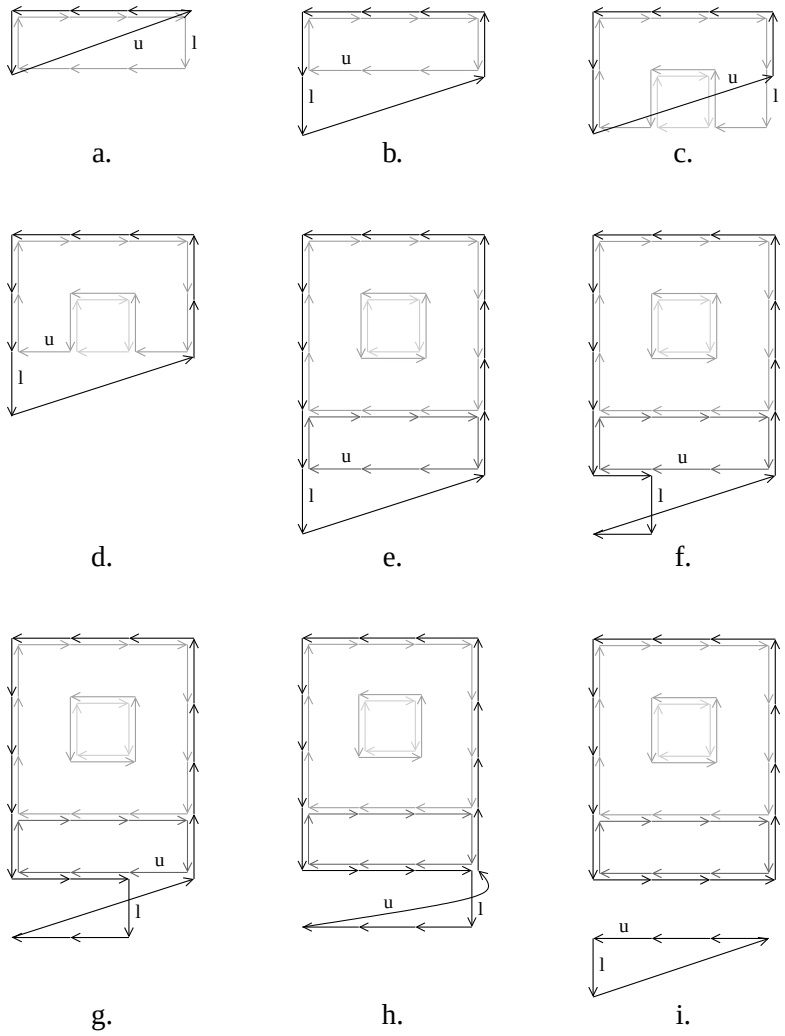


FIG. 4.29 – Déroulement de l'algorithme d'extraction de la carte lignel en 2d.

(n_1+1, y), $(x, 0)$ et (x, n_2+1)) appartenant tous à la région infinie. Ensuite, plaquer cette image sur un cylindre revient à identifier chaque pixel (n_1+1, j) au pixel $(0, j+1)$ (sur la figure 4.28.*a* nous identifions les pixels reliés par une flèche). Au cours du balayage de l'image, nous ne nous préoccupons plus de savoir si nous sommes en train de traiter un précode de bord ou non.

De par la conception initiale du bord, lorsque nous sommes sur le bord droit, nous créons le bord gauche de la ligne suivante. Pour cette raison, les coordonnées du prochain pointel à créer sont calculées modulo la largeur de l'image plus un. Si les coordonnées du pixel courant sont (i, j), les coordonnées du nouveau pointel sont $(i \text{ modulo } (n_1+1), j + (i \text{ div } (n_1+1))$.

Afin de comprendre le fonctionnement de notre algorithme optimal d'extraction (algorithme 5 page 51) lorsque nous traitons des précodes de bord, nous déroulons cet algorithme sur l'image présentée figure 4.28.*a* pour extraire la carte lignel. La carte initiale est donc le bord supérieur

de l'image (figure 4.28.*b*). La figure 4.29 montre quelques étapes importantes de l'algorithme. Les coordonnées des pixels sont données figure 4.28, et les coordonnées d'un pointel en haut et à gauche d'un pixel (x, y) sont $(x - 1, y - 1)$. De ce fait, le premier pointel de l'image a pour coordonnées $(0, 0)$.

Après avoir traité la première ligne, mais avant d'avoir traité le dernier pixel, nous avons obtenu la carte *a*. Nous traitons ensuite le dernier précode de cette ligne (qui est forcément le précode 3 quelle que soit l'image). Après ce traitement, nous obtenons la carte *b*. En effet, les coordonnées du pixel courant sont $(4, 1)$. Le nouveau sommet aurait dû être plongé aux coordonnées $(4, 1)$, mais grâce au cylindre, il va être plongé aux coordonnées $(0, 2)$. Nous voyons sur la carte *b* que *last* désigne bien le brin qui est à gauche du prochain pixel courant. Le brin *up* sera donc également correctement calculé. La carte *c* montre le cas similaire pour la ligne suivante. Cette carte montre la situation avant de traiter le dernier précode de cette ligne, et la carte *d* après l'avoir traité.

Pour le traitement de la dernière ligne, la carte *e* montre la carte avant le traitement du premier précode de cette ligne. Nous pouvons vérifier à l'aide des cartes *f*, *g* et *h* l'évolution de la carte lors du traitement des précodes de la dernière ligne. Enfin, la carte *i* montre le traitement du dernier précode de l'image. Après ce précode, la carte lignel de l'image est fermée, et le bord initial se retrouve maintenant en bas, et déconnecté. Étant devenu inutile, il est détruit.

Nous avons montré ici comment le bord intervenait lors de l'extraction d'une carte lignel. Cela fonctionnera exactement de la même manière pour l'extraction des autres niveaux de carte. En effet, quel que soit ce niveau, le bord de l'image est toujours le même et représente les lignels de celui-ci, les fusions d'arêtes étant exclusivement réalisées pour les brins au « milieu » de la carte en cours de construction.

4.6 Expérimentations et analyse

Nous avons implanté entièrement l'algorithme optimal d'extraction à base de précodes pour les trois niveaux de carte possible. Pour les niveaux 1 et 2 nous avons utilisé uniquement un plongement sommet où nous associons les coordonnées du pointel correspondant à chaque sommet topologique de la carte, et pour le niveau 3 le plongement sommet plus arête ouverte présenté section 4.3. Nous avons codé les courbes de dimension 1 avec des 1-cartes, ce qui nous permet d'avoir une structure hiérarchique. De plus, cette méthode s'étend sans problèmes en dimension supérieure, où nous pouvons plonger chaque cellule topologique de dimension i par une i-carte. Pour comparer l'espace mémoire nécessaire aux différents niveaux de carte de manière « équitable », nous avons implanté le plongement de manière optimisée pour chacun de ces niveaux.

Nous allons étudier et comparer les résultats d'extraction des trois niveaux de carte pour deux types d'images différentes. Dans un premier temps, section 4.6.1, nous avons étudié les images classiquement utilisées dans la communauté de la segmentation d'images, avec entre autres l'incontournable Lena. Section 4.6.2, nous présentons nos résultats sur quelques images médicales, qui sont des IRM de cerveau dans le cadre d'examens pour détecter des tumeurs cérébrales.

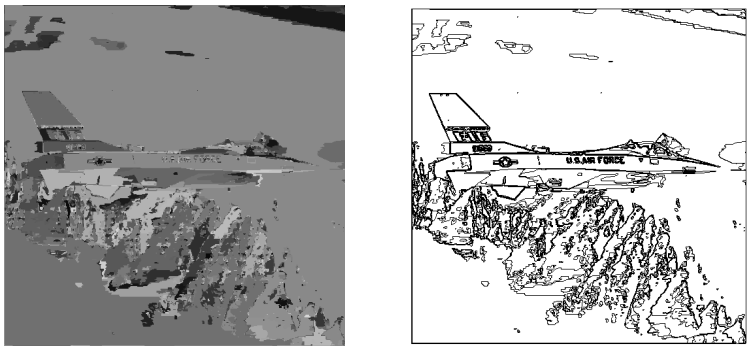


FIG. 4.30 – Airplane, 512×512, 6361 régions.

TAB. 4.1 – Airplane, 512×512, 6361 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	95 754	56 280	30 490
Nb de sommets topologiques	41 803	22 066	9 171
Espace mémoire en kilo-octets	2 744	1 665	1 230
Temps de calcul en secondes	3,45	3,26	3,12

4.6.1 Images standard

Nous présentons ici les différents résultats de l'extraction des trois niveaux de carte pour les six images segmentées présentées figures 4.30 à 4.35. Ces figures montrent dans un premier temps l'image segmentée en régions (une couleur par région) puis la carte obtenue qui représente les frontières interpixel. Les images ont fait l'objet d'une segmentation simple, sans pré ni post-traitement. Nous les utilisons ici uniquement afin de comparer les différents niveaux de carte.

Nous étudions, dans un premier temps, l'évolution du nombre de brins codant la topologie des objets, en fonction du niveau de carte. Nous regardons également le nombre de sommets topologiques, ce qui donne une idée du nombre de fusions d'arêtes effectuées. Puis nous comparons l'évolution de l'espace mémoire de la structure globale, comprenant le modèle topologique, le modèle géométrique et l'arbre d'inclusion. Nous présentons ces résultats tout d'abord de manière précise dans les tableaux 4.1 à 4.6. Dans ces tableaux, nous avons donné l'espace mémoire nécessaire en kilo-octets, et le temps d'extraction en secondes. Ce temps de calcul a été mesuré sur un Pentium III 600 Mhz, et inclus le temps nécessaire au calcul de l'arbre d'inclusion ainsi que le temps consacré aux entrées-sorties.

Afin de mieux visualiser l'évolution générale de ces caractéristiques, nous avons résumé ces résultats figure 4.36. Ces courbes montrent l'évolution de l'espace mémoire, du nombre de brins et du nombre de sommets topologiques sur nos trois niveaux de carte, en pourcentage du niveau 1. Elles permettent de vérifier que le gain entre les différents niveaux est important. Nous récapitulons le gain en espace mémoire, entre les divers niveaux dans le tableau 4.7. La première colonne de ce tableau donne le gain lorsque nous passons du niveau 1 au niveau 2. Nous pouvons voir qu'il

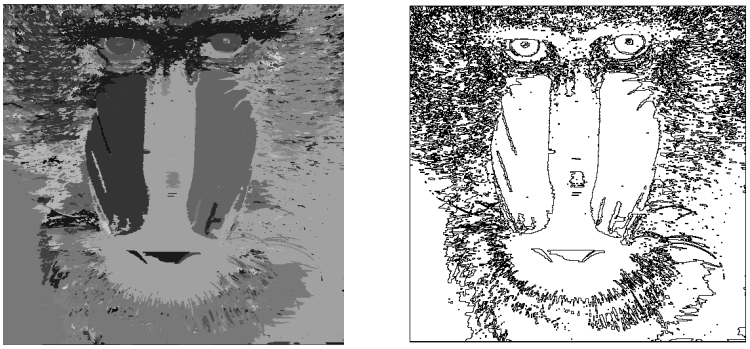


FIG. 4.31 – Baboon, 512×512, 6705 régions.

TAB. 4.2 – Baboon, 512×512, 6705 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	154 450	96 444	28 348
Nb de sommets topologiques	72 396	43 393	9 345
Espace mémoire en kilo-octets	4 368	2 782	1 560
Temps de calcul en secondes	4,95	4,82	4,75

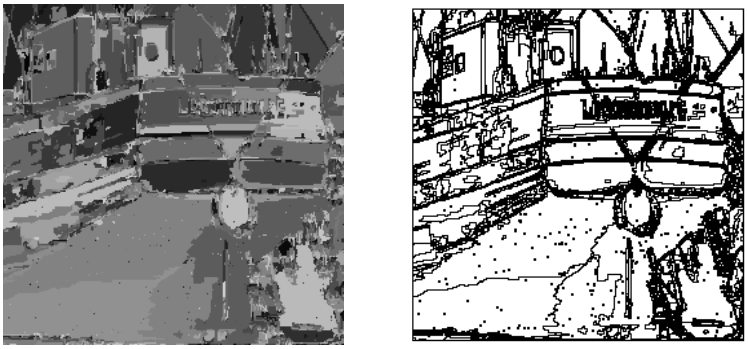


FIG. 4.32 – Cornouaille, 256×256, 5410 régions.

TAB. 4.3 – Cornouaille, 256×256, 5410 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	60 170	39 806	26 254
Nb de sommets topologiques	24 908	14 726	7 950
Espace mémoire en kilo-octets	1 752	1 195	970
Temps de calcul en secondes	1,85	1,76	1,69

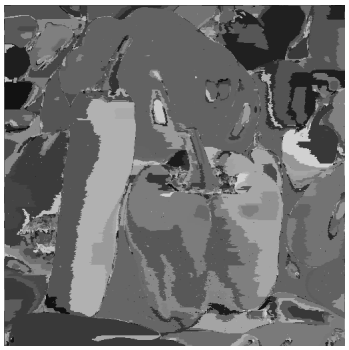



FIG. 4.33 – Peppers, 512×512, 5765 régions.

TAB. 4.4 – Peppers, 512×512, 5765 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	109 602	72 310	27 518
Nb de sommets topologiques	49 565	30 919	8 523
Espace mémoire en kilo-octets	3 113	2 093	1 285
Temps de calcul en secondes	3,16	3,01	2,92

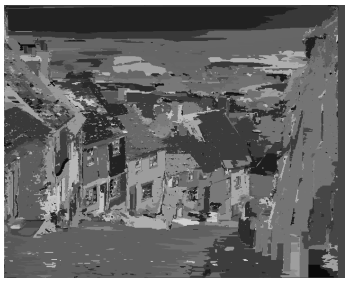



FIG. 4.34 – Goldhill, 720×576, 8006 régions.

TAB. 4.5 – Goldhill, 720×576, 8006 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	202 174	104 872	36 450
Nb de sommets topologiques	94 704	46 053	11 842
Espace mémoire en kilo-octets	5 697	3 036	1 815
Temps de calcul en secondes	9,65	9,35	9,24

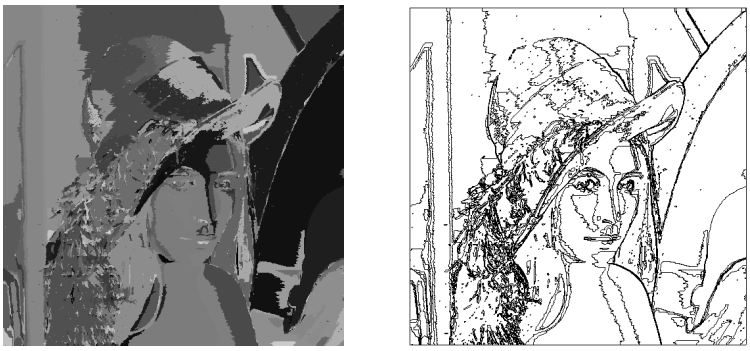


FIG. 4.35 – Lena, 512×512, 6198 régions.

TAB. 4.6 – Lena, 512×512, 6198 régions.

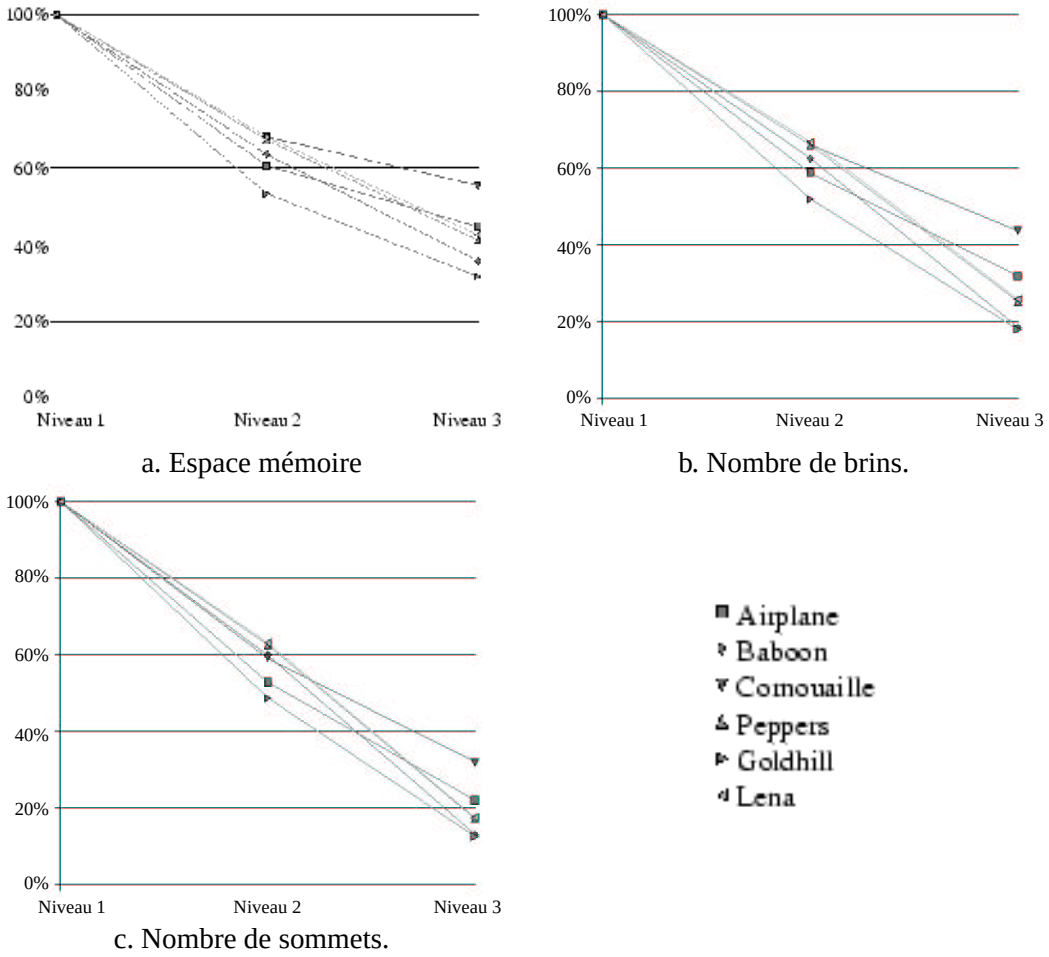
	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	109 188	72 920	28 226
Nb de sommets topologiques	49 057	30 923	8 576
Espace mémoire en kilo-octets	3 111	2 120	1 327
Temps de calcul en secondes	3,64	3,53	3,39

est en moyenne de l'ordre de 35%. La deuxième colonne donne le gain lorsque nous passons du niveau 2 au niveau 3 qui est du même ordre de grandeur. La dernière colonne donne le gain entre le niveau 1 et le niveau 3, qui est ici presque de 60%. Ces chiffres montrent qu'il est très intéressant de travailler avec le niveau 3 malgré le surcoût de difficulté d'implantation.

Nous pouvons également voir, sur ces divers exemples d'extraction, que le temps d'exécution ne varie que très peu pour les différents niveaux. Il y a même une légère diminution du temps nécessaire à l'extraction des cartes de plus grand niveau, ce qui peut sembler étrange au premier abord. Mais cette légère diminution provient en fait du nombre d'allocations mémoire qui est beaucoup plus important lorsque le niveau est petit, et qui entraîne donc un surcoût en temps d'exécution.

TAB. 4.7 – Récapitulatif des gains en espace mémoire.

	Gain Niveau 1 vers Niveau 2	Gain Niveau 2 vers Niveau 3	Gain Niveau 1 vers Niveau 3
Airplane	39,32%	26,13%	55,18%
Baboon	36,30%	43,90%	64,27%
Cornouaille	31,77%	18,81%	44,61%
Peppers	32,75%	38,62%	58,72%
Goldhill	46,70%	40,23%	68,14%
Lena	31,87%	37,37%	57,33%
Moyenne	36,45%	34,18%	58,04%



a. Espace mémoire

b. Nombre de brins.

c. Nombre de sommets.

FIG. 4.36 – Évolution des caractéristiques, en pourcentage du niveau 1.

4.6.2 Images médicales

Ces images proviennent d'examens IRM afin de détecter d'éventuelles tumeurs cérébrales. Elles résultent d'examens réalisés sur trois patients différents, et sont toutes de taille 256×256 pixels. Nous présentons figures 4.37 à 4.42 tout d'abord l'image segmentée, puis la carte résultante obtenue par notre algorithme d'extraction. Le nom de ces images est composé, tout d'abord du nom de l'examen, puis du numéro de la coupe correspondante, sachant que la première image correspond à la coupe inférieure, c'est-à-dire la plus proche de la base du crâne, et que la dernière correspond à la coupe supérieure.

Nous présentons dans un premier temps, comme pour les images standard, l'évolution du nombre de brins, du nombre de sommets topologiques et de l'espace mémoire total. Ces résultats sont présentés dans les tableaux 4.8 à 4.13, pour les trois niveaux de carte possibles.

Nous pouvons remarquer sur ces quelques résultats, que le temps d'exécution est beaucoup plus court que pour les images standard. En effet, tout d'abord les images sont ici plus petites,

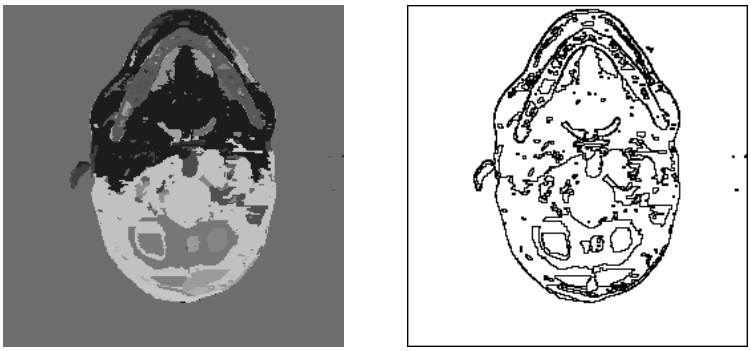


FIG. 4.37 – Pa2-i0004, 256×256, 771 régions.

TAB. 4.8 – Pa2-i0004, 256×256, 771 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	17 136	10 354	3 496
Nombre de sommets topologiques	7 910	4 519	1 090
Espace mémoire en kilo-octets	493	308	183
Temps de calcul en secondes	0,08	0,08	0,09

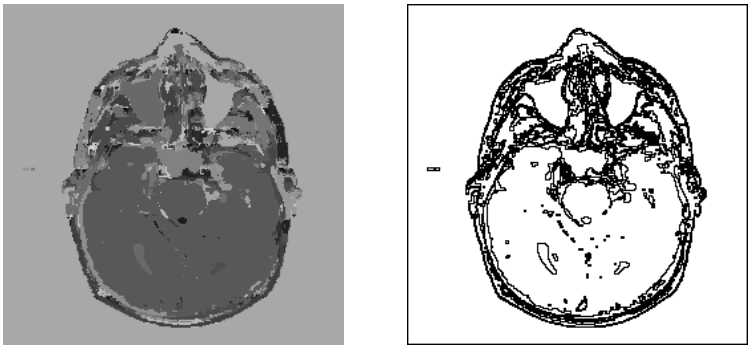
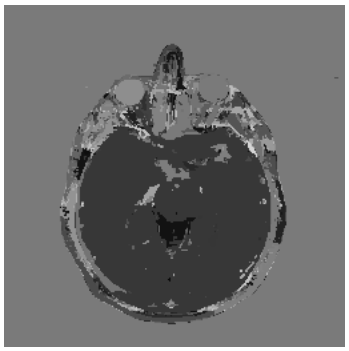


FIG. 4.38 – Pa2-i0038, 256×256, 2298 régions.

TAB. 4.9 – Pa2-i0038, 256×256, 2298 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	28 010	19 710	11 166
Nombre de sommets topologiques	11 811	7 661	3 389
Espace mémoire en kilo-octets	838	611	462
Temps de calcul en secondes	0,22	0,22	0,23



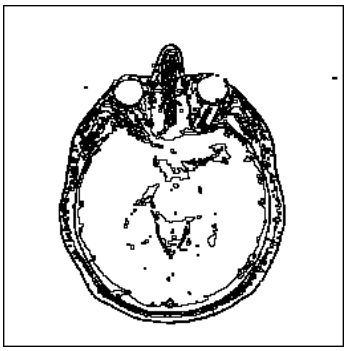
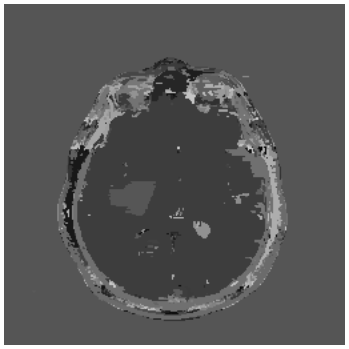


FIG. 4.39 – Pa5-i0030, 256×256, 1868 régions.

TAB. 4.10 – Pa5-i0030, 256×256, 1868 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	22 790	16 138	9 218
Nombre de sommets topologiques	9 605	6 279	2 819
Espace mémoire en kilo-octets	682	500	379
Temps de calcul en secondes	0,17	0,18	0,16



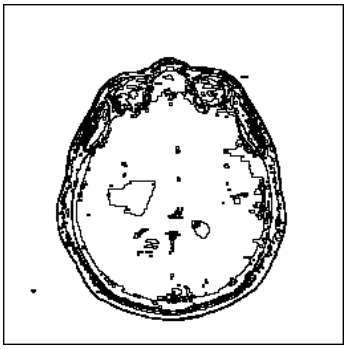


FIG. 4.40 – Pa5-i0046, 256×256, 1589 régions.

TAB. 4.11 – Pa5-i0046, 256×256, 1589 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	20 314	13 844	7 714
Nombre de sommets topologiques	8 655	5 420	2 355
Espace mémoire en kilo-octets	605	428	321
Temps de calcul en secondes	0,14	0,14	0,14

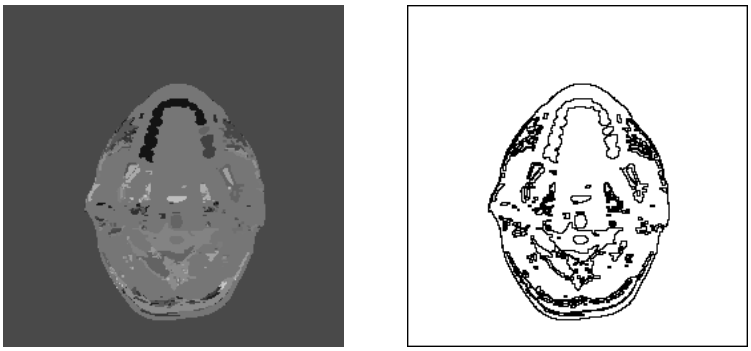


FIG. 4.41 – Pa6-i0012, 256×256, 595 régions.

TAB. 4.12 – Pa6-i0012, 256×256, 595 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	12 834	7 430	2 728
Nombre de sommets topologiques	5 894	3 192	841
Espace mémoire en kilo-octets	370	222	136
Temps de calcul en secondes	0,08	0,07	0,07

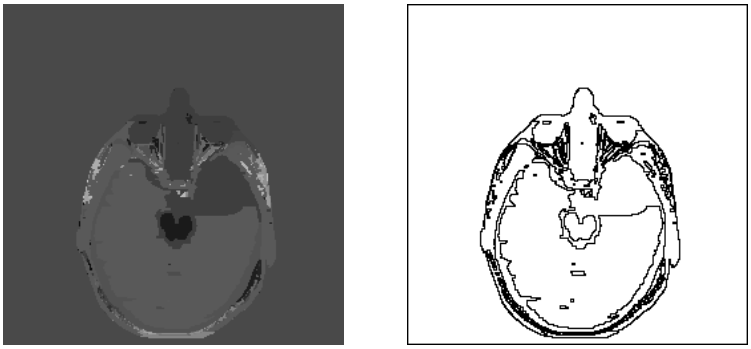
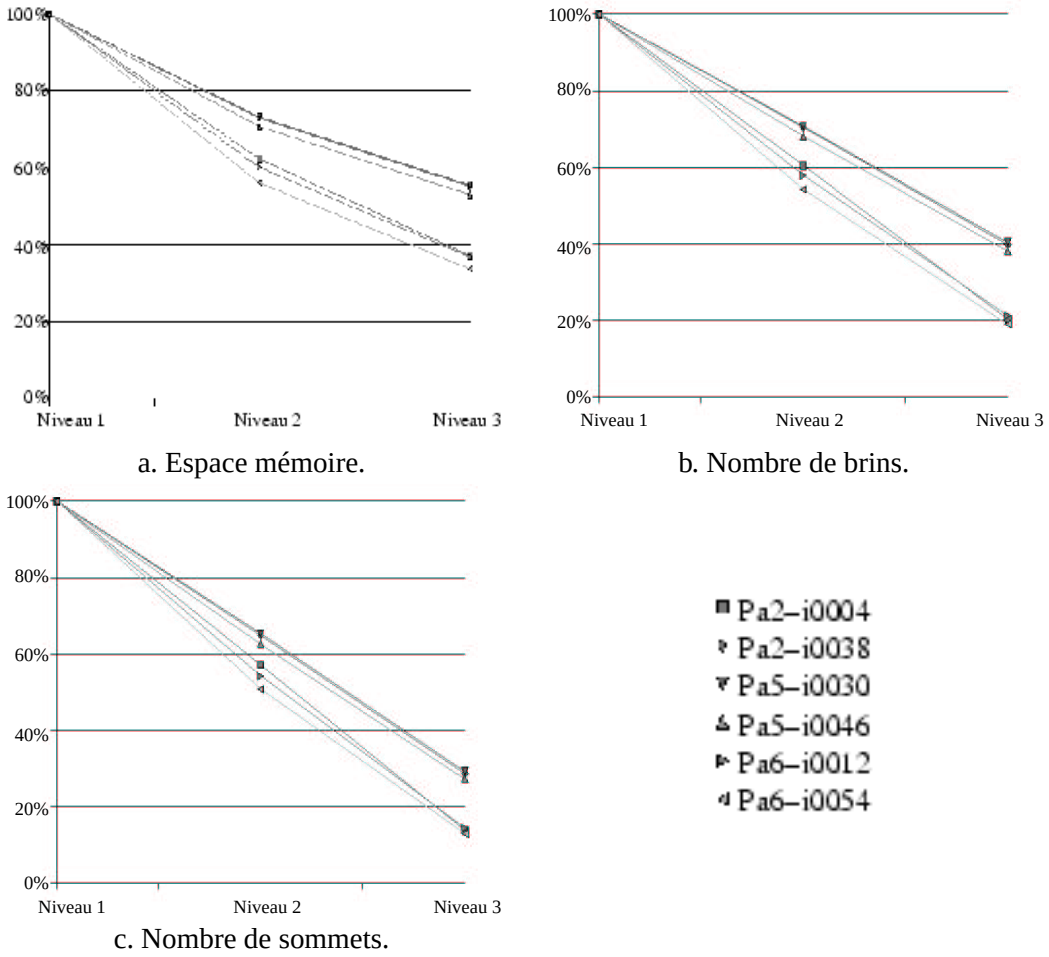


FIG. 4.42 – Pa6-i0054, 256×256, 452 régions.

TAB. 4.13 – Pa6-i0054, 256×256, 452 régions.

	Niveau 1	Niveau 2	Niveau 3
Nombre de brins	11 720	6 358	2 238
Nombre de sommets topologiques	5 441	2 760	700
Espace mémoire en kilo-octets	334	188	112
Temps de calcul en secondes	0,06	0,07	0,07



a. Espace mémoire.

b. Nombre de brins.

c. Nombre de sommets.

FIG. 4.43 – Évolution des caractéristiques, en pourcentage du niveau 1.

mais contiennent également beaucoup moins de régions. Afin de comparer, pour chacune de ces images, les différentes caractéristiques des trois niveaux de carte, nous présentons les résultats dans la figure 4.43 au moyen de courbes donnant l'espace mémoire, le nombre de brins et le nombre de sommets topologiques en pourcentage du niveau 1.

Cette figure permet de visualiser plus facilement l'évolution de ces caractéristiques pour les trois niveaux. Nous pouvons remarquer que le gain est à peu près comparable à celui des images standard, pour les trois caractéristiques. Nous pouvons vérifier plus précisément cette évolution pour l'espace mémoire dans le tableau 4.14 donnant le pourcentage de gain entre chaque niveau de carte.

Ici aussi, comme pour les images standard, nous observons des gains de l'ordre de 35% entre le niveau 1 et le niveau 2, de l'ordre de 30% entre le niveau 2 et le niveau 3 et de l'ordre de 55% entre le niveau 1 et le niveau 3.

TAB. 4.14 – Récapitulatif des gains en espace mémoire.

	Gain Niveau 1 vers Niveau 2	Gain Niveau 2 vers Niveau 3	Gain Niveau 1 vers Niveau 3
Pa2-i0004	37,57%	40,44%	62,81%
Pa2-i0038	27,06%	24,40%	44,86%
Pa5-i0030	26,66%	24,08%	44,32%
Pa5-i0046	29,20%	25,05%	46,93%
Pa6-i0012	39,92%	38,63%	63,13%
Pa6-i0054	43,78%	40,06%	66,30%
Moyenne	34,03%	32,11%	54,73%

4.7 Conclusion

Dans ce chapitre, nous avons défini un modèle combinatoire permettant de représenter les images segmentées en dimension 2, donné deux algorithmes d'extraction de ce modèle : un premier simple et facile à implanter, et un deuxième optimal s'appuyant sur la notion de précode. Nous avons également expliqué formellement cette notion de précode, et introduit une nouvelle notion permettant de regrouper plusieurs précodes. Enfin, nous avons testé et comparé nos algorithmes afin d'étudier précisément l'évolution de différentes caractéristiques sur différentes images.

De nombreux modèles existaient déjà en dimension 2, et notre modèle est similaire à deux d'entre eux : le TGF et les cartes discrètes. Notre apport, pour cette dimension, n'est pas d'avoir défini un nouveau modèle, qui de surcroît ne peut pas vraiment être considéré comme nouveau, mais plutôt de l'avoir défini de manière différente. En effet, nous avons introduit un nouveau concept de niveau de simplification permettant de donner une définition simple, générique, et amenant directement à la définition des algorithmes d'extraction. Notre approche progressive permet la définition de la carte topologique de manière hiérarchique, chaque étape pouvant se définir simplement à partir de l'étape précédente et d'un type particulier de fusion.

Cette définition progressive nous a ensuite permis de définir de manière immédiate un premier algorithme d'extraction, qui découle directement de cette définition. Cet algorithme est simple à comprendre et à implanter, mais n'est pas optimal, bien que linéaire en le nombre de pixels de l'image. En effet, il construit la carte topologique en plusieurs passes de l'image.

Nous avons donc ensuite présenté un algorithme optimal d'extraction de n'importe quel niveau de carte à partir d'une image segmentée en régions. Cet algorithme est générique, simple, et effectue la construction de n'importe quel niveau de carte en une unique passe de l'image. Pour cela, nous avons étudié précisément la notion de précode et de précode partiel, et cherché quels précodes nous devions traiter, et de quelle manière, pour chacun des différents niveaux de carte. Notre définition progressive nous a ici permis d'étudier ces précodes de manière simple, en étudiant à chaque niveau uniquement les nouveaux précodes à traiter par rapport au niveau précédent. Nous obtenons au final une hiérarchie de précodes classés par niveau, qui nous permet d'obtenir immédiatement les précodes à traiter afin d'extraire n'importe quel niveau de carte. Cette définition progressive nous a également permis de factoriser ces précodes sans difficulté supplémentaire, alors que cela aurait demandé beaucoup plus de travail et d'attention avec une définition directe.

Nous obtenons finalement 4 précodes pour le niveau 1, 6 pour le niveau 2 et 10 pour le niveau 3, au lieu des 15 précodes existant en dimension 2 (ou des 12 précodes variétés).

Ces avantages justifient donc ce travail en dimension 2. Mais cette définition présente encore un dernier avantage, qui est en fait le plus important. En effet, notre définition progressive peut s'étendre en dimension supérieure. De plus, les algorithmes utilisés en dimension 2 vont pouvoir également être étendus en dimension supérieure, et les méthodes de travail utilisées ici afin de définir les précodes nécessaires à chaque niveau vont être similaires. Et c'est en dimension supérieure que les avantages apportés par la simplicité de notre définition seront des avantages primordiaux dans la définition d'un modèle représentant les images segmentées en dimension n.

CHAPITRE 5

LA CARTE TOPOLOGIQUE EN DIMENSION 3

Nous définissons dans un premier temps la carte topologique en dimension 3, en étendant notre définition progressive donnée en dimension 2. Cette extension ne pose pas de problème majeur, à l'exception des problèmes de déconnexion. Ce type de problème se produit déjà en dimension 2, mais uniquement entre des régions, et est résolu par l'adjonction d'un arbre d'inclusion. Pour la dimension 3, ce problème se pose également pour les régions et la solution est toujours l'adjonction d'un arbre d'inclusion. Mais un autre problème de déconnexion apparaît pour les faces, et n'a pas d'équivalent en dimension 2. La résolution de ce problème consiste à conserver des éléments fictifs, ici des *arêtes fictives*, permettant que chaque face reste homéomorphe à un disque topologique. Nous étudions précisément ces arêtes fictives, afin de déterminer leur rôle et leurs propriétés. Elles jouent en effet un rôle crucial dans l'obtention d'une représentation minimale.

Nous présentons ensuite des algorithmes d'extraction. L'algorithme naïf ne pose pas de problème particulier, comme en dimension 2. L'algorithme optimal est également simple à définir en dimension 3, étant donné qu'il se résume à un balayage de l'image en exécutant le code correspondant au précode courant. Mais comme pour la dimension 2, la difficulté réside dans la définition des précodes à traiter pour chaque niveau de carte. Nous avons effectué une étude précise de ces précodes afin de définir exactement ceux à traiter pour extraire chaque niveau de carte, mais également afin de factoriser ceux pouvant se traiter de manière similaire.

Le plan de ce chapitre est volontairement très proche de celui du chapitre précédent présentant la carte topologique en dimension 2 pour mieux mettre en évidence les similarités ou les différences entre ces deux dimensions. Nous commençons section 5.1 par un rappel sur les images 3d segmentées en régions, et sur les frontières dans ces images, qui peuvent être définies à l'aide de la notion d'intervoxel. Mais nous avons vu en dimension 2 que la définition des frontières interpixel est inutile pour définir la carte topologique. Nous ne définissons donc pas formellement cette notion. Section 5.2, nous définissons la carte topologique en dimension 3. Puis section 5.3 nous traitons les problèmes de déconnexion et étudions en détail les éléments fictifs. Section 5.4 nous présentons quelques plongements possibles pour la carte topologique. Section 5.5 nous étudions l'algorithme d'extraction naïf qui découle, comme pour la dimension 2, directement de la définition de la carte topologique. Puis section 5.6 nous présentons l'algorithme optimal d'extraction, ainsi que les précodes à traiter pour extraire chaque niveau de carte. Section 5.7, nous donnons

un troisième algorithme d'extraction, qui est une solution intermédiaire intéressante entre les deux premiers algorithmes. Enfin section 5.8 nous analysons quelques résultats et étudions l'évolution de quelques caractéristiques sur les différents niveaux de carte, la section 5.9 conclut ce chapitre.

5.1 Images, segmentation en régions et intervoxel

Il s'agit de représenter des images 3d segmentées en régions. Nous rappelons d'abord cette notion, puis présentons la notion d'intervoxel et discutons de la notion de frontière en dimension 3, en montrant en quoi la définition de ces frontières est plus complexe qu'en dimension 2.

5.1.1 La segmentation en régions

Un *voxel* est un point coloré de l'espace discret $\mathbb{Z}^3$, et une *image* est un ensemble fini de voxels. Nous utilisons en 3d les notions de 6-*voisinage* et de 6-*connexité*. La notion de *segmentation en régions* est similaire à celle présentée en dimension 2, sauf que chaque élément de la partition, c'est-à-dire chaque région, doit maintenant être un ensemble 6-connexe. Comme pour la dimension 2, nous considérons que l'image est incluse dans une *région infinie* R_0 contenant l'ensemble des voxels n'appartenant pas à l'image. La notion d'inclusion s'étend également immédiatement en dimension 3.

Définition 22 (inclusion) *Une région R_i est* incluse *dans une région R_j si et seulement si tout chemin 6-connexe allant d'un voxel de R_i vers un voxel de R_0 (la région infinie) possède au moins un voxel appartenant à la région R_j.*

Nous ne revenons pas sur les propriétés de cette relation, qui sont les mêmes que pour la dimension 2. Il en va de même pour la relation *d'inclusion directe* présentée définition 13, définie à partir de la relation d'inclusion, et qui reste donc valable ici.

5.1.2 Intervoxel et frontières

L'*intervoxel* est simplement l'extension en dimension 3 de l'interpixel défini en dimension 2. L'espace de dimension 3 est composé de voxels, et nous considérons la subdivision de cet espace en cellules de dimension inférieure ou égale. Cette subdivision représente des *voxels*, cellules de dimension 3, des *surfels*, cellules de dimension 2 « entre » deux voxels, des *lignels*, cellules de dimension 1 « entre » deux surfels et des *pointels*, cellules de dimension 0 « entre » deux lignels. Nous parlons, comme pour la dimension 2, de *i-cellule* pour une cellule de dimension i. Ces différentes cellules sont présentées figure 5.1.

Les notions d'*adjacence* et d'*incidence*, présentées en dimension 2 section 4.1.2, restent valides en dimension 3. Informellement, une *frontière* entre deux régions est l'ensemble des cellules se trouvant entre ces deux régions. En dimension 2, une frontière est une courbe 1d. En effet, comme chaque région est 4-connexe, de par la définition de la segmentation en régions, une frontière ne peut pas être composée seulement d'un pointel. En dimension 3, une frontière est maintenant une surface, du fait de la définition de la segmentation en régions qui implique que chaque

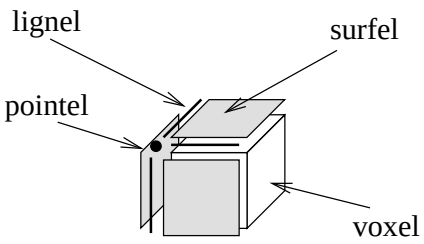


FIG. 5.1 – Les divers éléments intervoxel en dimension 3.

région soit 6-connexe. C'est ici qu'apparaît une difficulté supplémentaire. En effet, il faut étudier les bords de ces surfaces frontières, car ce sont eux qui sont représentés par les cartes combinatoires.

Le problème de définition de surface en dimension 3 est difficile et relève du domaine de la topologie discrète. Il a été étudié par de nombreux travaux qui utilisent la topologie discrète afin de définir les surfaces comme des ensembles de 2-cellules, ou surfels, qui sont les bords des voxels. Des algorithmes de détection de telles surfaces par suivi de contour ont tout d'abord été proposés par [AFH81, GU89]. Les preuves de la validité de ces algorithmes ont été publiées seulement quelques années plus tard dans [HW83, KU92] de par les difficultés de celles-ci. Mais ces deux approches ne s'intéressent pas à la définition théorique de ces surfaces.

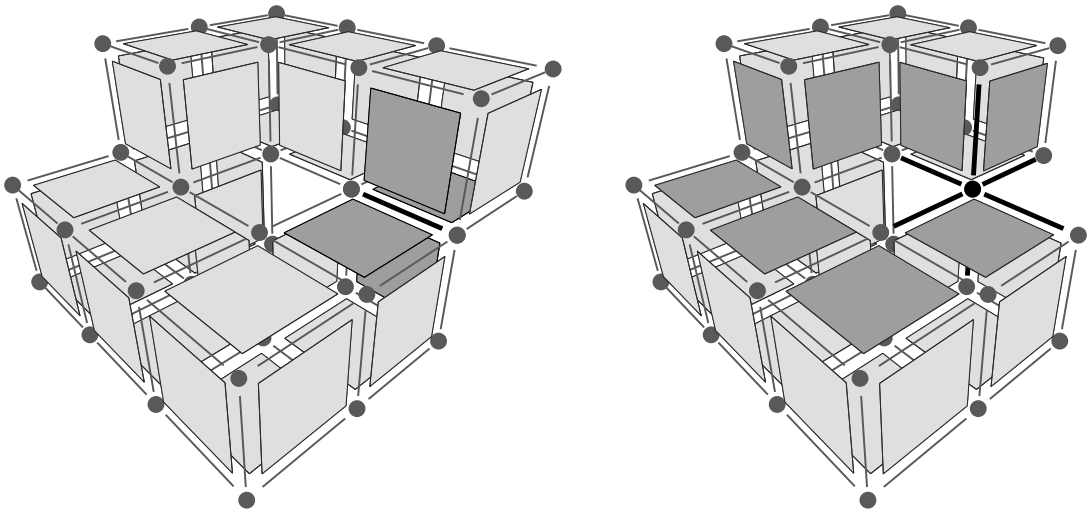
D'autres travaux plus théoriques se sont penchés sur la définition des surfaces dans le cadre de la topologie interpixel. [KF81] prouve qu'un ensemble de faces est une 2-variété si elle satisfait aux trois conditions suivantes :

- les faces s'intersectent seulement par des arêtes ou des sommets ;
- chaque arête appartient seulement à deux faces ;
- les faces autour d'un même sommet peuvent être ordonnées de sorte que deux faces consécutives pour cet ordre s'intersectent par une arête incidente à ce sommet.

Ces surfaces ont été étudiées également par [RKW91]. Mais elles ne sont pas utilisables pour notre travail, car elles sont définies uniquement pour des objets noir et blanc, afin d'utiliser une connexité différente pour l'objet et pour le fond.

Nous présentons ici les propriétés que nous aimerions voir satisfaites par les surfaces que nous traitons et montrons les problèmes que cela soulève. Cela nous permet lors de la présentation de la carte topologique section 5.2, de montrer comment ces problèmes sont résolus avec les cartes combinatoires alors que ce n'est pas le cas en intervoxel.

Il faut tout d'abord que la définition des surfaces en dimension 3 soit « cohérente » avec les objets en dimension 2, tout simplement car une surface en dimension 3 est un objet 2d. Mais cette propriété pose un premier problème. En effet, en dimension 2 chaque lignel d'un objet 2d est incident à un ou deux surfels de cet objet. Mais cela n'est plus vrai pour les surfaces en dimension 3, comme nous pouvons voir sur l'exemple présenté figure 5.2.*a*. Cet exemple montre un objet 3d, représenté par sa surface. Nous pouvons voir qu'elle possède un lignel incident à 4 surfels, ce qui est impossible en dimension 2. Cela pose des problèmes dans la définition des surfaces, car nous devons différencier ce type de cas de ceux non valides de surfaces repliées sur elles-mêmes. Une solution, utilisée par exemple dans [AAF95], consiste dans ce cas à ne pas considérer ce lignel



a. Un lignel incident à 4 surfels. b. Un pointel incident à 6 lignels.

FIG. 5.2 – Deux exemples de problèmes pour la définition des surfaces.

comme faisant partie de la surface. Mais cette solution n'est pas très satisfaisante, car des trous apparaissent localement, ce qui n'est pas cohérent avec les définitions existantes en dimension 2.

Il faut également que les bords des surfaces soient des courbes de dimension 1. En effet, les cartes combinatoires représentent le bord des faces. Cette propriété pose un autre problème, comme nous pouvons le voir figure 5.2.*b*. Dans ce cas, un pointel est incident à 6 lignels, et la courbe 1d représentant le bord de la surface en gris foncé[1] passe par 4 de ces lignels, et deux fois par le pointel. Il faut donc redéfinir la notion de courbe 1d afin d'autoriser ce type de courbe, tout en interdisant certaines configurations impossibles. Un problème similaire se pose pour l'objet de la figure 5.2.*a*, où un lignel appartient deux fois dans la même courbe 1d.

Définir les surfaces en dimension 3 est donc un problème difficile. Mais nous avons vu au chapitre précédent, lors de la définition de la carte topologique en dimension 2, que la définition des frontières interpixel n'est pas nécessaire pour définir cette carte. En effet, cette définition s'effectue uniquement à partir de la carte complète, qui représente toutes les cellules de l'espace sans utiliser la notion de frontière, puis par application de plusieurs opérations de fusions qui n'ont plus aucun lien avec les frontières interpixel. Étant donné que nous étendons cette définition de la carte topologique en dimension 3, nous n'avons donc pas besoin de la définition précise des frontières intervoxel car nous conservons le même principe de définition progressive basé sur nos différents niveaux de simplification.

Nous utilisons ces frontières afin de prouver que la carte topologique représente bien toutes les propriétés topologiques de l'image. Pour cela, nous définissons partiellement ces frontières, uniquement pour leurs surfels. Une *frontière* intervoxel entre deux régions R_i et R_j est un ensemble de surfaces, tel que chaque surfel de celles-ci est incident exactement à un voxel de R_i et à un voxel de R_j. Nous appelons *surface frontière* une surface appartenant à une des frontières de

[1] Cette surface pourrait être une frontière entre l'objet 3d représenté sur cette figure et un autre objet posé dessus.

l'image. Ces surfaces sont maximales : tout surfel de l'image incident à un voxel de R_i et à un voxel de R_j appartient forcément à une de ces surfaces, et deux surfels adjacents appartenant à une frontière appartiennent forcément à la même surface frontière.

Cette définition est incomplète, tout d'abord car elle s'appuie sur la définition de surface que nous n'avons pas donnée. De ce fait, nous n'avons pas de contraintes sur les lignels et pointels, et ne pouvons donc pas définir le bord de ces frontières en termes de courbes 1d. Cette définition serait nécessaire afin de définir la carte topologique de manière directe. Nous conservons les mêmes notations déjà présentées en dimension 2. Nous notons $frontière(R_i, R_j) = \{s_1, \ldots, s_l\}$ la frontière entre les régions R_i et R_j composée des surfaces $s_1 \ldots s_l$, posons $frontière(R_i, R_i) = \emptyset$ et notons $Frontières(R_i) = \cup_{j=1}^{k} frontière(R_i, R_j)$. Nous parlons de *surfel frontière* pour un surfel appartenant à une frontière de l'image.

5.2 Les niveaux de simplification

Afin de définir la carte topologique en dimension 3, nous reprenons la définition progressive donnée en dimension 2, et la modifions afin de tenir compte des différences avec la dimension 3. Nous partons de la carte combinatoire complète représentant chaque cellule de l'espace intervoxel, que nous appelons comme en dimension 2 la carte de niveau 0. Puis nous la simplifions progressivement au moyen de l'opération de fusion, en construisant des niveaux de simplification intermédiaires, jusqu'à obtenir la carte minimale qui ne peut plus être simplifiée sans perte d'information topologique.

Nous présentons dans un premier temps, comme en dimension 2, uniquement l'aspect topologique, afin de bien le séparer des aspects géométriques. Le lien avec des éventuels modèles de plongements est l'objet de la section 5.4. Par rapport à la dimension 2, un problème de déconnexion supplémentaire apparaît. Il faut le résoudre afin de définir correctement la carte topologique. Comme pour le plongement, nous commençons par ne pas traiter ces problèmes de déconnexion, ce qui permet de définir simplement nos différents niveaux de carte. Puis, section 5.3, nous résolvons ces problèmes.

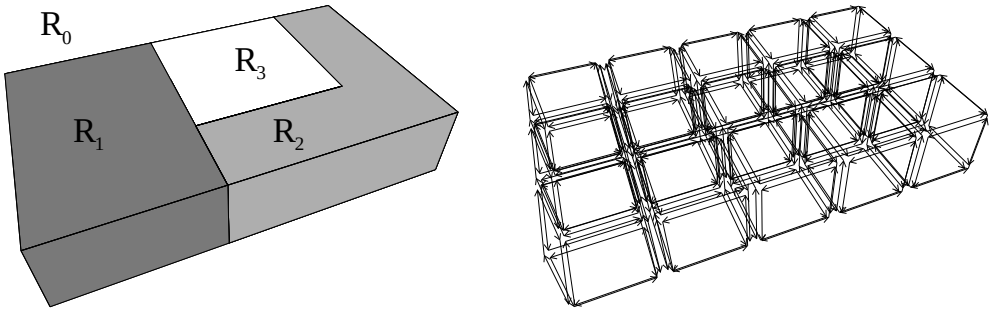
5.2.1 Le niveau 0 : la carte complète

Cette première carte est, comme en 2d, le point de départ de notre processus, et ne code pas les frontières intervoxel de l'image.

Définition 23 (carte de niveau 0) *La* carte de niveau 0 *correspondant à une image de $n_1 \times n_2 \times n_3$ voxels, est la carte combinatoire ayant $n_1 \times n_2 \times n_3$ cubes, β_3-cousus entre eux lorsqu'ils sont adjacents. Chacun de ces cubes représente un voxel de l'image, et un volume « entourant » ces cubes, représente la région infinie.*

Cette carte code tous les éléments intervoxel de l'image. La figure 4.4.*a* représente une image 3d segmentée en régions, et la figure 4.4.*b* la carte de niveau 0 correspondante.

Cette carte est composée, pour une image de taille $n_1 \times n_2 \times n_3$, de $(n_1 \times n_2 \times n_3) + 1$. $n_1 \times n_2 \times n_3$ cubes composés chacun de 6 faces carrées, chacune composée par 4 brins, représentant



a. Une image 3d segmentée en régions. b. La carte de niveau 0 correspondante.

FIG. 5.3 – Le niveau 0 d'une image 3d.

l'ensemble des voxels de l'image. Le volume supplémentaire représente la région infinie, et est composé de $2 \times (n_1 \times n_2 + n_1 \times n_3 + n_2 \times n_3)$ faces carrées. Ces faces sont cousues par β_3 aux faces extérieures des cubes correspondants. Nous ne représentons pas ce volume infini sur toutes les figures comme par exemple sur la figure 5.3.*b*, afin de ne pas surcharger les schémas. Pour cette même raison, nous avons volontairement choisi une image de petite taille pour cet exemple, de $5 \times 3 \times 1$ voxels.

Cette carte de niveau 0 code donc tous les éléments intervoxel de l'image ainsi que toutes les relations d'adjacence et d'incidence. Comme en dimension 2, nous la simplifions progressivement afin de définir la carte topologique.

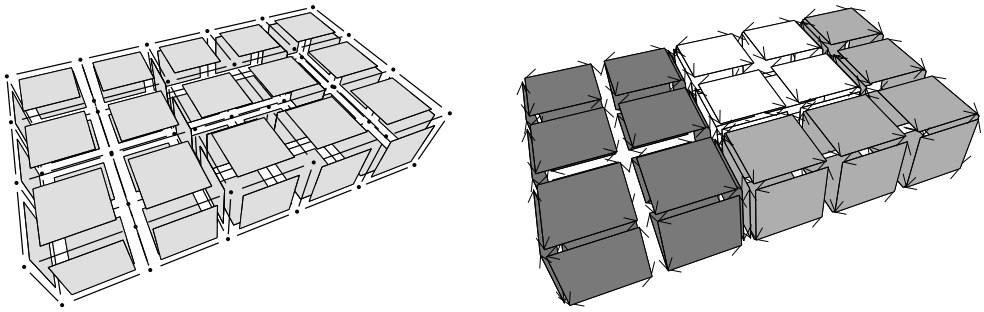
5.2.2 Le niveau 1 : la carte lignel

Pour définir cette première carte représentant les frontières intervoxel de l'image, nous « supprimons » les faces de la carte de niveau 0 représentant des surfels n'appartenant pas à une surface frontière de l'image. Comme pour la dimension 2, nous utilisons l'opération de fusion, détaillée section 5.5.

Définition 24 (carte de niveau 1) *La* carte de niveau 1 *est la carte obtenue à partir de la carte de niveau* 0 *en fusionnant chaque couple de volumes adjacents appartenant à la même région.*

Nous pouvons voir figure 5.4.*a* notre image d'exemple, présentée avec ses éléments intervoxel, et figure 5.4.*b* la carte de niveau 1 correspondante.

Nous vérifions facilement que chaque face de cette carte représente bien un surfel frontière de l'image. En effet, pour la carte de niveau 0, chaque face représente un surfel de l'image. Pour obtenir la carte de niveau 1, nous avons fusionné chaque couple de volumes adjacents et de même région. Cette fusion a donc entraîné la disparition des faces entre deux voxels de même région. Il ne reste donc plus, dans la carte de niveau 1, que les faces entre deux voxels de régions différentes qui représentent bien toutes des surfels frontière. De plus, chaque arête de cette carte est de longueur unitaire, car cette propriété est vérifiée par construction de la carte de niveau 0, et nous n'avons pas modifié la longueur des arêtes entre ces deux niveaux. De ce fait, chaque arête de la carte de



a. L'image et ses frontières intervoxel. b. La carte de niveau 1 correspondante.

FIG. 5.4 – La carte de niveau 1.

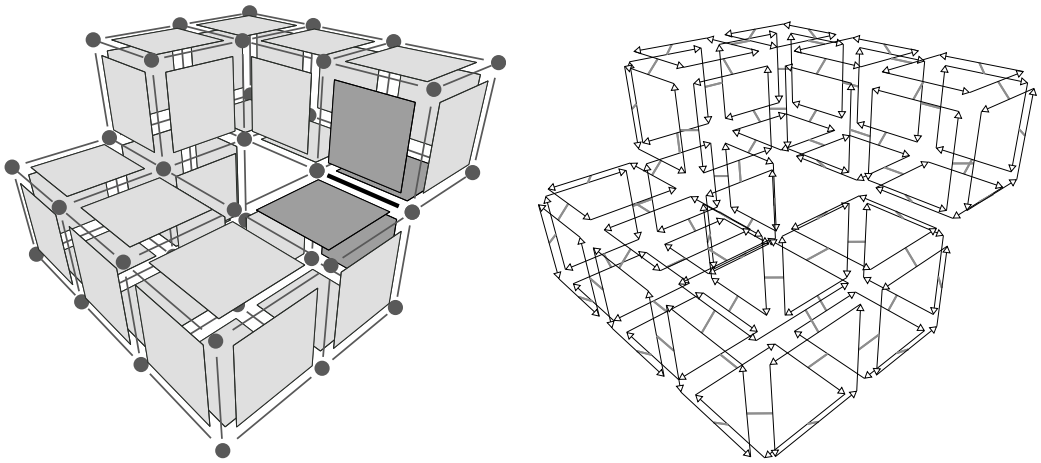
niveau 1 représente un lignel de l'image. C'est pour cette raison que nous appelons cette carte la *carte lignel*.

Comme pour la dimension 2, cette carte de niveau 1 représente chaque région de l'image par une surface extérieure, et éventuellement plusieurs surfaces intérieures. Pour les surfaces extérieures, cette remarque découle simplement du fait que chaque région de l'image est finie et possède donc une surface extérieure, à l'exception de la région infinie R_0. Les surfaces intérieures apparaissent seulement lorsqu'une région est totalement incluse dans une autre. Dans ce cas, la surface extérieure est déconnectée des éventuelles surfaces intérieures. Il existe alors plusieurs composantes connexes dans la carte, et nous avons perdu les informations permettant de les positionner les unes par rapport aux autres. Cette information d'inclusion est une information topologique importante qu'il faut conserver. Nous verrons section 5.3 que ce problème se résout, comme pour la dimension 2, par l'adjonction d'un arbre d'inclusion des régions.

Cette carte lignel dotée d'un arbre d'inclusion représente bien toutes les propriétés topologiques contenues dans l'image. En effet, chaque surfel frontière est représenté, et toutes les relations d'incidence et d'adjacence sont correctement représentées par la carte combinatoire. Les problèmes évoqués lors de la définition de surface en intervoxel n'en sont plus ici, comme nous pouvons le vérifier figure 5.5.*b* où nous présentons la carte de niveau 1 de l'objet 3d présenté figure 5.5.*a* qui posait problème pour la définition de surface.

Sur cet exemple, nous considérons que l'objet 3d présenté figure 5.5.*a* est totalement inclus dans une autre région. De ce fait, il ne possède qu'une seule face frontière avec cette région. Afin de ne pas surcharger la figure, nous ne représentons pas les faces de cette région, qui seraient normalement des copies de toutes les faces de l'objet représenté, cousues entre elles par β_3. Nous avons représenté la relation β_2 par de petits segments gris.

Nous voyons sur la carte de niveau 1 représentant cet objet qu'il n'y a pas de problème relatif au lignel incident à 4 surfels. En effet, la surface frontière de l'objet est obtenue par un parcours d'orbite $< \beta_1, \beta_2 >$ d'origine un brin de cette surface. Nous parcourons donc correctement cette surface frontière. De plus, étant donné que nous ne considérons pas β_3, deux arêtes distinctes représentent ce lignel. Nous obtenons donc bien une surface, qui est une carte combinatoire 2d, où chaque arête est toujours incidente à deux faces. Du point de vue de l'objet, tout se passe au niveau de ce lignel comme s'il y avait deux arêtes topologiques différentes : nous ne représentons pas la



a. Un objet 3d. b. La carte de niveau 1 correspondante.

FIG. 5.5 – La carte de niveau 1 d'un objet 3d ayant un lignel incident à 4 surfels.

18-adjacence. Ce n'est plus vrai si nous considérons les brins de la région infinie, et donc cette arête non plus uniquement pour l'objet mais pour l'objet et son complémentaire. Dans ce cas, nous obtenons une seule arête topologique qui représente le lignel correspondant dans la subdivision de l'espace.

Remarquons que la construction de cette carte de niveau 1 à partir de la carte de niveau 0 n'utilise pas la notion de degré de cellule, mais plus simplement l'information de région. En effet, une face est toujours incidente à un ou deux volumes, et nous devons effectuer la fusion de deux volumes uniquement lorsqu'ils appartiennent à la même région. De plus, c'est uniquement pour ce niveau de carte que nous effectuons un lien entre la carte et l'image segmentée en régions. Les autres niveaux vont désormais être construits uniquement à partir de ce premier niveau de carte, en effectuant certaines fusions suivant le degré de cellules, sans utiliser l'image segmentée en régions.

5.2.3 Le niveau 2

Afin de définir la carte topologique, nous suivons le même principe que celui utilisé en dimension 2. Nous avons tout d'abord défini la carte de niveau 0 qui représente tous les éléments intervoxel de l'image, puis la carte de niveau 1 en fusionnant les volumes appartenant à la même région. Nous effectuons maintenant la fusion des cellules de degré deux, qui comme en dimension 2, représentent la même information topologique. Mais par rapport à la dimension 2, il existe un type de fusion supplémentaire. En effet, il existe la fusion de volumes, déjà utilisée pour définir la carte de niveau 1, la fusion de faces et la fusion d'arêtes. Il faut appliquer la fusion de faces avant la fusion d'arêtes, car sinon il n'existe pas d'arêtes incidentes à des sommets de degré deux étant donné que les faces n'ont pas encore été fusionnées. De manière générale, nous devons effectuer les fusions par dimension décroissante, en commençant par la fusion de même dimension que l'espace.

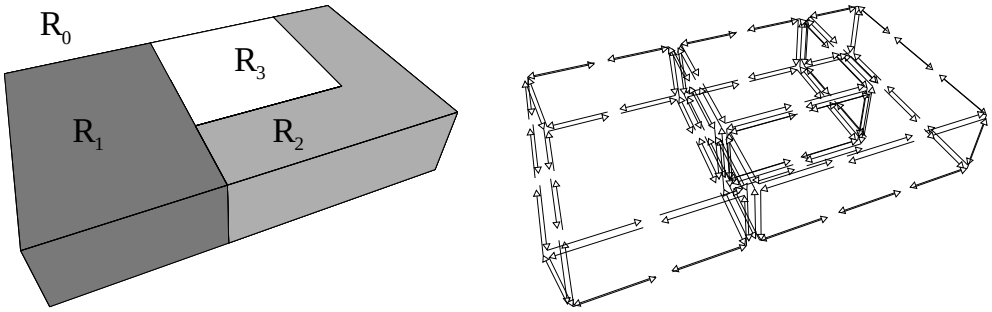


FIG. 5.6 – La carte de niveau 2.

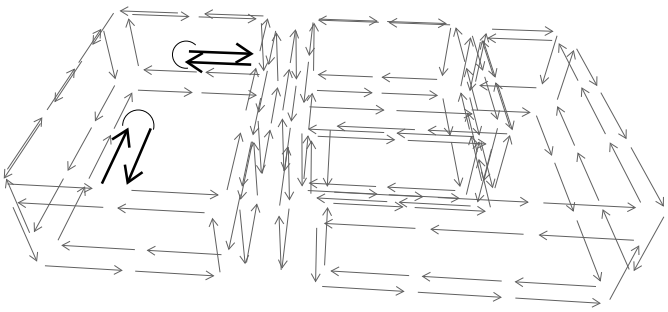


FIG. 5.7 – La carte de niveau 2 en cours de construction.

Définition 25 (carte de niveau 2) *La* carte de niveau 2 *est la carte obtenue à partir de la carte de niveau* 1*, en fusionnant chaque couple de faces adjacentes, coplanaires, et incidentes à une arête de degré un ou deux.*

Comme pour la dimension 2, nous commençons dans un premier temps par fusionner uniquement les cellules « alignées », i.e. les faces coplanaires, afin d'obtenir la définition de la carte des frontières comme l'un des niveaux de simplification. Rappelons que cette carte des frontières présente l'avantage de pouvoir être plongée simplement, en associant à chacun de ses sommets topologiques un point géométrique de l'espace, étant donné que chacune de ses arêtes représente un segment de droite. De plus, le fait de simplifier la carte petit à petit nous permet ensuite de décomposer plus finement l'étude sur les précodes.

La carte de niveau 2 de notre image d'exemple est présentée figure 5.6. Nous voyons sur cet exemple que les faces coplanaires incidentes à des arêtes de degré deux ont été fusionnées. Par exemple la face supérieure de la région R_1 est désormais représentée par une seule face composée de 10 brins.

Mais contrairement aux différents niveaux définis en dimension 2, nous fusionnons, pour cette définition, les faces adjacentes, coplanaires et incidentes à une arête de degré *un ou deux*. En effet, au fur et à mesure des fusions, certaines arêtes qui étaient de degré deux, peuvent devenir de degré un, c'est-à-dire incidentes à une seule face, comme le montre la figure 5.7. Cette figure présente la

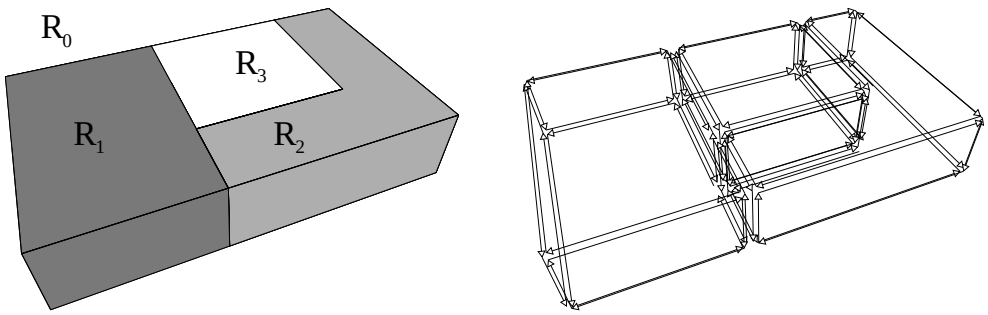


FIG. 5.8 – La carte de niveau 3.

carte de niveau 2 en cours de construction. Nous n'avons pas effectué toutes les fusions de faces coplanaires nécessaires. Il reste en effet les deux arêtes noires, incidentes à une seule et même face. Mais ces deux arêtes étaient de degré deux dans la carte de niveau 1 initiale, et sont devenues de degré un au fur et à mesure des fusions. Nous devons supprimer ces deux arêtes par deux fusions de faces, afin de ne pas dépendre de l'ordre dans lequel sont effectuées les fusions. De plus, ces arêtes n'ont aucune raison d'être conservées, car elles ne représentent pas une partie d'un bord d'une face frontière. Nous verrons au chapitre 6 qu'une autre solution consiste à considérer que l'arête est incidente localement à deux faces différentes.

Ces fusions de faces sont similaires aux fusions de faces effectuées en dimension 2 pour définir la carte de niveau 1. De ce fait, nous pouvons avoir le même problème de déconnexion que celui rencontré en dimension 2. Nous verrons section 5.3 comment il peut se produire et comment le résoudre.

5.2.4 Le niveau 3 : la carte des frontières

Après avoir fusionné les faces adjacentes, coplanaires, et incidentes à des arêtes de degré un ou deux, nous fusionnons maintenant les arêtes alignées incidentes à des sommets de degré deux. Nous obtenons alors une carte où chaque arête représente un segment de droite et peut donc être plongée uniquement par ses sommets. C'est cette carte qui est définie dans [BFP99] et qui est appelée *carte des frontières*.

Définition 26 (carte de niveau 3) *La* carte de niveau 3 *est la carte obtenue à partir de la carte de niveau* 2*, en fusionnant chaque couple d'arêtes adjacentes, alignées, et incidentes à un sommet de degré deux.*

La carte de niveau 3 de notre exemple est présentée figure 5.8. Comme pour la dimension 2, chaque arête d'une carte des frontières correspond à un segment de droite maximal du bord d'une frontière de l'image initiale. Contrairement à la définition de la carte de niveau 2, il ne faut pas ici fusionner les arêtes incidentes à un sommet de degré un. En effet, dans ce cas cette arête forme une boucle autour de ce sommet, et nous ne pouvons ni ne devons effectuer la fusion de cette arête avec elle-même. Remarquons enfin que, comme pour la dimension 2, cette fusion d'arêtes ne

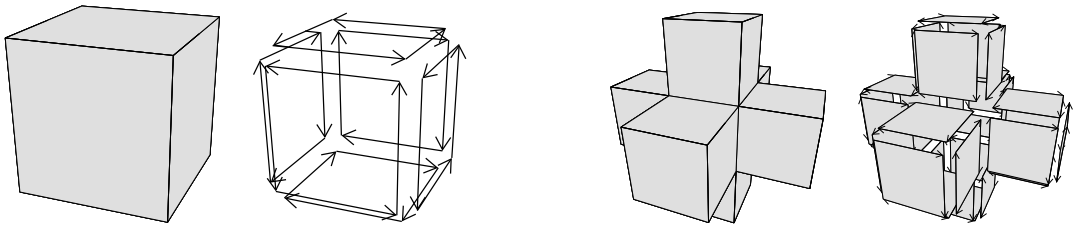


FIG. 5.9 – Deux objets topologiquement équivalents et leurs cartes des frontières.

peut pas entraîner de déconnexion étant donné qu'elle revient à étirer les arêtes, tout en conservant exactement les mêmes relations topologiques.

Ce niveau de carte présente les mêmes inconvénients qu'en dimension 2 : un même objet topologique sera représenté par des cartes totalement différentes suivant sa géométrie. Ce niveau n'est pas stable par rotation, translation et homothétie, comme nous pouvons le vérifier sur la figure 5.9. Cette figure montre deux objets 3d topologiquement équivalents, et les cartes des frontières les représentants. Ces deux cartes sont totalement différentes. De plus, la représentation de ces objets n'est pas minimale, et les cartes de ce niveau pourront avoir beaucoup de brins si la géométrie des objets contenus dans l'image est complexe.

5.2.5 Le niveau 4

Avec la carte des frontières, nous avons simplifié au maximum les faces coplanaires et les arêtes alignées. Afin de continuer notre processus, nous allons maintenant refaire les mêmes opérations de fusion que précédemment, pour les faces non coplanaires et les arêtes non alignées, qui sont les seuls éléments restants à simplifier.

Définition 27 (carte de niveau 4) *La* carte de niveau 4 *est la carte obtenue à partir de la carte de niveau* 3*, en fusionnant chaque couple de faces adjacentes et incidentes à une arête de degré un ou deux.*

Nous ne précisons pas dans cette définition que les faces adjacentes et incidentes à une arête de degré un ou deux doivent être non coplanaires, car les faces coplanaires vérifiant ces contraintes ont déjà été fusionnées lors du calcul de la carte de niveau 2.

La carte de niveau 4 de notre image d'exemple est présentée figure 5.10. Nous pouvons observer sur cette figure que le nombre de brins de cette carte est beaucoup moins important que celui du niveau précédent. Cette carte est plus difficile à visualiser étant donné que les faces ne sont plus forcément planaires, contrairement aux niveaux précédents.

Nous rencontrons pour ce niveau exactement les mêmes problèmes que pour la définition de la carte de niveau 2. Nous fusionnons les faces incidentes à des arêtes de degré un ou deux, afin que la définition ne dépende pas de l'ordre dans lequel ces fusions sont effectuées. Le même problème de déconnexion de face peut se produire, mais il sera présenté et résolu dans le chapitre 5.3, en même temps que le problème de déconnexion du niveau 2. En effet ces deux problèmes sont topologi-

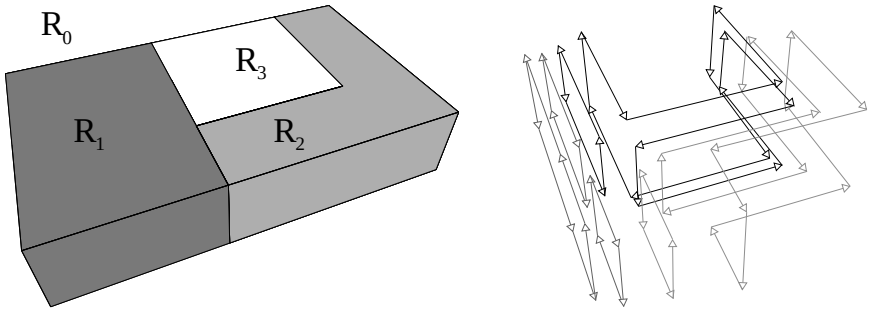


FIG. 5.10 – La carte de niveau 4.

quement équivalents. La seule différence se situe au niveau de la géométrie des faces fusionnées, ce qui ne rentre pas du tout en compte pour ce problème qui est uniquement topologique.

5.2.6 Le niveau 5 : la carte topologique

Il ne reste plus qu'à fusionner les arêtes incidentes à un sommet de degré deux afin d'obtenir la carte de niveau 5 qui est le dernier niveau de simplification. En effet, nous avons effectué toutes les fusions possibles conservant les informations topologiques. Il est donc impossible de simplifier quoi que ce soit dans ce dernier niveau de carte, sans entraîner la perte d'informations. C'est pour cette raison que nous appelons ce niveau 5 la *carte topologique*.

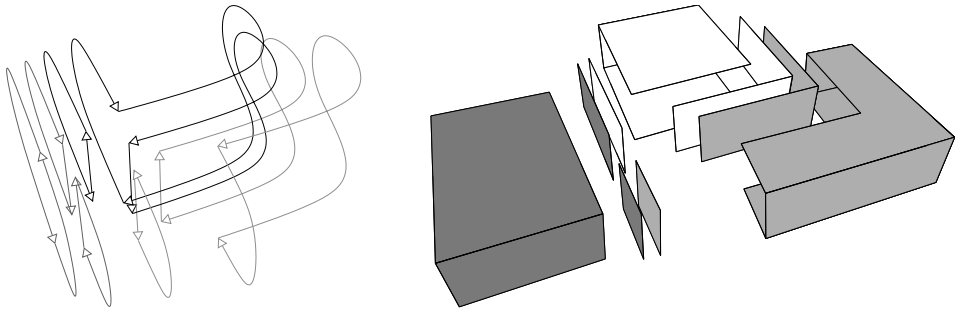
Définition 28 (carte de niveau 5) *La* carte de niveau 5 *est la carte obtenue à partir de la carte de niveau* 4*, en fusionnant chaque couple d'arêtes adjacentes incidentes à un sommet de degré deux.*

Les arêtes fusionnées lors de cette étape sont forcément non alignées, étant donné que celles qui sont alignées ont déjà été fusionnées lors de la construction de la carte des frontières. Cette fusion, comme pour la carte des frontières, ne peut pas entraîner de déconnexion. De plus, nous ne fusionnons pas les arêtes incidentes à un sommet de degré un, pour les raisons déjà présentées lors de la définition de la carte de niveau 3.

La carte de niveau 5 de notre image d'exemple est présentée figure 5.11. Nous avons représenté la carte de niveau 5 figure 5.11.*a*, ainsi que les faces frontières de cette image figure 5.11.*b*. En effet, la carte topologique représente de manière minimale ces faces frontières ainsi que toutes les relations d'incidence et d'adjacence, comme nous pouvons le vérifier sur cette figure. Ces faces frontières sont représentées par leurs bords.

La carte de niveau 5 de notre exemple est composée uniquement de 18 brins, si l'on ne considère pas les 6 brins représentant la région infinie. Ce nombre est beaucoup moins important que, par exemple, les 88 brins de la carte des frontières représentant la même image. Rappelons que les cinq niveaux de simplification représentent exactement les mêmes informations topologiques, propriété démontrée chapitre 6.

Mais cette carte de niveau 5 n'est pas vraiment la carte topologique. En effet, nous n'avons pas résolu les problèmes de déconnexion, et cette carte ne représente donc pas forcément toutes



a. La carte de niveau 5. b. Les frontières intervoxel qu'elle représente.

FIG. 5.11 – La carte de niveau 5, et les faces frontières représentées.

les informations topologiques de l'image. La carte topologique est donc la carte de niveau 5 légèrement modifiée afin de résoudre ces problèmes de déconnexion et conserver effectivement toutes ces informations.

5.3 Problèmes de déconnexion et solutions : arbres d'inclusions et éléments fictifs

Nous avons vu lors des définitions des différents niveaux de simplification que deux problèmes de déconnexion peuvent se produire. Afin que la carte topologique conserve correctement toutes les informations topologiques, nous devons résoudre ces problèmes. Nous présentons les solutions à ces deux problèmes, en montrant dans un premier temps que le problème de déconnexion de volume peut se résoudre sans difficulté par l'ajout d'un arbre d'inclusion de régions, de manière similaire à la solution présentée en dimension 2, mais que le problème de déconnexion de face est plus difficile et nécessite la conservation d'arêtes fictives. La conservation de ces éléments particuliers doit être prise en compte pour la définition de la carte topologique. Nous étudions donc les propriétés de ces arêtes particulières afin de comprendre comment elles interagissent avec la carte topologique, et comment les gérer afin de garantir la minimalité de cette carte.

5.3.1 La déconnexion de volume

Ce type de déconnexion peut survenir lors de la fusion de volumes, lorsqu'un volume est totalement inclus dans un autre. Nous pouvons voir figure 5.12 un exemple pour lequel une région est totalement incluse dans une autre. La figure 5.12.*a* montre une carte de niveau 1 en cours de construction, avant d'avoir effectué la dernière fusion de volumes le long de la face grise. Notons que cette carte est connexe. Après cette fusion de volume, nous obtenons la carte présentée figure 5.12.*b*, dans laquelle la surface extérieure du gros cube est déconnectée de la surface intérieure représentant un trou dans ce cube. L'information permettant de relier ces deux surfaces et de les positionner l'une par rapport à l'autre est perdue. Elle doit être conservée sous peine de ne pas caractériser entièrement les différents objets.

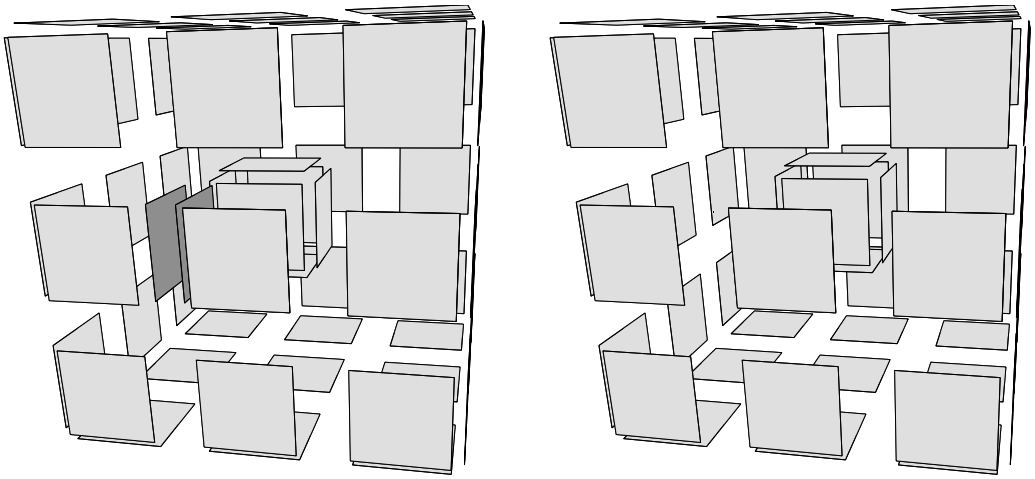


FIG. 5.12 – Un exemple de déconnexion lors de la fusion de volumes.

Ce type de déconnexion est similaire à celui déjà rencontré en dimension 2, où nous avions alors des déconnexions lors de fusion de faces. Étant donné que nous travaillons en dimension supérieure, nous rencontrons maintenant ces déconnexions pour la fusion de volumes. La solution est la même que pour la dimension 2, et consiste à ajouter un *arbre d'inclusion* des régions. Cet arbre possède un nœud par région de l'image. Sa racine est toujours la région infinie, et un nœud N_1 est fils d'un nœud N_2 si et seulement si la région R_1 correspondant à N_1 est directement incluse dans la région R_2 correspondant à N_2.

À l'aide de cet arbre, nous conservons la relation d'inclusion qui permet de positionner les différentes composantes connexes de la carte les unes par rapport aux autres. Remarquons qu'une autre solution consiste à ne pas effectuer la dernière fusion de volumes, et donc à conserver la carte présentée figure 5.12.*a* comme représentation finale. Cette deuxième solution est équivalente à la première au niveau topologique, mais pas au niveau de sa mise en œuvre. En effet, cette solution entraîne la conservation de faces ayant des propriétés particulières, étant donné qu'elles sont incidentes à une seule et même région. Ces faces auraient normalement dû être fusionnées lors de la construction de la carte de niveau 1, et le fait de les conserver va nous obliger à considérer deux cas différents pour chaque opération, suivant que le brin traité appartient à une de ces faces particulières ou non. De plus, ces faces servent uniquement à représenter la relation d'inclusion, et ne portent aucune autre information topologique.

Ces raisons nous font préférer la première solution, qui permet de s'affranchir de ces faces particulières et de simplifier les traitements. De plus, c'est cette solution qui a été retenue en dimension 2, et que nous étendons donc ici. Enfin, l'information principale utilisée par des algorithmes de traitement porte sur les régions de l'image. L'arbre d'inclusion pourra donc être utilisé afin de contenir d'autres informations sur les régions, comme par exemple leurs couleurs, ou leurs nombres de voxels.

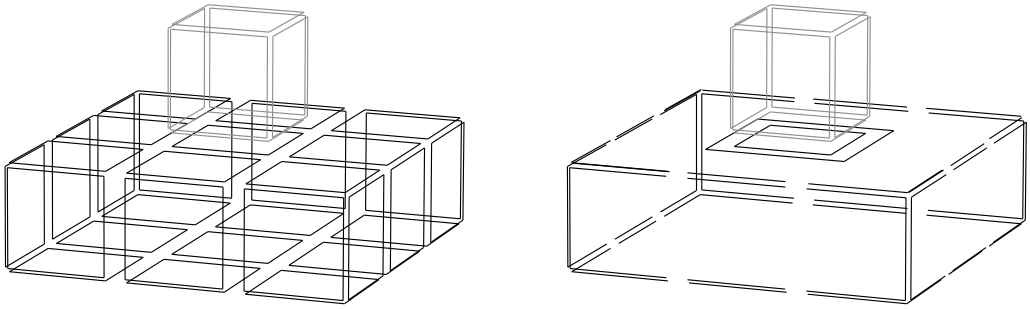


FIG. 5.13 – Un exemple de déconnexion lors de la fusion de faces coplanaires.

5.3.2 La déconnexion de face et les arêtes fictives

La déconnexion de face peut se produire lors des fusions de faces, donc pour la construction des niveaux 2 et 4. En effet, la différence entre ces niveaux se situe uniquement sur l'aspect géométrique. Pour le niveau 2 seules les faces coplanaires sont fusionnées, alors que pour le niveau 4 ce sont les faces non-coplanaires. Mais ce problème de déconnexion est uniquement topologique et ne dépend pas de la géométrie.

Nous pouvons voir figure 5.13 un exemple pour lequel il se produit une déconnexion de face lors de la construction de la carte de niveau 2. Cette figure présente tout d'abord la carte de niveau 1 d'un objet composé d'un pavé sur lequel est posé au centre un cube, puis la carte de niveau 2 de ce même objet. Pour cette deuxième carte, nous avons déconnecté le bord extérieur de la face supérieure du pavé de son bord intérieur. Nous sommes alors incapables de positionner topologiquement ces deux composantes connexes.

Il est possible d'appliquer la même solution que pour le problème de déconnexion de volume, en ajoutant un arbre d'inclusion des faces qui pour chaque face donnerait l'ensemble des faces incluses. Mais cette première solution n'est pas satisfaisante. En effet, contrairement au problème de déconnexion de volume, l'information perdue n'est pas uniquement une information d'inclusion. En effet, par exemple lorsqu'un volume est représenté par une surface fermée, cette surface n'a pas de bord et ne serait alors pas du tout représentée dans la carte de niveau 5. Nous aurions alors perdu le genre de l'objet, malgré la présence des arbres d'inclusion de faces.

Une solution adéquate consiste à conserver une arête permettant de relier le bord extérieur d'une face et son bord intérieur. Nous pouvons voir figure 5.14 la carte de niveau 2 de l'objet déjà présenté figure 5.13, mais pour lequel nous avons conservé une arête reliant le bord extérieur de la face supérieure du pavé avec son bord intérieur. Cette arête permet de conserver la notion de face introduite lors de la présentation des cartes combinatoires, c'est-à-dire au sens orbite $< \beta_1 >$. En effet, si nous partons de n'importe quel brin de la face supérieure du pavé et effectuons un parcours suivant cette orbite, nous parcourons bien l'ensemble des brins de cette face, ce qui n'était pas le cas lorsque ces deux bords étaient déconnectés.

Ces arêtes particulières peuvent être caractérisées sans difficulté, étant donné qu'elles sont de degré un et incidentes deux fois à la même face. Les arêtes de degré un sont normalement supprimées lors de la construction du niveau 2 ou du niveau 4 (suivant la géométrie de la face), mais ces

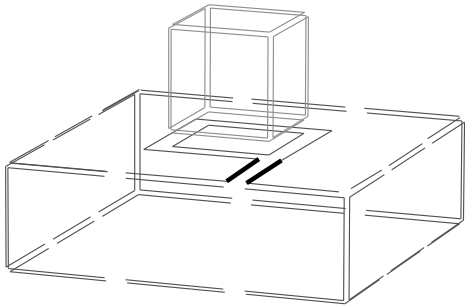


FIG. 5.14 – L'objet présenté figure 5.13 avec une arête fictive.

arêtes particulières ne doivent pas l'être afin de conserver chaque face connexe. De plus, ces arêtes n'ont pas de plongement, étant donné qu'elles ne représentent pas le bord d'une surface frontière de l'image. Elles ont une existence uniquement topologique. C'est pour ces raisons que nous les appelons des *arêtes fictives*. Par opposition, nous appelons les autres arêtes des *arêtes réelles*. Étant donné que ces arêtes fictives n'ont qu'une existence topologique, elles ne sont pas figées et peuvent être déplacées à l'intérieur de la face, à condition toutefois de la conserver connexe. Nous verrons section 5.3.3 que cette propriété est très importante, et que son utilisation garantit que la carte topologique est bien minimale.

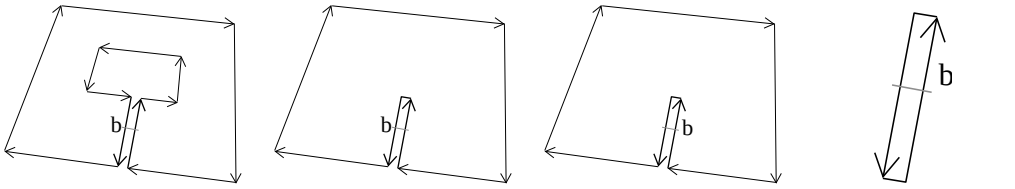
Afin de conserver ces arêtes fictives, nous modifions les définitions 25 et 27 des niveaux 2 et 4. En effet, c'est uniquement pour la construction des cartes de ces deux niveaux que le problème de déconnexion de face peut se produire. C'est donc uniquement pour ces niveaux qu'il s'agit éventuellement de conserver des arêtes fictives.

Définition 29 (carte de niveau 2) *La* carte de niveau 2 *est la carte obtenue à partir de la carte de niveau* 1*, en fusionnant chaque couple de faces adjacentes, coplanaires et incidentes à une arête de degré un ou deux, n'entraînant pas de déconnexion de face.*

Définition 30 (carte de niveau 4) *La* carte de niveau 4 *est la carte obtenue à partir de la carte de niveau* 3*, en fusionnant chaque couple de faces adjacentes et incidentes à une arête de degré un ou deux, n'entraînant pas de déconnexion de face, ni la suppression totale de la face.*

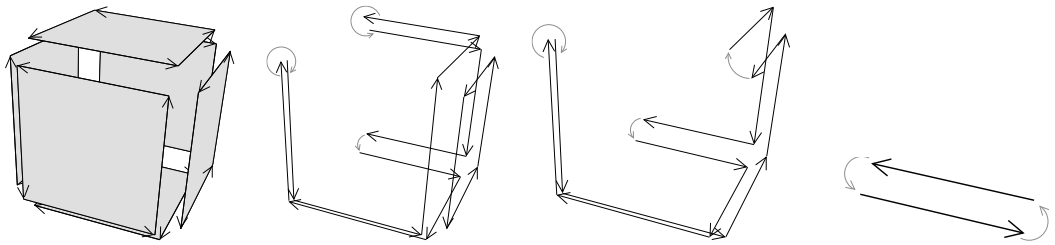
Nous pouvons voir sur les définitions 29 et 30 que nous avons simplement ajouté une condition supplémentaire afin de ne pas effectuer une fusion de faces lorsqu'elle entraîne une déconnexion. Remarquons qu'avec ces définitions modifiées, nous obtenons bien pour notre exemple de la figure 5.13, la carte de niveau 2 de la figure 5.14. Nous pouvons noter sur cette carte la présence d'une arête fictive qui permet de conserver la connexité de la face supérieure du pavé. Avec ces nouvelles définitions, chaque face de ces cartes est homéomorphe à un disque topologique.

Pour tester si une fusion de faces entraîne une déconnexion ou non, il suffit d'effectuer un parcours de l'orbite $< \beta_1 >$ d'origine un brin b de l'arête le long de laquelle doit être effectuée la fusion. Si durant ce parcours, nous atteignons le brin $\beta_2(b)$, alors l'arête est de degré un car les deux brins b et $\beta_2(b)$ appartiennent à la même face. Dans ce cas la fusion le long de cette arête va entraîner une déconnexion si les deux brins b et $\beta_2(b)$ ne sont pas cousus entre eux par β_0 et



a. $\beta_0(b) \neq \beta_2(b)$ $\beta_1(b) \neq \beta_2(b)$. b. $\beta_0(b) = \beta_2(b)$ $\beta_1(b) \neq \beta_2(b)$. c. $\beta_1(b) = \beta_2(b)$ $\beta_0(b) \neq \beta_2(b)$. d. $\beta_0(b) = \beta_2(b)$ $\beta_1(b) = \beta_2(b)$.

FIG. 5.15 – Les quatre configurations possibles de faces autour d'une arête de degré un.



a. Le niveau 3. b. Deux étapes intermédiaires. c. Les niveaux 4 et 5.

FIG. 5.16 – Le cas de la sphère pour les niveaux 3, 4 et 5.

par β_1. Nous pouvons voir figure 5.15 les quatre configurations possibles de faces autour d'une arête de degré un, où nous pouvons vérifier que le seul cas pouvant entraîner déconnexion est celui présenté figure 5.15.*a*.

Le dernier cas de cette figure est un autre cas particulier, qui bien que n'entraînant pas de déconnexion nécessite une condition supplémentaire. Nous ne devons pas effectuer la fusion de deux faces si cela entraîne la suppression totale d'une face. Cette condition est ajoutée dans la définition 30 de la carte de niveau 4. Ce cas ne peut pas se produire pour le niveau 2, étant donné que chaque surface représente le bord d'un volume, et que ce bord ne peut jamais être constitué uniquement d'une face plane fermée. Il se produit lorsque les deux brins b et $\beta_2(b)$ formant l'arête le long de laquelle doit être effectuée la fusion de faces, sont cousus entre eux par β_1. Ce cas survient lorsque l'objet représenté est une sphère totalement incluse dans une autre région.

Nous présentons figure 5.16.*a* la carte de niveau 3 d'une sphère topologique (ici un cube) entièrement incluse dans un autre volume. Nous pouvons remarquer que chaque arête de cette carte est de degré deux. Lors de la construction de la carte de niveau 4 à partir de cette carte, nous pouvons effectuer les fusions de faces le long de toutes ces arêtes sans entraîner de déconnexion de face.

Nous pouvons voir figure 5.16.*b* deux étapes intermédiaires de cette construction, où nous avons seulement effectué certaines fusions de faces. Sur cette figure, nous avons représenté explicitement β_1 par des arcs de cercle gris lorsque les brins concernés ne sont pas dessinés de manière consécutive. Nous voyons sur ces étapes intermédiaires que les fusions de face entraînent progressivement la suppression de faces. Nous pouvons également remarquer que les différentes fusions

ne peuvent pas être effectuées dans n'importe quel ordre. En effet, une fusion peut entraîner une déconnexion si elle est effectuée avant une autre, alors que ce ne serait pas le cas si ces fusions étaient effectuées dans l'ordre inverse. Mais il existe un ordre de fusion qui n'entraîne aucune déconnexion, et nous pouvons donc effectuer la fusion de faces le long de chaque arête de la carte. Cela peut se vérifier sur la figure 5.16, où intuitivement nous commençons par fusionner les faces le long des arêtes aux extrémités de la carte.

Dans ce cas, nous supprimons tous les brins représentant cette face ainsi que, du même coup, le volume complet. C'est pour cette raison que nous avons ajouté dans la définition 30, la contrainte selon laquelle nous n'effectuons pas une fusion de faces le long d'une arête si cette fusion entraîne la suppression totale de la face. Avec cette contrainte supplémentaire, nous obtenons alors la carte de niveau 4 présentée figure 5.16.*c*. Cette carte est également la carte de niveau 5, étant donné que les deux sommets de cette carte sont de degré un, et ne vont donc pas être supprimés par une fusion d'arêtes. Nous obtenons bien une représentation minimale de la sphère. En effet, cette carte possède une face, une arête et deux sommets, et la formule d'Euler nous permet de vérifier que le genre de cet objet est égal à zéro.

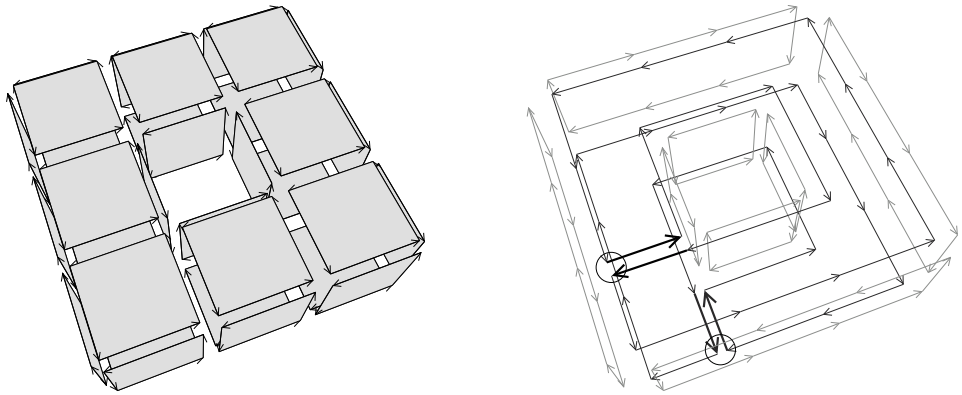
Cette représentation minimale n'est pas unique, car avec le même nombre de brins, nous pouvons représenter une sphère par deux faces composées d'un brin chacune et β_2 cousus entre eux. Cette autre représentation possède donc deux faces, une arête et un sommet, et donc est également de genre zéro. Ces deux représentations minimales sont duales l'une de l'autre. Mais nous préférons la première représentation, tout d'abord car c'est celle que nous obtenons à partir de notre définition, mais également car la deuxième représentation possède deux faces différentes. Dans ce cas, nous ne représentons plus les faces frontières du volume correspondant. En effet, lorsqu'un objet est totalement inclus dans un autre, il y a une seule face frontière entre ces deux objets. Cette information n'apparaît pas immédiatement avec la deuxième représentation minimale, alors que la première représentation possède bien une seule face topologique correspondant à cette face frontière.

Avec ces définitions modifiées des niveaux 2 et 4, nous représentons maintenant correctement toutes les configurations possibles, en conservant chaque face homéomorphe à un disque topologique au moyen d'arêtes fictives lorsque c'est nécessaire. Mais la carte topologique doit être la représentation minimale de l'image. Nous étudions maintenant comment gérer ces arêtes fictives afin de garantir cette minimalité.

5.3.3 Gestion des arêtes fictives et représentation minimale

Le fait de conserver des arêtes fictives pour les cartes de niveau 2 et 4 permet de résoudre le problème de déconnexion de face. Mais ces arêtes introduisent de nouveaux problèmes lors de la construction des cartes de niveau 3 et 5, c'est-à-dire lors de fusions d'arêtes. Leur résolution garantit l'obtention de la représentation minimale.

Un premier problème se pose lors de la construction des cartes de niveau 3 et 5, lorsque nous sommes en présence d'arêtes fictives. En effet, ces arêtes fictives peuvent, suivant leur position dans la carte, empêcher la fusion de deux arêtes qui l'auraient été autrement. Ce n'est pas du tout satisfaisant car le rôle des arêtes fictives est uniquement de conserver les faces connexes. Elles ne doivent pas intervenir dans la construction des autres niveaux.



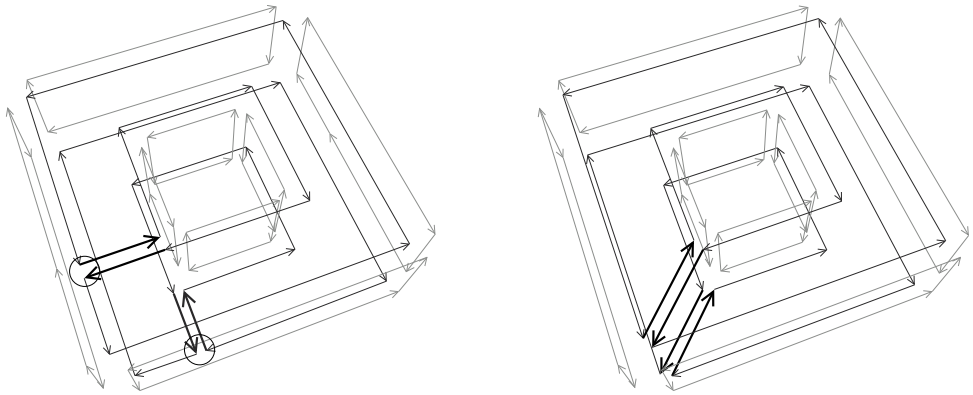
a. La carte de niveau 1. b. La carte de niveau 2.

FIG. 5.17 – Les cartes de niveau 1 et 2 d'un tore totalement inclus dans une autre région.

La figure 5.17 permet de visualiser ce problème. La figure 5.17.*a* montre la carte de niveau 1 d'un tore totalement inclus dans une autre région. Cette carte représente donc une seule face frontière qui est fermée. La figure 5.17.*b* montre la carte de niveau 2 de ce même tore. Nous pouvons noter la présence de deux arêtes fictives (représentées en noir épais sur le dessin) qui permettent de conserver les faces supérieure et inférieure du tore connexes. Nous avons sur cette figure représenté en foncé les brins appartenant à ces deux faces, étant donné que ce sont elles qui nous intéressent, et en plus clair les autres brins.

Nous pouvons voir que les arêtes fictives sont positionnées, par rapport aux autres brins de la face, de manière totalement arbitraire étant donné que leurs positions dépend de l'ordre des fusions de faces et que cet ordre est lui même totalement arbitraire. Cela ne pose aucun problème pour cette carte de niveau 2, mais en pose lors du calcul du niveau suivant. En effet, lors de la construction de la carte de niveau 3, nous fusionnons chaque couple d'arêtes adjacentes, alignées, et incidentes à un sommet de degré deux. Or les deux sommets de la carte de niveau 2 (figure 5.17.*b*) entourés de cercles noirs ne sont pas de degré deux à cause de la présence des arêtes fictives. Si nous n'effectuons pas de distinction entre les arêtes fictives et les autres arêtes lors de la construction du niveau 3, nous obtenons la carte présentée figure 5.18.*a*.

Sur cette carte, nous voyons que ces deux sommets n'ont pas été supprimés par une fusion d'arêtes, alors qu'ils l'auraient été sans la présence des arêtes fictives, ou si elles avaient été positionnées de manière différente. La figure 5.18.*b* présente la carte de niveau 3 que nous souhaitions obtenir. En effet, les arêtes fictives servent uniquement à conserver chaque face connexe, mais ne doivent pas empêcher la fusion de certaines arêtes. Nous devons régler ce problème afin de ne pas avoir la carte de niveau 3 qui dépend de l'ordre dans lequel les fusions de faces coplanaires ont été effectuées. En effet, le cas échéant nous n'aurions pas une représentation unique de cette carte, et donc de la carte topologique. De plus, la représentation obtenue ne serait alors pas minimale, étant donné que certaines arêtes ne seraient pas fusionnées. Pour cela, nous considérons de manière différente les arêtes fictives des autres arêtes lors des calculs des niveaux 3 et 5, c'est-à-dire pour les fusions d'arêtes. Nous modifions donc les définitions 26 et 28 en conséquence.



a. Problème pour le niveau 3. b. La carte de niveau 3 correcte.

FIG. 5.18 – Le problème rencontré lors du calcul de la carte de niveau 3 du tore si l'on ne considère pas différemment les arêtes fictives, et la carte de niveau 3 correcte.

Définition 31 (carte de niveau 3) *La* carte de niveau 3 *est la carte obtenue à partir de la carte de niveau* 2*, en fusionnant chaque couple d'arêtes* (a_1, a_2) *adjacentes, alignées et incidentes à un sommet* s *tel qu'il n'existe pas d'arête réelle différente de* a_1 *et de* a_2 *incidente à* s*.*

Définition 32 (carte de niveau 5) *La* carte de niveau 5 *est la carte obtenue à partir de la carte de niveau* 4*, en fusionnant chaque couple d'arêtes* (a_1, a_2) *adjacentes, n'étant pas des boucles et incidentes à un sommet* s *tel qu'il n'existe pas d'arête réelle différente de* a_1 *et de* a_2 *incidente à* s*.*

Par rapport aux anciennes définitions de ces deux niveaux, nous avons remplacé la condition que le sommet soit de degré deux par la nouvelle condition qu'il n'existe pas d'arête réelle différente de a_1 et de a_2 incidente à ce sommet. Remarquons que ces deux conditions sont équivalentes lorsqu'il n'y a pas d'arête fictive. Mais cette nouvelle condition permet de ne pas dépendre de la position des arêtes fictives qui ne vont plus entrer en compte lors des fusions d'arêtes.

Pour la définition du niveau 5, une contrainte supplémentaire impose que a_1 et a_2 ne soient pas des boucles. En effet, il est impossible de fusionner une boucle avec une autre arête. La fusion d'arête peut être considérée comme un étirement de la première arête afin qu'elle « couvre » la seconde, ce qui n'est pas possible à réaliser lorsqu'une des deux arêtes est une boucle. Cette condition est inutile pour le niveau 3 étant donné que nous ne pouvons pas avoir de boucle, puisque nous fusionnons uniquement des arêtes alignées.

Avec ces définitions modifiées, nous pouvons donc fusionner deux arêtes réelles le long du sommet s, malgré la présence d'arêtes fictives incidentes à s. Mais dans ce cas, afin de réaliser cette fusion, nous devons effectuer préalablement un traitement particulier. En effet, une fusion d'arête peut être réalisée uniquement le long d'un sommet de degré deux. Pour cela, avant d'effectuer la fusion de deux arêtes a_1 et a_2, nous déplaçons simplement toutes les arêtes incidentes à s, à l'exception de a_1 et a_2, sur le deuxième sommet de a_1.

En effet, ces arêtes sont forcément des arêtes fictives, et nous avons vu qu'elles ont uniquement un sens topologique, et peuvent donc être déplacées, à condition de conserver la carte connexe. De plus, nous savons que a_1 et a_2 ne sont pas des boucles et donc il existe bien un autre sommet différent de s incident à a_1. Remarquons enfin que nous pourrions choisir de décaler les arêtes fictives sur le deuxième sommet de a_2 sans que cela ne change quoi que ce soit à la suite des traitements.

Nous présentons l'algorithme 17 effectuant le décalage de toutes les éventuelles arêtes fictives incidentes à un sommet donné. Cet algorithme est composé de deux boucles très similaires.

Algorithme 17 Décalage de toutes les arêtes fictives incidentes à un sommet donné

Entrée : Deux brins b_1 et b_2 incident au même sommet

Résultat : Les arêtes fictives (différentes des arêtes incidentes à b_1 et b_2), incidente au même sommet sont décalées sur le sommet incident à $\beta_1(b_1)$.

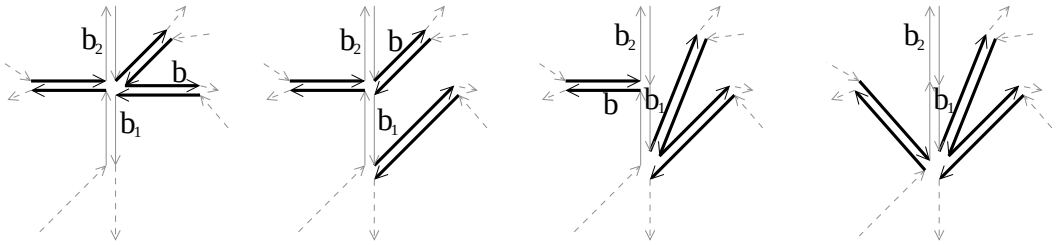
$b \leftarrow \beta_{02}(b_1)$;
tant que $b \neq b_2$ **faire**
  $b_{next} \leftarrow \beta_{02}(b)$;
  Décaler_arête$(b, \beta_1(b_1))$;
  $b \leftarrow b_{next}$;
$b \leftarrow \beta_{21}(b_1)$;
tant que $b \neq b_2$ **faire**
  $b_{next} \leftarrow \beta_{21}(b)$;
  Décaler_arête$(b, \beta_2(b_1))$;
  $b \leftarrow b_{next}$;

La première traite les éventuelles arêtes fictives se trouvant entre b_1 et b_2, en tournant dans le sens trigonométrique. Pour chacune de ces arêtes, elle appelle simplement l'algorithme 18 qui se charge de décaler cette arête fictive entre les brins b_1 et $\beta_1(b_1)$. La deuxième boucle effectue un traitement similaire pour chaque arête fictive se trouvant entre b_1 et b_2, en tournant dans le sens trigonométrique inverse. Ces arêtes fictives sont quant à elles décalées entre les brins $\beta_2(b_1)$ et $\beta_{20}(b_1)$.

Nous devons effectuer ces deux traitements de manière différente car suivant la position originale des arêtes fictives, elles ne doivent pas être décalées au même endroit. En effet, lorsque f_1 la face incidente à b_1 n'est pas incidente à b_2, les arêtes fictives conservant la face f_1 connexe doivent être décalées à l'intérieur de la face f_1. Par contre, les arêtes fictives conservant f_2 la face incidente à b_2 connexe doivent être décalées à l'intérieur de f_2.

Nous présentons figure 5.19 le déroulement de cet algorithme. La figure 5.19.*a* montre une configuration où deux arêtes réelles (incidentes à b_1 et b_2) sont incidentes à un sommet tel qu'il n'existe pas d'autre arête réelle incidente à ce sommet. Les flèches en pointillés gris symbolisent la présence de brins n'étant pas concernés par cet algorithme de décalage, mais rappellent leur existence.

La première boucle commence à traiter la première arête fictive après b_1 en tournant dans le sens trigonométrique. La figure 5.19.*b* montre l'état de la carte après le premier décalage de cette arête, et la figure 5.19.*c* son état après le deuxième décalage. Après ce décalage, nous avons $b = b_2$, ce qui entraîne la sortie de la première boucle. La deuxième boucle effectue pour cet exemple un



a. Initialisation. b. Après un décalage. c. Après deux décalages : fin de la première boucle. d. Fin de l'algorithme.

FIG. 5.19 – Déroulement de l'algorithme de décalage des arêtes fictives.

seul décalage et nous obtenons au final la carte présentée figure 5.19.*d*. Nous avons bien décalé toutes les arêtes fictives incidentes au sommet incident à b_1 et à b_2 : ce sommet est désormais de degré deux.

L'algorithme « Décaler_arête », présenté algorithme 18, effectue simplement le décalage de l'arête fictive incidente à b_{fictif} entre les brins $\beta_0(b_{dest})$ et b_{dest}.

Algorithme 18 Décalage d'une arête fictive donnée

Entrée : Deux brins b_{fictif} et b_{dest}

Résultat : L'arête fictive incidente à b_{fictif} est décalé entre les brins $\beta_0(b_{dest})$ et b_{dest}.

$b_1 \leftarrow \beta_0(b_{fictif})$; $b_2 \leftarrow \beta_{21}(b_{fictif})$; $b_3 \leftarrow \beta_0(b_{dest})$;

1 β_0-découdre(b_{fictif}) ; β_1-découdre$(\beta_2(b_{fictif}))$;
β_1-découdre$(\beta_3(b_{fictif}))$; β_0-découdre$(\beta_{23}(b_{fictif}))$;

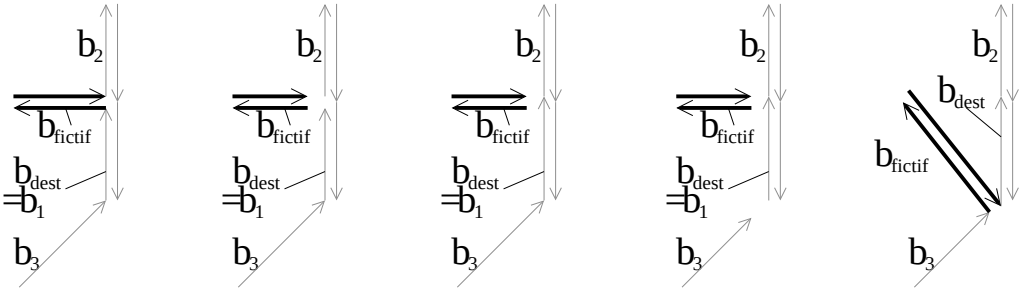
2 β_1-coudre(b_1, b_2) ; β_0-coudre$(\beta_3(b_1), \beta_3(b_2))$;

3 β_1-découdre(b_3) ; β_0-découdre$(\beta_3(b_3))$;

4 β_1-coudre(b_3, b_{fictif}) ; β_1-coudre$(\beta_2(b_{fictif}), b_{dest})$;
β_0-coudre$(\beta_3(b_3), \beta_3(b_{fictif}))$; β_0-coudre$(\beta_{23}(b_{fictif}), \beta_3(b_{dest}))$;

Ce décalage s'effectue en 4 étapes que nous illustrons sur l'exemple présenté figure 5.20. Après la phase d'initialisation (figure 5.20.*a*), la première étape de l'algorithme effectue la décousure de l'arête fictive incidente à b_{fictif} (figure 5.20.*b*). Une arête fictive est forcément composée de 4 brins. En effet, il existe deux brins cousus entre eux par β_2, par demi-face, et une face topologique est composée de deux demi-faces identiques β_3-cousues. Nous ne représentons pas la deuxième demi-face sur nos figures pour des raisons de lisibilité.

La deuxième étape de l'algorithme β_1-coud b_1 et b_2, les deux brins qui ont été décousus lors de la première phase (figure 5.20.*c*). Afin de vérifier les contraintes des cartes combinatoires, nous effectuons la couture similaire pour les brins correspondant dans la deuxième demi-face. La troisième étape β_1-découd b_3 pour insérer l'arête fictive (figure 5.20.*d*). De même que précédemment, nous effectuons la décousure similaire pour $\beta_3(b_3)$. Enfin, la dernière étape (figure 5.20.*e*) insère l'arête fictive entre les brins b_3 et b_{dest} (ainsi qu'entre $\beta_3(b_{dest})$ et b_3 pour l'autre demi-face).



a. Initialisation. b. Décousure de l'arête incidente à b_{fictif}. c. β_1-couture de b_1 et b_2. d. β_1-décousure de b_3. e. Couture de l'arête incidente à b_{fictif}.

FIG. 5.20 – Déroulement de l'algorithme décalant une seule arête fictive.

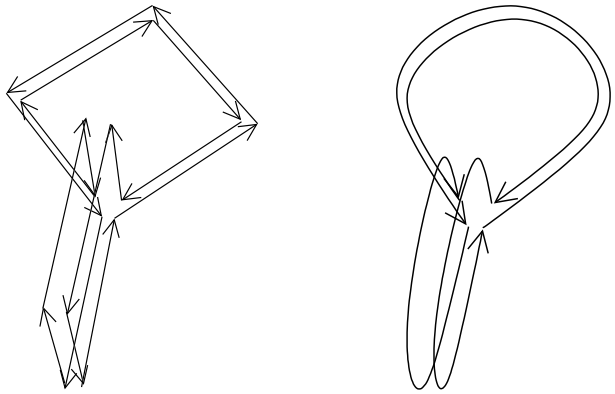
La complexité de cet algorithme est en temps constant, étant donné que nous effectuons uniquement des opérations unitaires de couture et de décousure. Remarquons que cet algorithme fonctionne quelle que soit la configuration de la carte de départ, la seule contrainte étant que l'arête incidente à b_{fictif} doit être une arête fictive. La complexité de l'algorithme 17 effectuant le décalage de l'ensemble des arêtes fictives incidentes à un sommet est en *O(nombre d'arêtes fictives décalées)*. En effet, les deux boucles « tant que » parcourent l'ensemble des arêtes fictives une à une, et effectuent uniquement des opérations en temps constant. Remarquons enfin que cet algorithme fonctionne même s'il n'y a aucune arête fictive à décaler, et dans ce cas n'entraîne pas de modification de la carte.

Lors des calculs des cartes de niveaux 3 et 5, nous testons si deux arêtes a_1 et a_2 doivent être fusionnées, à l'aide des contraintes présentées définitions 31 et 32. Si c'est le cas, nous appelons l'algorithme 17, qui va décaler les éventuelles arêtes fictives incidentes à ce sommet. Enfin, nous effectuons la fusion des deux arêtes a_1 et a_2 restantes de manière classique, étant donné que le sommet est désormais de degré 2.

Lorsque nous appliquons ce principe à la carte de niveau 2 pour notre exemple du tore (cf. figure 5.17.*b*), nous obtenons bien la carte de niveau 3 correcte présentée figure 5.18.*b*, où les arêtes fictives n'ont pas empêché de fusion d'arêtes. La position des arêtes fictives dépend de l'ordre dans lequel nous avons effectué les fusions d'arêtes et dans lequel nous avons choisi les brins de ces arêtes. Nous obtenons donc une représentation qui est unique à la position des arêtes fictives près.

Afin de calculer la carte de niveau 4, nous fusionnons maintenant chaque couple de faces adjacentes incidentes à une arête de degré un ou deux n'entraînant pas de déconnexion de face, ni la suppression totale de la face (d'après la définition 30). Nous observons sur la carte de niveau 3 du tore (cf. figure 5.18.*b*) que toutes ses arêtes sont de degré deux. Seules les conditions supplémentaires sur la déconnexion et la suppression totale vont nous contraindre à ne pas effectuer certaines fusions.

La carte de niveau 4 de cet exemple est présentée figure 5.21.*a*. Cette carte est composée d'une



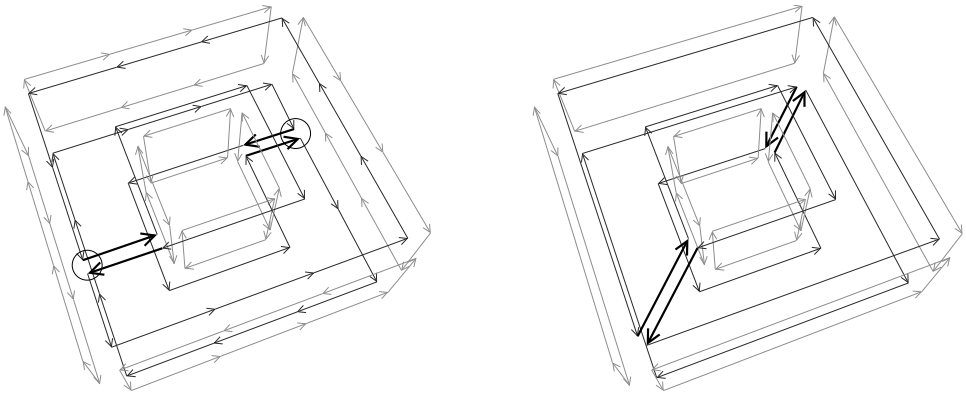
a. La carte de niveau 4. b. La carte de niveau 5.

FIG. 5.21 – Suite de l'exemple du tore pour les niveaux 4 et 5.

seule face. Il est impossible d'effectuer une fusion de faces le long de n'importe quelle arête de cette carte sans entraîner une déconnexion. Cette carte ne peut donc plus être simplifiée par des fusions de faces. De ce fait, chaque arête de cette carte est une arête fictive. En effet, toute ces arêtes sont de degré un et n'ont pas été fusionnées uniquement car cela entraînait une déconnexion. Mais c'est bien ce que nous voulions obtenir, étant donné que la face frontière que représente cette carte est fermée. Elle n'a donc pas de bord, et par conséquence pas d'arête réelle.

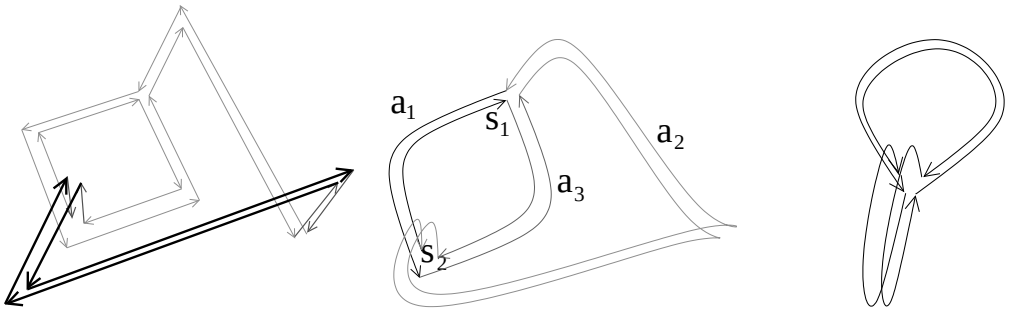
Nous pouvons maintenant construire la carte de niveau 5 à partir de cette carte, en appliquant la définition 32. Remarquons que, lors de cette construction, nous fusionnons des arêtes fictives. En effet, les nouvelles définitions des niveaux 3 et 5 imposent qu'il n'existe pas d'arête réelle incidente au même sommet différente des deux arêtes à fusionner, mais il n'y a pas de contrainte sur ces deux arêtes. Elles peuvent donc être indifféremment deux arêtes réelles ou deux arêtes fictives. Mais il n'est pas possible de fusionner une arête réelle avec une arête fictive. En effet, s'il n'existe pas d'autre arête réelle incidente au même sommet, c'est que l'arête réelle est une boucle et dans ce cas nous n'effectuons pas la fusion. Pour notre exemple du tore, nous obtenons finalement la carte présentée figure 5.21.*b*. En effet, tous les sommets de la carte de niveau 4 sont de degré deux, à l'exception du sommet central où se croisent les deux boucles. Il ne reste donc plus que ce sommet dans la carte de niveau 5, et nous obtenons bien une représentation minimale classique du tore, composée d'une face, de deux arêtes et d'un sommet.

Nous pouvons vérifier sur cet exemple que les arêtes fictives portent plus d'information que celle d'inclusion, et donc que la solution consistant à ajouter un arbre d'inclusion à chaque face n'est pas suffisante. En effet, ces arêtes fictives sont utiles afin de palier le problème de déconnexion de face, mais permettent également la représentation d'objets ayant une surface fermée comme surface frontière, comme nous venons de le voir. Avec ces nouvelles définitions des niveaux de simplification, nous avons obtenu, dans l'exemple que nous venons de traiter, la représentation minimale de manière directe. En effet, la carte de niveau 5 a été obtenue directement à partir de la carte de niveau 4 en fusionnant les arêtes incidentes à des sommets de degré deux. Cette représentation minimale est unique étant donné que le nombre de faces est fixée : il est égal au nombre de faces frontières de l'image.



a. La carte de niveau 2. b. La carte de niveau 3.

FIG. 5.22 – Les cartes de niveau 2 et 3 du tore de l'exemple précédent avec d'autres positions des arêtes fictives.



a. La carte de niveau 4. b. Le niveau 5 en cours de construction. c. La carte de niveau 5.

FIG. 5.23 – Les cartes de niveau 4 et 5, et une étape intermédiaire de construction.

Mais ce n'est pas forcément toujours le cas. Pour illustrer cela, nous reprenons le cas du tore, mais avec d'autres positions des arêtes fictives dans la carte de niveau 2. En effet, ces positions dépendent de l'ordre dans lequel sont réalisées les fusions de faces, étant donné que l'arête fictive s'obtient en n'effectuant pas la dernière fusion. Nous pouvons voir figure 5.22.*a* la carte de niveau 2 du tore de l'exemple précédent, mais avec d'autres positions des arêtes fictives, et figure 5.22.*a* la carte de niveau 3 correspondante. Pour ces deux niveaux, les nouvelles définitions donnent bien les résultats corrects. De plus, ces deux niveaux sont identiques au deux déjà présentés, à la position des arêtes fictives près.

Lorsque nous calculons la carte de niveau 4, nous obtenons alors la carte présentée figure 5.23.*a*. Cette carte est composée d'une seule face, de dix arêtes et de neuf sommets. La caractéristique d'Euler nous permet de vérifier qu'elle représente bien un tore. Pour vérifier que cette carte possède une seule face, il suffit de partir d'un brin quelconque et de parcourir l'orbite $< \beta_1 >$. Nous parcourons alors l'ensemble des brins de cette carte. Cette carte est bien la carte

de niveau 4, car nous ne pouvons pas effectuer d'autre fusion de faces le long de n'importe quelle arête sans entraîner une déconnexion. Remarquons que cette carte n'est pas identique à celle que nous avons obtenu précédemment (figure 5.21.*a*). En effet, ce niveau de carte est dépendant de la géométrie, car nous avons, pour le moment, fusionné uniquement les arêtes alignées.

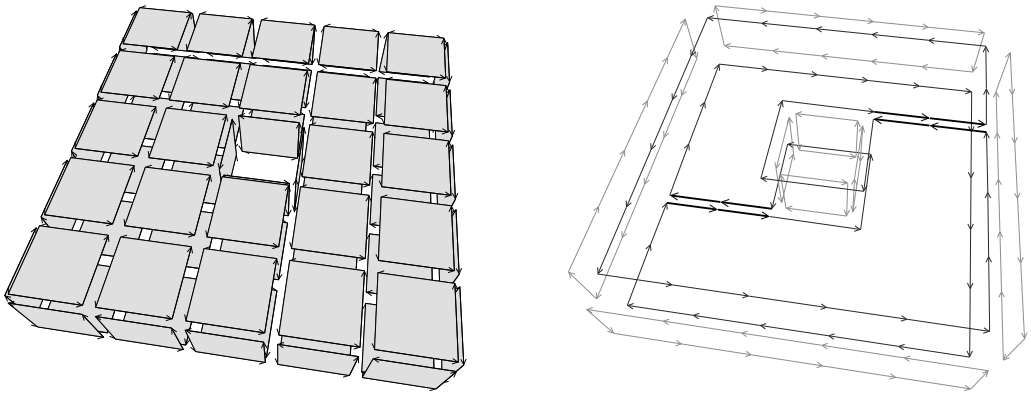
Nous calculons la carte de niveau 5 à partir de cette carte. La figure 5.23.*b* montre la carte obtenue en fusionnant uniquement les arêtes incidentes à des sommets de degré deux. Cette carte n'est pas une représentation minimale d'un tore, et n'est pas celle que nous voulons obtenir. Mais avec notre définition modifiée de la carte de niveau 5, nous fusionnons chaque couple d'arêtes n'étant pas des boucles tel qu'il n'existe pas d'autre arête réelle incidente au même sommet. Nous voyons sur la figure 5.23.*b* que certains couples d'arêtes répondent à ce critère et nous allons les fusionner. Il y a exactement trois couples d'arêtes possibles : (a_1, a_2), (a_1, a_3) et (a_2, a_3). Comme nous ne contrôlons pas l'ordre dans lequel sont effectuées les fusions, c'est une de ces trois fusions qui va être réalisée. Mais quel que soit le couple d'arêtes choisi, nous obtenons le même résultat qui est la carte de niveau 5 présenté figure 5.23.*c*. Remarquons que cette carte est la même que celle obtenue avec l'exemple précédent (cf. figure 5.21.*b*). Nous avons bien une représentation unique et minimale.

Afin d'obtenir cette carte, nous utilisons le principe déjà présenté, consistant à déplacer les éventuelles arêtes fictives incidentes au même sommet avant d'effectuer la fusion. Par exemple, si nous fusionnons les deux arêtes a_1 et a_2 le long du sommet s_1, nous commençons par décaler l'arête a_3 incidente également à ce sommet, sur s_2 qui est le deuxième sommet incident à a_1. Ce décalage rend l'arête a_3 deux fois incidente au même sommet : c'est une boucle. Ensuite, nous effectuons la fusion des deux arêtes a_1 et a_2, étant donné que le sommet s_1 est de degré deux. Cette fusion entraîne la création de la deuxième boucle, et nous obtenons alors la carte de niveau 5 finale.

Ces deux exemples mettent en évidence que les définitions modifiées des niveaux de simplification et notre principe de décalage d'arêtes fictives permettent bien d'obtenir la carte topologique, quelle que soit la position des arêtes fictives. Mais il reste encore un cas particulier à traiter afin d'avoir totalement étudié la gestion des arêtes fictives. Ce cas se produit lors de la fusion de deux arêtes fictives lors du calcul de la carte de niveau 3.

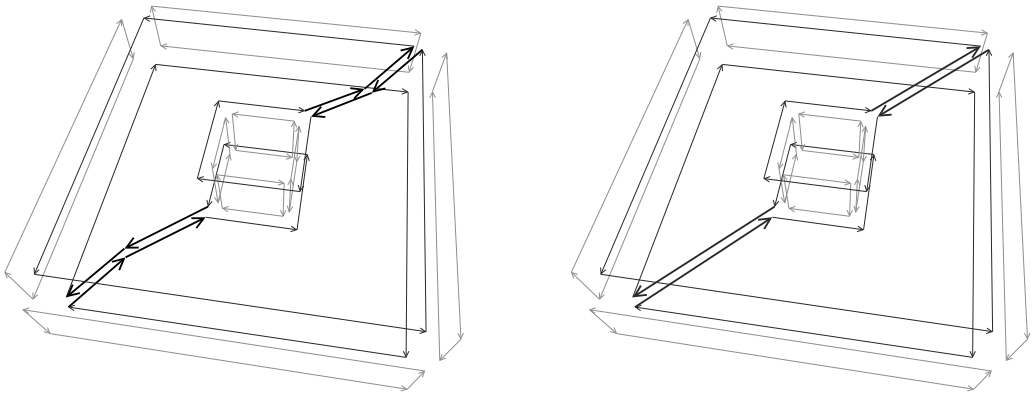
Il est illustré figure 5.24, à nouveau pour un tore, mais contrairement aux deux exemples précédents, celui-ci est de taille $5 \times 5 \times 1$ voxels. Nous pouvons remarquer sur la carte de niveau 2 représentant ce tore (figure 5.24.*b*), la présence d'arêtes fictives permettant de conserver les faces inférieure et supérieure de ce tore connexes. Mais contrairement aux exemples précédents, il y a maintenant deux arêtes fictives pour chacune de ces faces, étant donné que les bords extérieur et intérieur de ces faces sont éloignés de deux lignels. Ce n'est pas gênant pour cette carte de niveau 2, étant donné que chaque arête de cette carte représente un lignel. Mais cela pose problème lors du calcul de la carte de niveau 3.

En effet, d'après la définition 31, nous fusionnons chaque couple d'arêtes adjacentes et alignées tel qu'il n'existe pas d'autre arête réelle incidente au même sommet. Si nous appliquons cette définition telle quelle, ainsi que notre principe de décalage d'arêtes fictives, nous obtenons alors la carte présentée figure 5.25.*a*, sur laquelle nous voyons qu'il y a toujours deux arêtes fictives pour les faces supérieure et inférieure du tore. En effet seule les arêtes alignées ont été fusionnées, or les arêtes fictives non pas de plongement, et n'ont donc pas été fusionnées.



a. La carte de niveau 1. b. La carte de niveau 2.

FIG. 5.24 – Les cartes de niveau 1 et 2 d'un autre tore totalement inclus dans une région.



a. La carte de niveau 3 posant problème. b. La carte de niveau 3 correcte.

FIG. 5.25 – Carte de niveau 3 d'un tore totalement inclus dans une autre région : le problème et la carte correcte.

Il est clair que ce n'est pas la carte que nous souhaitons obtenir. En effet, les arêtes fictives permettent de conserver chaque face connexe. Le fait d'en conserver plusieurs n'apporte aucune information supplémentaire. Il n'y a donc aucun intérêt à conserver ici deux arêtes au lieu d'une. De plus, pour cet exemple nous avons seulement deux arêtes par face, mais suivant l'ordre des fusions de faces, nous pourrions en avoir beaucoup plus, comme pour l'exemple présenté figure 5.26, où nous avons douze arêtes fictives pour la face inférieure du tore, plus deux pour la face supérieure.

Afin de régler ce problème et obtenir la carte de niveau 3 présenté figure 5.25.*b*, nous considérons simplement que deux arêtes fictives adjacentes sont alignées. Cela peut être envisagé car ces arêtes n'ont pas de plongement. Cette simple modification permet de résoudre ce dernier problème. Nous obtenons alors la carte de niveau 3 ayant une seule arête fictive par face trouée, y compris dans le dernier cas. À partir de cette carte, nous continuons le calcul des différents ni-

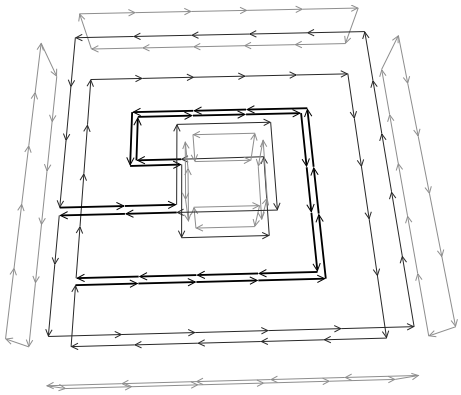
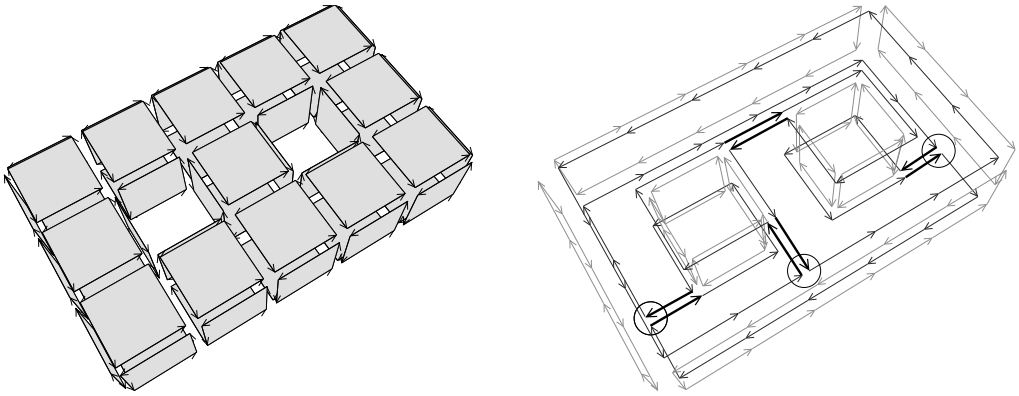


FIG. 5.26 – La carte de niveau 2 du même tore, avec de nombreuses arêtes fictives.



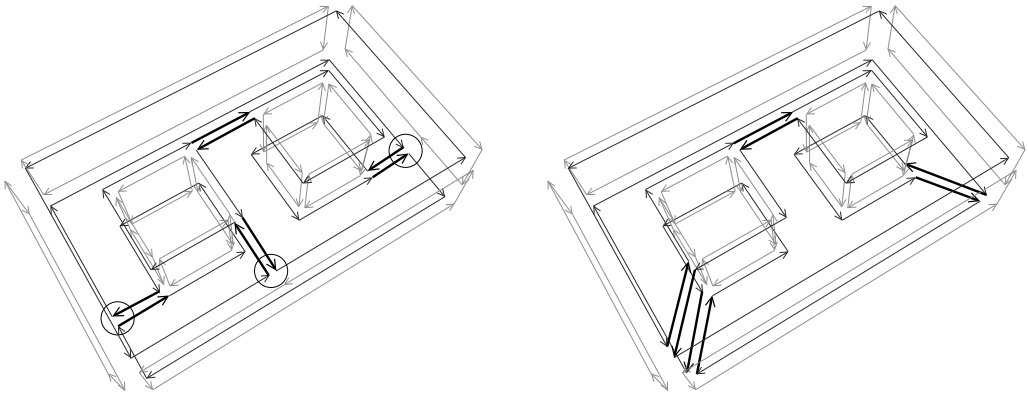
a. Le niveau 1. b. Le niveau 2 et ses arêtes fictives.

FIG. 5.27 – Les cartes de niveaux 1 et 2 d'un tore à deux trous.

veaux de simplification, que nous ne détaillons pas ici étant donné que nous obtenons exactement les mêmes cartes de niveaux 4 et 5 que pour le tore précédent (cf. figure 5.23).

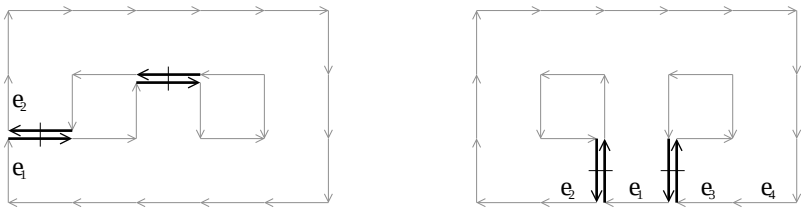
Nous étudions maintenant un dernier exemple un peu plus complexe, non pas pour introduire un nouveau problème, mais simplement pour vérifier le résultat obtenu pour un objet plus complexe : c'est le cas du tore à deux trous. Nous pouvons voir figure 5.27.*a* la carte de niveau 1 d'un tel tore, totalement inclus dans une autre région comme pour nos précédents exemples. Figure 5.27.*b* est présentée la carte de niveau 2 correspondante. Comme les faces supérieure et inférieure du tore possèdent chacune deux trous, nous avons donc pour chacune de ces faces deux arêtes fictives permettant de les conserver connexes. La figure 5.28.*a* présente la carte de niveau 3 obtenue en fusionnant uniquement les arêtes incidentes à des sommets de degré deux, et la figure 5.28.*b* la carte de niveau 3 correcte que nous obtenons avec les définitions modifiées.

Afin de bien comprendre le passage entre le niveau 2 et le niveau 3, et par la même occasion le principe d'extraction modifiée, nous l'étudions pour deux configurations différentes d'arêtes fictives. Pour simplifier les schémas et en faciliter la compréhension, nous nous limitons à l'étude



a. Problème pour le niveau 3. b. Le niveau 3 correct.

FIG. 5.28 – Le niveau 3 du tore à deux trous : le problème rencontré si l'on ne traite pas les arêtes fictives de manière particulière, et la carte de niveau 3 correcte.



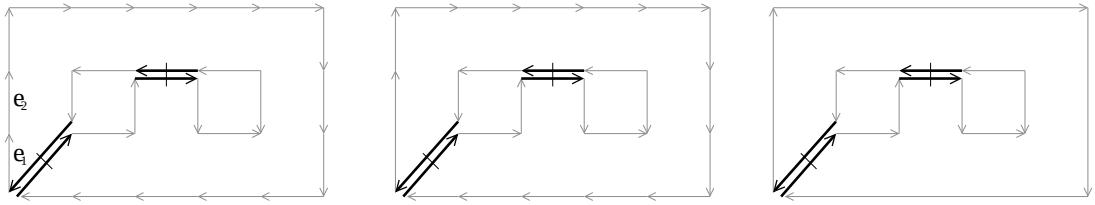
a. b.

FIG. 5.29 – La face supérieure du tore à deux trous avec différentes positions des arêtes fictives.

de la face supérieure de ce tore. Nous présentons figure 5.29 cette face supérieure de la carte de niveau 2 présentée 5.27.*b*. Sur cette figure, nous avons dessiné comme précédemment les arêtes fictives en noir épais, et représenté la relation β_2 au moyen des petits segments de droite. Nous avons représenté deux possibilités pour cette face supérieure, suivant la position des arêtes fictives, mais il en existe d'autres.

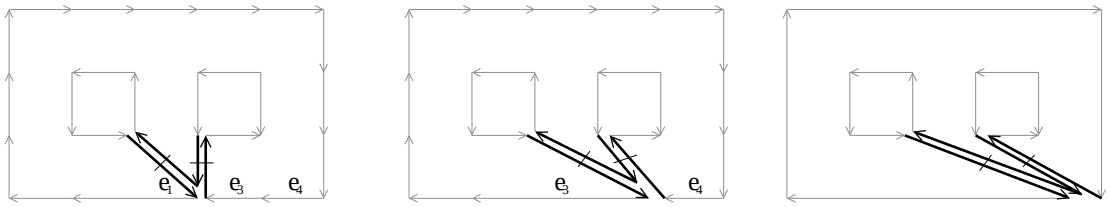
Nous pouvons voir figure 5.30.*a* le résultat du déplacement de l'arête fictive, pour la face supérieure du tore présenté figure 5.29.*a*. Après avoir décalé cette arête, le sommet incident à e_1 et e_2 est vraiment de degré 2, même en tenant compte des arêtes fictives. Nous pouvons donc ensuite effectuer la fusion d'arêtes de manière normale, et obtenir la carte présentée figure 5.30.*b*. Au final, nous obtenons, pour la face supérieure du tore, la carte de niveau 3 présentée figure 5.30.*c*. Nous pouvons vérifier, sur cette carte, que les fusions d'arêtes réelles ont toutes été réalisées de manière identique aux fusions effectuées si nous n'avions pas d'arête fictive.

Nous montrons maintenant quelques étapes intermédiaires de la construction de la carte de niveau 3 pour la deuxième configuration de la face supérieure du tore à deux trous présentée figure 5.29.*b*. Cette face diffère de la précédente par la position des arêtes fictives. La figure 5.31.*a* montre la carte obtenue après la fusion des arêtes e_1 et e_2. Après cette fusion, deux arêtes fictives sont incidentes au même sommet. L'opération de décalage effectuée avant la fusion des deux



a. Après avoir décalé l'arête fictive. b. Après la fusion de e_1 et e_2. c. La carte de niveau 3 finale.

FIG. 5.30 – La face supérieure du tore à deux trous (figure 5.29.*a*) : quelques étapes avant d'arriver à la carte de niveau 3.



a. Après la fusion de e_1 et e_2. b. Après la fusion de e_3 et e_1. c. La carte de niveau 3 finale.

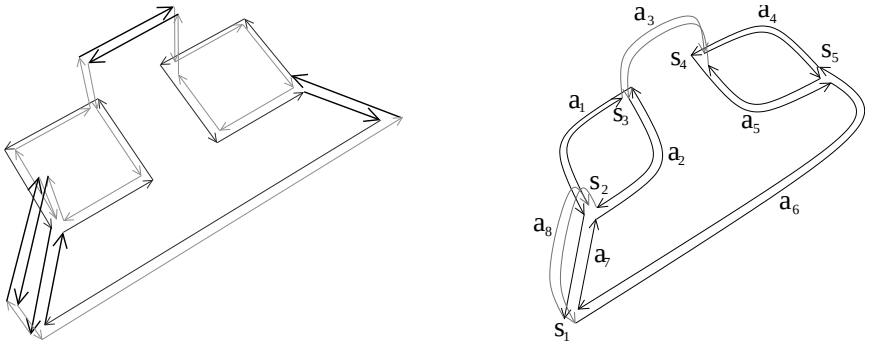
FIG. 5.31 – La face supérieure du tore à deux trous (figure 5.29.*b*) : quelques étapes avant d'arriver à la carte de niveau 3.

arêtes e_3 et e_1 va donc décaler deux arêtes fictives, et non plus une seule comme pour les exemples précédents. La figure 5.31.*b* montre la carte obtenue après la fusion de e_3 et e_1, et la figure 5.31.*c* la carte de niveau 3 obtenue au final. Cette carte est identique à la carte de niveau 3 obtenue en partant de l'autre configuration, à l'exception de la position des arêtes fictives.

Pour le tore entier, nous obtenons la carte de niveau 3 déjà présentée figure 5.28.*b*. À partir de ce niveau, nous fusionnons chaque couple de faces adjacentes et incidentes à une arête de degré un ou deux, n'entraînant pas de déconnexion de face, ni la suppression totale de la face (définition 30 du niveau 4 modifié). La carte de niveau 4 obtenue est présentée figure 5.32.*a*. Cette carte est composée d'une seule face, de dix-sept arêtes et de quatorze sommets, et la formule d'Euler donne bien un genre égal à deux.

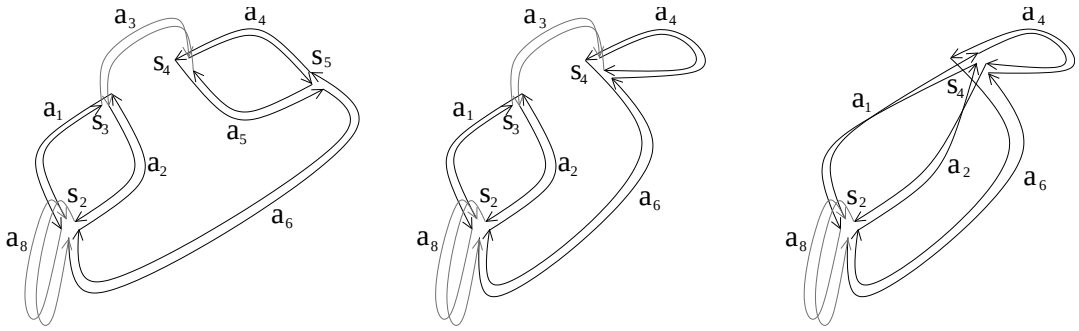
La figure 5.32.*b* montre la carte de niveau 5 en cours de construction. Nous avons pour le moment fusionné uniquement les arêtes adjacentes et incidentes à des sommets de degré deux. Cette carte est maintenant composée d'une face, de huit arêtes et de cinq sommets : nous caractérisons toujours un double dore. Mais cette carte n'est pas une représentation minimale. Nous continuons donc de fusionner les arêtes respectant les contraintes de la définition 32 du niveau 5 modifié.

Nous illustrons de manière progressive la construction de la carte de niveau 5 à partir de cette carte. Remarquons que, comme pour le cas du tore étudié figure 5.23, nous montrons ici un exemple de fusions d'arêtes. En effet, l'ordre dans lequel sont effectuées ces fusions est totalement



a. Le niveau 4. b. Le niveau 5 en cours de construction.

FIG. 5.32 – Le niveau 4 du tore à deux trous, et le niveau 5 en cours de construction, obtenu en fusionnant seulement les arêtes incidentes à des sommets de degré deux.



a. Après fusion de a_8 et a_7 le long de s_1. b. Après fusion de a_6 et a_5 le long de s_5. c. Après fusion de a_1 et a_3 le long de s_3.

FIG. 5.33 – Quelques étapes intermédiaires de la construction du niveau 5 du tore à deux trous.

arbitraire et aurait donc pu être différent. Mais nous pouvons vérifier que quel que soit cet ordre, nous obtenons toujours la même carte finale.

La figure 5.33.*a* présente la carte obtenue à partir de la carte précédente (cf. figure 5.32.*b*) en fusionnant les arêtes a_8 et a_7 le long du sommet s_1. Les arêtes fictives différentes de a_8 et a_7 sont dans un premier temps décalées sur le deuxième sommet incident à a_8, c'est-à-dire s_2. En pratique, une seule arête répond à ce critère et est donc décalée : c'est l'arête a_6.

Nous fusionnons ensuite les arêtes a_6 et a_5 autour du sommet s_5. Nous obtenons alors la carte présentée figure 5.33.*b*. Avant cette fusion, l'arête a_6 est décalée sur s_4 qui est le deuxième sommet incident à a_6. La figure 5.33.*c* montre la carte obtenue après la fusion des arêtes a_1 et a_3 le long de s_3. Nous pouvons remarquer que chaque fusion entraîne la suppression d'un sommet et d'une arête topologique, car nous fusionnons forcément deux arêtes n'étant pas des boucles. Cette avant-dernière carte possède deux sommets, cinq arêtes et une face.

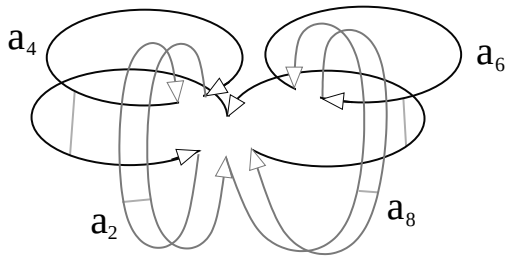


FIG. 5.34 – La carte de niveau 5 du tore à deux trous, obtenue par exemple après la fusion de a_6 et a_1 le long de s_2.

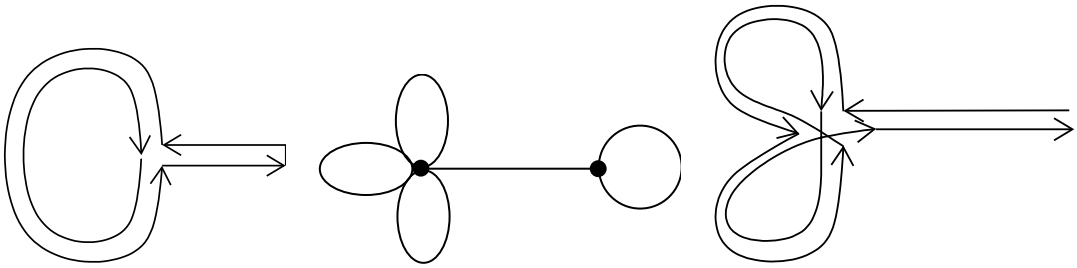
Enfin, nous fusionnons les arêtes a_6 et a_1 le long du sommet s_2. Nous obtenons alors la carte topologique présentée figure 5.34. Nous avons réorganisé les arêtes pour rendre cette figure plus compréhensible. Cette carte est la seule représentation minimale du tore à deux trous ayant une seule face. Elle est composée d'un sommet, quatre arêtes et une face. Chaque arête est une boucle, et nous ne pouvons donc plus effectuer aucune fusion. Nous obtenons bien, une nouvelle fois, la représentation minimale.

5.3.4 Preuve de la minimalité

Les exemples étudiés permettent de vérifier que les nouvelles définitions des niveaux de simplification amènent bien chaque fois à la représentation minimale. Cette représentation minimale est unique car le nombre de faces est fixée : il est égal au nombre de faces frontières de l'image. Nous avons étudié uniquement des objets totalement inclus dans une autre région, car ce sont ceux pour lesquels la gestion des arêtes fictives est primordiale afin d'obtenir cette représentation. De plus, nous avons vu lors de la présentation de notre premier exemple qui ne contenait pas d'arête fictive, que la représentation minimale est obtenue directement et sans problème. Nous donnons ici la preuve informelle afin de montrer que la carte topologique est la représentation minimale. La preuve formelle est possible mais serait très longue et technique ; nous préférons en exposer seulement les idées générales.

Nous classons les régions en deux catégories distinctes : celles représentées par des surfaces fermées et les autres. Remarquons tout d'abord que les seuls objets topologiques représentés sont des sphères et des tores à n trous. En effet, chaque surface est orientable et fermé. Avec la relation d'inclusion, nous pouvons caractériser plus finement une région, en parlant par exemple de sphère contenant un tore, mais toujours uniquement à partir de sphères et de n-tores. Nous ne traitons pas ici le cas de la sphère, pour lequel nous avons déjà montré section 5.3.2 que la représentation obtenue est bien minimale.

Pour les objets représentés par une surface fermée, la carte de niveau 3 correspondante est composée uniquement d'arêtes de degré un ou deux. En effet, cet objet est forcément totalement inclus dans une autre région, sinon sa surface ne serait pas fermée. Au cours du calcul de la carte de niveau 4, nous fusionnons toutes les faces le long de toutes les arêtes, à l'exception des fusions entraînant des déconnexions. Une déconnexion de face peut survenir uniquement pour une fusion de deux faces le long d'une arête de degré un (cf. section 5.3.2). Comme la carte de niveau 4



a. b. c.

FIG. 5.35 – Trois configurations impossible à obtenir pour la carte de niveau 5.

représentant cet objet est connexe et que toutes ses arêtes sont fictives, nous en déduisons que cette carte est composée d'une seule face. Nous allons ensuite, lors du calcul de la carte de niveau 5 fusionner chaque couple d'arêtes non boucles, étant donné que la carte est entièrement composée d'arêtes fictives. Nous arrivons forcément à une carte ayant un seul sommet et uniquement des arêtes en boucles.

Afin de prouver cela, il suffit de montrer qu'il est impossible d'arriver à une configuration semblable à celles présentées figure 5.35. En effet, ce type de configuration possède deux sommets différents, et une seule arête non boucle. Dans ce cas, il n'y a plus de couple d'arêtes non boucle à fusionner, et la carte de niveau 5 finale ne possède pas un seul sommet. Mais ces configurations ne peuvent pas survenir. En effet, lors de la construction de la carte de niveau 4, toutes les faces pouvant être fusionnées sans entraîner de déconnexion l'ont été. Or, dans ces cas, ainsi que dans les cas similaires, certaines fusions de faces n'entraînent pas de déconnexion. C'est évident pour le cas présenté figure 5.35.*a*, un peu plus difficile à voir pour celui de la figure 5.35.*b* qui ne représente pas la carte mais la schématise par ses sommets et arêtes.

Intuitivement, la carte de niveau 4 est composée uniquement d'arêtes de degré un, et ne possède aucun sommet de degré un. En effet, un tel sommet est adjacent à une seule arête, et la fusion de faces le long de cette arête n'entraîne pas de déconnexion. De plus, si la carte de niveau 4 n'a pas de sommet de degré un, alors la carte de niveau 5 n'en n'a pas non plus. En effet, seules les arêtes incidentes à des sommets de degrés deux sont fusionnées, et nous ne pouvons pas obtenir une carte composée seulement d'une boucle. Les seules configurations possibles ressemblant à celle de la figure 5.35.*b* vérifiant ces deux contraintes sont similaires à celle présentée figure 5.35.*c*. La configuration des brins appartenant aux deux boucles est la seule possible afin de ne pas avoir de sommet de degré un. Cette configuration est extensible à un nombre quelconque de boucles. Dans ce cas, nous pouvons vérifier que la fusion de faces le long d'une de ces deux boucles n'entraîne pas de déconnexion, ce qui montre que ce type de cas ne peut pas se produire.

Comme ces configurations n'existent jamais, lorsque nous avons plus de deux sommets, il existe toujours au moins deux arêtes n'étant pas des boucles et pouvant donc être fusionnées. Chaque fusion d'arêtes diminue les nombres de sommets et d'arêtes de un, au final la carte obtenue possède toujours un seul sommet et plusieurs arêtes, suivant le genre du tore. Un tore à n trous, est donc représenté par une face, un sommet et $2n$ arêtes, ce qui est bien la représentation minimale.

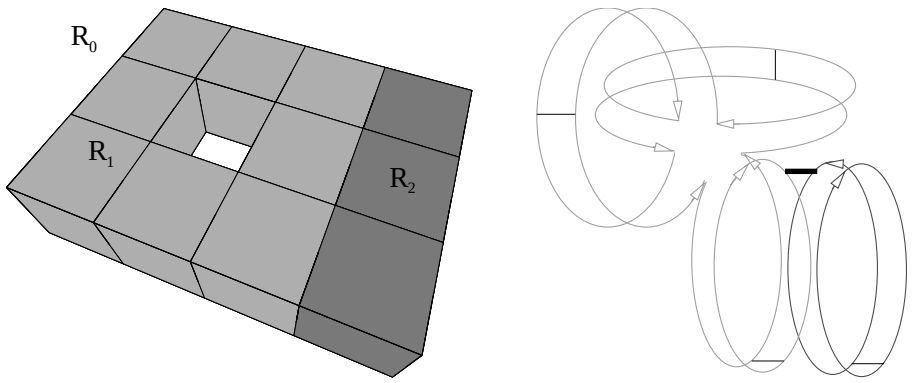


FIG. 5.36 – Un tore adjacent à un autre volume.

Lorsqu'un objet est adjacent à d'autres régions, il est représenté par un certain nombre de faces frontières. Ces faces frontières ne peuvent pas être fermées car une telle face représente entièrement un volume, et nous sommes alors dans le cas d'un objet totalement inclus dans une autre région. Ces faces possèdent toutes un ou plusieurs bords, représentés par des arêtes réelles. Ces bords sont minimaux car chaque couple d'arêtes incidentes à des sommets de degré deux ont été fusionnées. Lorsqu'une face frontière est représentée par k bords, il y a $k-1$ arêtes fictives permettant de conserver cette face connexe. Ces arêtes fictives sont bien en nombre minimal. Mais certaines faces peuvent représenter plus d'information que seulement leurs bords.

Ce cas est illustré par la figure 5.36. Cet exemple représente un tore (région R_1) adjacent à un cube topologique (région R_2), tous deux inclus dans la région infinie (R_0). Sur la carte topologique correspondante, nous ne représentons pas les brins de la région infinie. Le tore est représenté par deux faces, une pour la frontière entre R_1 et R_2, et une autre pour celle entre R_1 et R_0. La région R_1 est représentée par deux faces, trois arêtes et un sommet, ce qui correspond bien au genre d'un tore. La première face est simplement une boucle, car tous les sommets appartenant à son bord sont de degré deux. La deuxième face est composée de trois arêtes et d'un sommet. En effet, cette face « contient » l'information que l'objet modélisé est un tore. Elle est composée de deux arêtes fictives, plus un brin appartenant à l'arête réelle. Le brin appartenant à l'arête réelle ne peut être ni déplacé, ni supprimé. Par contre, les autres arêtes étant fictives, elles peuvent être déplacées. Intuitivement, elles vont être positionnées sur le sommet incident à l'arête réelle. Elles vont être ensuite fusionnées lors du calcul de la carte de niveau 5, tant qu'il existe deux arêtes non boucles. Nous obtenons finalement uniquement des boucles incidentes à un seul sommet, en montrant que les configurations présentées figure 5.35 ne peuvent pas non plus se produire ici.

5.4 Le plongement de la carte topologique

Nous étudions maintenant la manière de plonger la carte topologique. Nous nous limitons à cette carte de niveau 5, tout d'abord car c'est celle qui nous intéresse le plus, mais également car c'est la plus délicate à plonger. En effet, pour les niveaux 1, 2 et 3, les solutions déjà présentées en dimension 2 peuvent s'étendre facilement. Par exemple pour la carte de niveau 3, nous pouvons

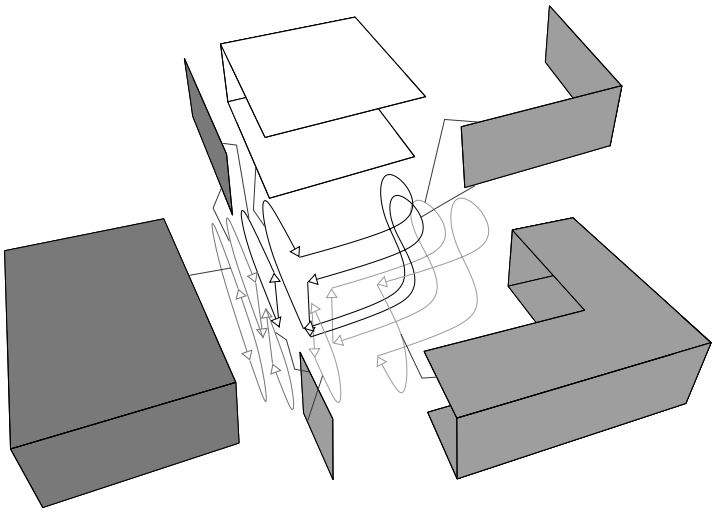


FIG. 5.37 – La carte de niveau 5 de notre premier exemple, avec son plongement.

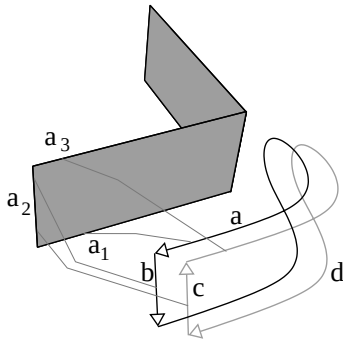


FIG. 5.38 – Zoom sur une face topologique et son plongement.

associer à chaque sommet topologique les coordonnées d'un point géométrique. Étant donné que chaque arête de ce niveau est un segment de droite, ce plongement simple suffit pour reconstruire le plongement de chaque cellule topologique.

5.4.1 Plongement face

Une première solution afin de plonger la carte topologique consiste à associer une surface à chaque face de la carte. Cette solution est relativement simple à mettre en œuvre, étant donné que nous plongeons un seul type de cellule. Nous pouvons voir figure 5.37 ce plongement pour la carte topologique du premier exemple que nous avons étudié. Nous pouvons vérifier sur cet exemple que deux demi-faces topologiques (β_3-cousues entre elles) sont associées à la même surface, comme pour la face distinguée figure 5.38. Mais afin de reconstruire correctement les plongements de chaque cellule, les liaisons entre les brins de la carte et le plongement doivent être

réalisées « correctement ». Un brin b doit désigner le bord de la surface de plongement représentant l'arête incidente à b, et le sommet incident à b.

Sur notre exemple figure 5.38, le brin a désigne l'arête a_1 du plongement, du côté du sommet entre les arêtes a_1 et a_2, et le brin d (qui est β_3-cousu au brin a) désigne l'arête a_3 du côté du sommet entre les arêtes a_3 et a_2. Pour récupérer le plongement d'un sommet topologique, il suffit de récupérer les coordonnées du sommet du bon côté de l'arête désignée par ce brin. Pour récupérer le plongement d'une arête topologique incidente à un brin b, il faut parcourir le bord de la surface de plongement à partir de l'arête désignée par b, jusqu'à l'arête de plongement désignée par $\beta_3(b)$. Nous obtenons une suite de coordonnées qui est une courbe 1d représentant le plongement de cette arête. Pour récupérer le plongement d'une face topologique, il suffit de récupérer toute la surface de plongement, et enfin pour récupérer le plongement d'un volume topologique, il faut récupérer l'ensemble des plongements des faces topologiques qui le compose, et les mettre en contact suivant les β_2-coutures de la carte.

Une possibilité intéressante pour représenter ce plongement est l'utilisation des 2-G-cartes. Nous devons en effet représenter des surfaces qui sont donc des objets de dimension 2. De plus, il est intéressant ici d'utiliser les G-cartes au lieu des cartes afin de profiter de leur absence d'orientation. Si nous utilisons les cartes, chaque surface de plongement est représentée par une carte, donc avec une certaine orientation. Certaines opérations vont entraîner la fusion de deux surfaces de plongement. Si les deux orientations de ces surfaces ne sont pas identiques, nous devons inverser une des deux cartes avant de pouvoir les fusionner. Ce problème est résolu par l'utilisation des G-cartes, étant donné qu'elles ne sont pas orientées. Ces G-cartes sont l'équivalent des cartes topologiques en dimension 2, c'est-à-dire qu'elles ne contiennent pas de sommet de degré deux, et que nous associons à chacune de leurs arêtes un plongement arête ouverte et à chacun de leurs sommets les coordonnées d'un point 3d.

Ces 2-G-cartes de plongement représentent la géométrie des faces topologiques de la carte, comme nous pouvons le voir sur l'exemple de la figure 5.39. Chaque brin d'une même face désigne un brin du bord de la même 2-G-carte de plongement, à l'exception des brins appartenant à des arêtes fictives. En effet, une arête fictive est une arête « à l'intérieur » d'une même face, et sera donc associée à une arête n'appartenant pas au bord de la 2-G-carte de plongement. De manière intuitive, ce plongement peut être vu comme le complémentaire de la carte topologique. Tous les brins supprimés lors de la construction des niveaux 4 et 5 sont ajoutés dans des 2-G-cartes de plongement. En effet, ce plongement est un peu l'équivalent de la carte de niveau 3 étant donné qu'il représente les arêtes alignées de longueur maximale et uniquement les faces coplanaires. De plus, comme chaque fusion effectuée lors de la construction des niveaux 4 et 5 entraîne une perte d'information géométrique, cette information est conservée dans les 2-G-cartes. Nous pouvons voir figure 5.40 comment est réalisé ce plongement pour la face présenté figure 5.38. Les deux demi-faces topologiques pointent vers des brins du bord de la même 2-G-carte.

L'inconvénient de ce plongement est le même que celui rencontré en dimension 2 : il représente des informations géométriques de manière redondante. En effet, un sommet sera représenté dans chaque G-carte de plongement, donc autant de fois qu'il y a de faces incidentes à ce sommet dans la carte. Pour l'exemple figure 5.39, le sommet central est incident à chacune des 6 faces topologiques. Il est donc représenté 6 fois, une fois dans chaque G-carte de plongement. Ces informations redondantes entraînent une perte en espace mémoire, mais également en temps

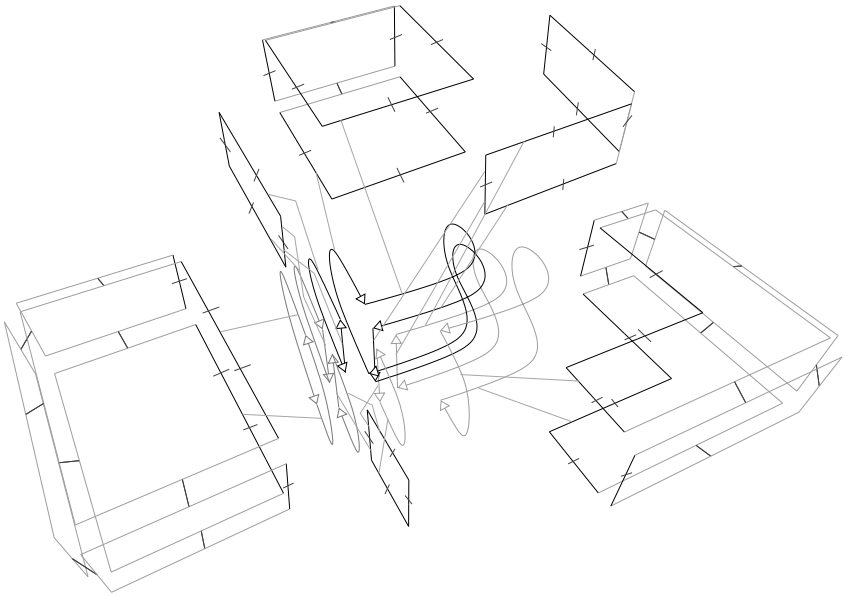


FIG. 5.39 – Le plongement face réalisé avec des 2-G-cartes.

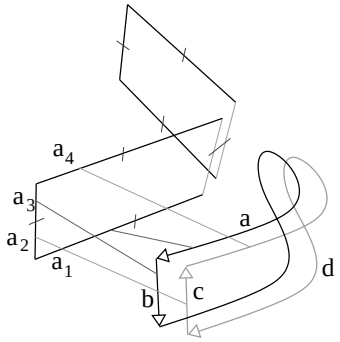


FIG. 5.40 – Zoom sur une face topologique et son plongement réalisé avec des 2-G-cartes.

d'exécution. En effet, lorsque nous voulons modifier les coordonnées d'un sommet topologique, nous devons le faire pour chacune des G-cartes de plongements le contenant.

5.4.2 Plongement face ouverte, arête ouverte et sommet

Un deuxième plongement, permettant d'éviter la représentation redondante de cellule géométrique, consiste à plonger chaque face topologique par une surface ouverte (c'est-à-dire sans son bord), chaque arête topologique par une courbe 1d ouverte (sans ses sommets extrémités) et chaque sommet topologique par un point géométrique.

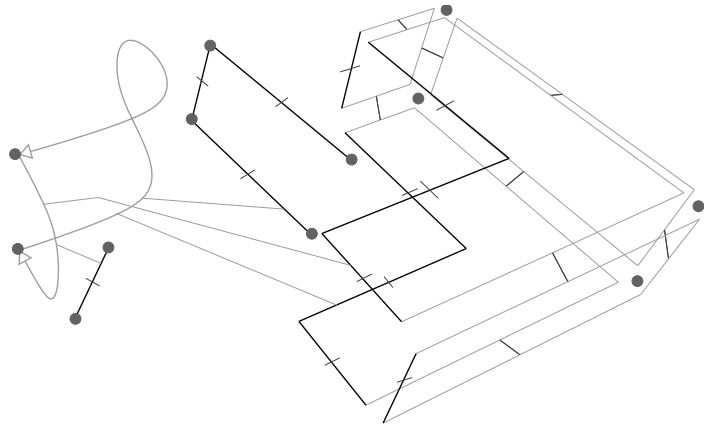


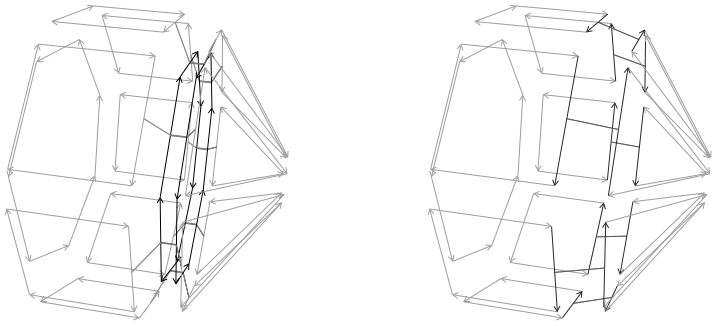
FIG. 5.41 – Le plongement face ouverte, arête ouverte et sommet pour une face topologique de notre exemple.

Comme pour le plongement précédent, une manière d'implanter ce plongement consiste à associer à chaque face une 2-G-carte représentant la surface géométrique correspondante, mais cette fois sans en plonger le bord. Ce plongement sera en effet reconstruit à l'aide des plongements arêtes et sommets. Nous associons également à chaque arête topologique une 1-G-carte de plongement, représentant la courbe 1d ouverte et enfin à chaque sommet topologique les coordonnées du point géométrique correspondant.

Nous ne donnons pas d'exemple précis de ce plongement, car les figures correspondantes sont illisibles, mais présentons simplement figure 5.41 le plongement d'une face de la carte topologique utilisée pour le plongement précédent. Sur cette figure, nous voyons que les deux sommets topologiques de la face sont associés à deux sommets géométriques (symbolisés par les points). Les deux arêtes pointent vers des 1-G-cartes représentant le plongement des arêtes ouvertes. Enfin, chaque brin pointe vers un brin du bord de la 2-G-carte de plongement, qui est surface « ouverte », c'est-à-dire que son bord ne possède pas de plongement.

L'avantage de ce plongement est que, contrairement au précédent, il ne code aucune information géométrique de manière redondante. Mais nous voyons sur cet exemple qu'il est plus complexe à mettre en œuvre. De plus, afin de récupérer le plongement d'une cellule topologique, nous devons le reconstruire en combinant les plongements sommets, arêtes, et faces, ce qui entraîne un surcoût de complexité. Enfin, chaque brin doit maintenant conserver des informations sur trois types de plongements différents, ce qui entraîne une augmentation de l'espace mémoire nécessaire au codage d'un brin.

Il est clair que suivant les utilisations de la carte topologique et les besoins spécifiques d'une application particulière, nous utiliserons plutôt un plongement que l'autre. Mais une étude plus complète devra être effectuée afin de comparer l'espace mémoire et la complexité en temps d'exécution dans le cadre précis d'une application, et d'une implantation particulière. En effet, ces deux complexités dépendent fortement de la manière dont sont implantés les cartes topologiques et les plongements associés. C'est pour cette raison qu'il est difficile de faire ici cette étude de manière générale.



a. Avant la fusion. b. Après la fusion.

FIG. 5.42 – Fusion de volumes en 3d.

Dans la suite de ce travail, nous utilisons le premier plongement présenté afin de simplifier les explications. Mais ce choix n'est en aucun cas une contrainte, car il est facile de passer d'un plongement à l'autre, et car des fonctions génériques manipulent ce plongement sans dépendre de la manière dont il est implanté. De ce fait, pour changer le plongement utilisé, il suffira de modifier ces fonctions de base et non les fonctions de plus haut niveau manipulant la carte topologique.

5.5 Un premier algorithme d'extraction

Les définitions des différents niveaux de carte présentées section 4.2 donnent immédiatement, comme pour la dimension 2, un premier algorithme d'extraction qui en est leur transcription. Mais avant de présenter cet algorithme, nous revenons brièvement sur les opérations de fusion en 3d, qui sont à la base de ces définitions et de cet algorithme.

5.5.1 Les fusions dans les 3-cartes

Nous avons déjà vu, section 4.4.1, le principe général des fusions en dimension n. Étant donné que nous travaillons ici en dimension 3, nous avons trois types différents de fusions qui sont la fusion de volumes (dimension 3), la fusion de faces (dimension 2) et la fusion d'arêtes (dimension 1). La fusion de deux i-cellules C_1 et C_2 le long d'une $(i-1)$-cellule C n'est possible uniquement lorsque C est de degré 1 ou 2.

La figure 5.42 présente un exemple de fusion de volumes. La figure 5.42.*a* montre une carte représentant deux objets β_3-cousus le long de la face dessinée en foncé, et la figure 5.42.*b* le résultat de la fusion de ces deux volumes.

L'algorithme 19 effectuant cette opération est simple à définir. Il est local car il traite un ensemble de brins (ici les brins appartenant à la demi-face[2] incidente à b) sans ordre particulier, en effectuant un traitement dépendant uniquement de la configuration locale autour de ce brin. De manière générale, cette technique simplifie le nombre de cas différents à traiter et donc l'écriture des algorithmes.

[2]Rappelons que les brins de la demi-face incidente à un brin b sont ceux de l'orbite $< \beta_1 > (b)$.

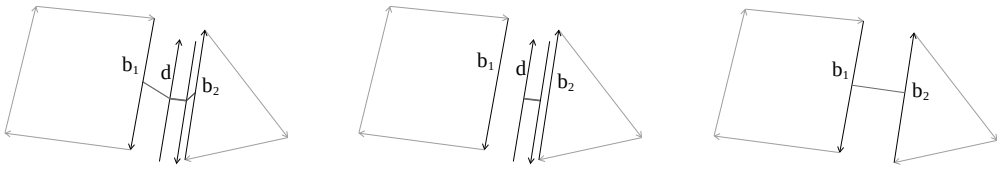
Algorithme 19 Fusion de volumes en 3d

Entrée : Un brin b β_3 cousu

Résultat : Les deux volumes incidents à b et à $\beta_3(b)$ sont fusionnées.

pour chaque *brin d appartenant à la demi-face incidente à b* **faire**

$b_1 \leftarrow \beta_2(d)$; $b_2 \leftarrow \beta_{32}(d)$;
β_2-découdre(d) ; β_2-découdre$(\beta_3(d))$;
β_2-coudre(b_1, b_2);
détruire les brins $\beta_3(d)$ et d;



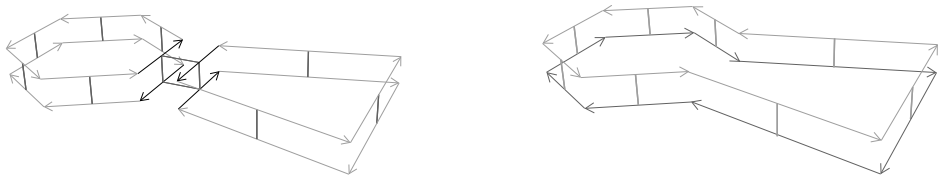
a. Avant de traiter d. b. β_2 décousures de d et $\beta_3(d)$. c. β_2 couture de b_1 et b_2.

FIG. 5.43 – Fusion de volumes en 3d, un traitement local.

Pour chaque brin d de cette demi-face, nous commençons par conserver les brins qui sont β_2-cousus à d et à $\beta_3(d)$ (figure 5.43.*a*), puis nous β_2-décousons ces deux brins (figure 5.43.*b*), et enfin nous β_2-cousons b_1 et b_2, les deux brins initialement cousus à d et $\beta_3(d)$, et détruisons les brins $\beta_3(d)$ et b (figure 5.43.*c*). Ce traitement étant effectué pour chaque brin de la demi-face, nous obtenons bien au final la carte présentée figure 5.42.*b*, résultat de la fusion des deux volumes. La complexité de cet algorithme est linéaire en le nombre de brins de la demi-face incidente au brin donné en entrée de l'algorithme, et ce quelle que soit la taille des deux régions. Nous voyons ici un exemple de traitement où les cartes combinatoires permettent un gain important en complexité, par rapport au traitement similaire par exemple sur la matrice de voxels. De plus, cet algorithme fonctionne quelle que soit la configuration de la carte, à condition que le brin de départ de l'algorithme soit β_3-cousu. Il fonctionne de manière identique lorsque la face le long de laquelle nous effectuons la fusion est de degré un (cas d'une face à l'intérieur d'un même volume), ce qui permet de ne pas différencier ces deux cas.

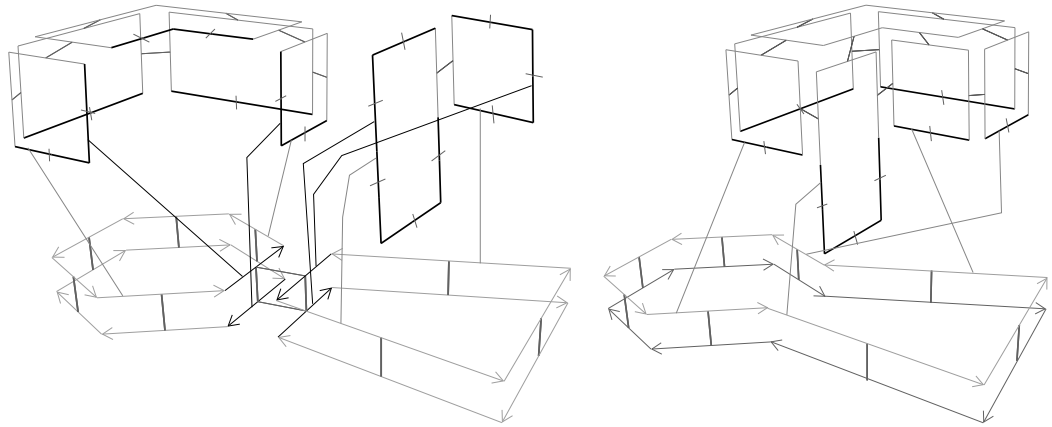
Cet algorithme présente seulement la partie topologique de la fusion de volumes. Mais aucune modification de plongement n'est nécessaire. En effet, nous ne modifions aucun plongement de face, nous détruisons seulement celui associé à la face topologique supprimée. Si nous avons un plongement sommet, arête ouverte et face ouverte, les opérations de couture et décousure se chargent des mises à jour nécessaires. Par exemple, si lors d'une décousure, nous déconnectons un sommet topologique en deux, le plongement sommet correspondant sera dupliqué afin que chaque sommet topologique ait exactement un plongement sommet. De manière réciproque, si une opération de couture regroupe deux orbites sommets distinctes, l'un des deux plongements sommets sera supprimé. Ces opérations sont faites de manière transparente, ce qui permet de changer le plongement sans avoir à changer quoi que ce soit dans les opérations de haut niveau.

La fusion de faces en 3d peut être vue comme deux fusions de faces 2d. En effet, une face topologique en dimension 3 est composée de deux demi-faces identiques β_3-cousues entre elles.



a. Avant la fusion. b. Après la fusion.

FIG. 5.44 – Fusion de faces en 3d.

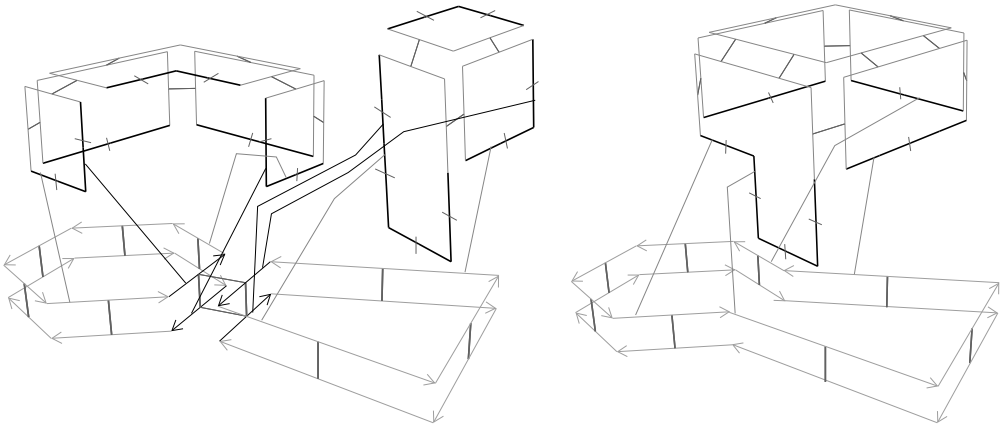


a. Avant la fusion. b. Après la fusion.

FIG. 5.45 – Modification du plongement lors de fusion de faces 3d.

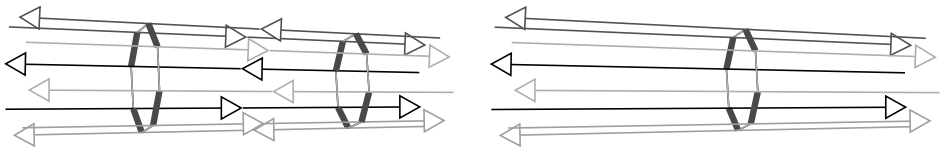
La figure 5.44.*a* montre deux faces adjacentes, et incidentes à une arête de degré deux, et la figure 5.44.*b* le résultat de leur fusion de long de cette arête. L'algorithme effectuant cette fusion peut se ramener à deux appels consécutifs de l'algorithme effectuant la fusion de faces en 2d, une fois pour le brin b et une deuxième fois pour le brin $\beta_3(b)$. Nous ne le détaillons donc pas ici. Cet algorithme fonctionne quelle que soit la configuration des deux faces, à condition que le brin donné à l'algorithme soit β_2-cousu.

La partie topologique effectuant cette fusion de faces ne pose donc pas de problème. Nous présentons maintenant brièvement les modifications du plongement. Nous considérons ici que nous avons un plongement uniquement face, implanté avec des 2-G-cartes. La figure 5.45.*a* présente les deux mêmes faces topologiques de l'exemple précédent, mais maintenant avec leur plongement. Afin de ne pas rendre les figures illisibles, nous avons seulement représenté quelques liaisons entre la carte et le plongement. Ces deux faces étant différentes, elles possèdent donc deux G-cartes de plongement distinctes. Après la fusion de ces deux faces, nous devons modifier le plongement afin qu'il tienne compte de cette fusion. Nous pouvons voir le résultat de cette modification figure 5.45.*b*. Afin d'effectuer cette modification, nous devons simplement α_2-coudre deux à deux tous les brins de plongement représentant l'arête supprimée lors de la fusion. Cette opération s'effectue sans difficulté particulière. De plus, nous utilisons les fonctions de la 2-G-carte, qui suivant son plongement font les mises à jour nécessaires afin qu'il reste valide. De ce fait, nous n'avons



a. Avant la fusion. b. Après la fusion.

FIG. 5.46 – Un autre exemple de modification du plongement lors de fusion de faces 3d.



a. Avant la fusion. b. Après la fusion.

FIG. 5.47 – Fusion d'arêtes en 3d.

pas besoin de nous préoccuper de la manière dont sont plongées les 2-G-cartes. Nous pouvons également remarquer qu'il n'y a pas d'autre modification à faire : chaque brin de la carte conserve le même lien vers un brin de plongement, si l'on excepte les 4 brins supprimés lors de la fusion.

Mais cette opération de modification de plongement n'est pas tout le temps une α_2-couture, comme nous pouvons le voir sur l'exemple présenté figure 5.46. En effet, dans ce cas, nous α_2-cousons des faces coplanaires. Afin de conserver les G-cartes minimales, nous devons alors fusionner ces faces après les avoir cousues. Ce traitement est effectué de manière transparente par la G-carte sans traitement supplémentaire. De plus, après ces opérations de fusions, certaines arêtes peuvent être fusionnées lorsqu'elles sont incidentes à un sommet de degré deux. Mais ces fusions sont également effectuées de manière transparente par la G-carte. C'est pour ces raisons que nous ne détaillons pas dans la plupart de nos algorithmes la gestion du plongement. Lorsque nous devons modifier le plongement de manière précise, nous employons alors des méthodes génériques, comme par exemple l'affectation des coordonnées d'un sommet topologique, l'affectation d'un plongement à une face, ou encore la fusion de deux plongements faces sans détailler ces opérations.

Nous étudions figure 5.47 un exemple de fusion d'arêtes le long d'un sommet de degré deux. Sur cette figure, nous avons représenté uniquement les deux arêtes à fusionner. Nous avons dessiné de même couleur deux brins cousu par β_2, (relation représentée par les trais gris clair fin), et

représenté β_3 par les traits épais et foncés. Les deux arêtes de la figure 5.47.*a* sont composées de 8 brins chacune. De manière générale, une arête possède au minimum 4 brins (car nous avons toujours des cartes fermées) et au maximum 8 brins car nous représentons des voxels et avons donc au maximum 4 faces incidentes à une arête (et deux brins par face). La figure 5.47.*a* montre le résultat de la fusion d'arêtes.

L'algorithme 20 effectue cette fusion d'arêtes. Pour qu'il puisse fonctionner correctement, le sommet entre les deux arêtes à fusionner doit être de degré deux. Cet algorithme n'est pas local,

Algorithme 20 Fusion d'arêtes en 3d

Entrée : Un brin b incident à un sommet de degré deux

Résultat : Les deux arêtes incidentes à b et à $\beta_0(b)$ sont fusionnées.

```
d ← β21(b);
β2-découdre(b);
old ← β0(b);
pair ← faux;
tant que d ≠ b faire
    b1 ← β0(d) ; b2 ← β1(d);
    β0-découdre(d) ; β1-découdre(d);
    β1-coudre(b1, b2);
    si pair est faux alors
        β2-découdre(d) ; β2-coudre(old, b1);
        new ← β31(d) ; β3-découdre(d);
    sinon
        β3-découdre(d) ; β3-coudre(old, b1);
        new ← β21(d) ; β2-découdre(d);
    pair ← ¬pair ; old ← b1;
    détruire le brin d;
    d ← new;
b1 ← β0(d) ; b2 ← β1(d);
β0-découdre(d) ; β1-découdre(d);
β1-coudre(b1, b2);
β3-découdre(d) ; β3-coudre(old, b1);
détruire le brin d;
```

contrairement à celui effectuant la fusion de volumes, car la définition des cartes combinatoires n'est pas homogène pour la définition des sommets[3]. De ce fait, nous devons parcourir les brins du sommet incident au brin de départ dans l'ordre, en alternant une involution β_2 et une involution β_3, et en effectuant un traitement légèrement différent dans ces deux cas.

Afin de comprendre le fonctionnement de cet algorithme, nous étudions son déroulement sur l'exemple présenté figure 5.48. Nous avons pris un exemple assez simple où les deux arêtes à fusionner sont composées chacune de 6 brins. La figure 5.48.*a* montre la configuration de départ, avant de rentrer dans la boucle « Tant que ». Nous avons initialisé certains brins, et β_2-décousu le brin b qui est le brin de départ. Dans la boucle, nous traitons le brin courant, d, et conservons le

[3]L'algorithme similaire pour les G-cartes serait lui local et donc beaucoup plus simple.

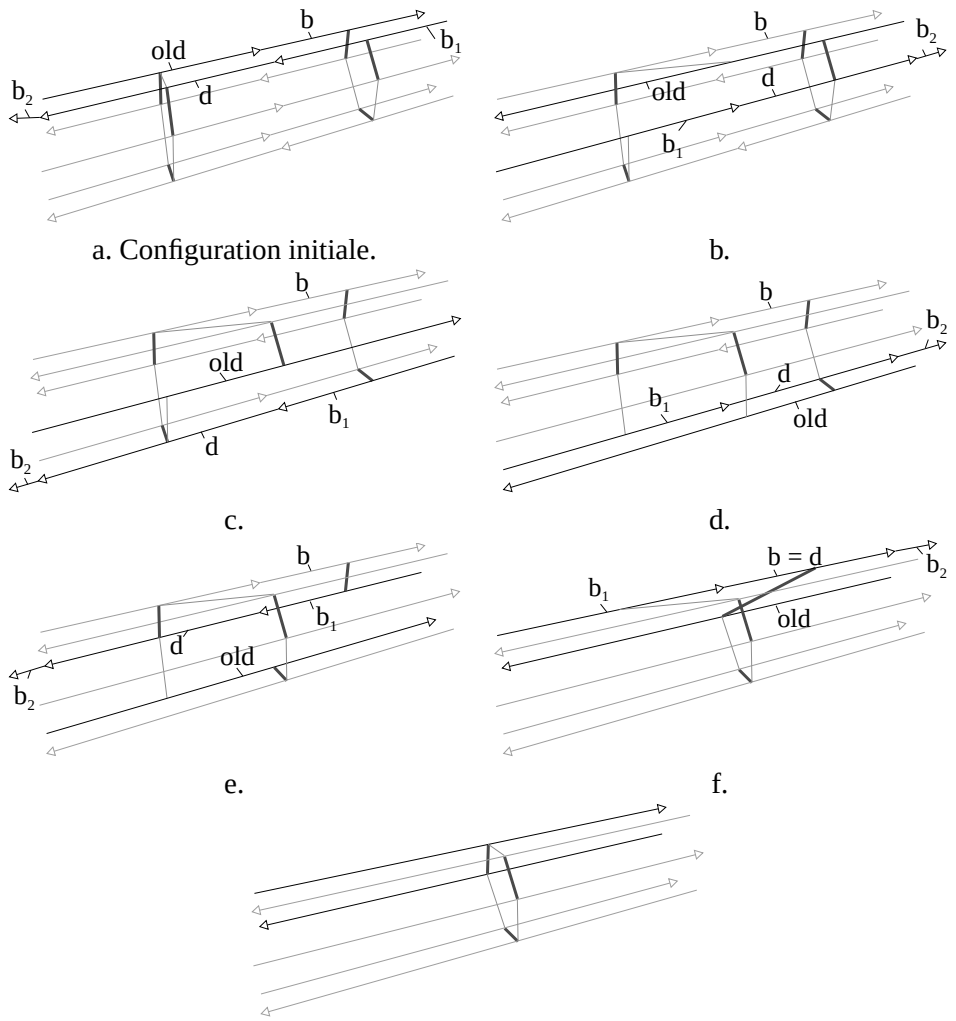


FIG. 5.48 – Déroulement de l'algorithme de fusion d'arêtes en 3d.

brin de la dernière arête traitée afin de le coudre correctement à la nouvelle arête, c'est le brin old. Nous commençons par effectuer le traitement local consistant à découdre d pour β_0 et β_1 puis par β_1-coudre b_1 et b_2 qui étaient les deux brins β_0 et β_1 cousus à d. Ensuite nous différencions deux cas, suivant que old est cousu par β_2 ou par β_3 à d. Ces deux cas se produisent alternativement, étant donné que nous utilisons β_2 et β_3 à tour de rôle. Pour cette première itération, nous sommes dans le premier cas où nous β_2-décousons old puis le β_2-cousons avec b_1 qui est localement le brin résultat de la fusion. Puis nous affectons le prochain brin à traiter, qui sera $\beta_{31}(d)$ (figure 5.48.*b*). Nous pouvons alors détruire le brin d, après l'avoir β_3-décousu, et traiter le prochain brin. Ce prochain brin aura un traitement similaire, sauf que nous sommes maintenant dans le deuxième cas étant donné que old est β_3-cousu à d. Le traitement de ce cas est similaire au précédent, en échangeant β_2 et β_3.

Nous traitons ainsi chaque brin de l'arête (cf. figures 5.48.*b* à *e*), et retombons finalement sur le brin de départ (cf. figure 5.48.*f*). Nous devons alors traiter ce dernier brin en dehors de la boucle, car son traitement est légèrement différent des autres brins. Nous obtenons alors la carte présentée figure 5.48.*g* qui est bien le résultat de la fusion des deux arêtes initiales.

Cet algorithme fonctionne dans n'importe quelle configuration, à condition que le sommet supprimé soit de degré deux, et que la carte soit fermée. Sa complexité est linéaire en le nombre de brins de l'arête. La modification du plongement est faite de manière transparente lors des opérations de coutures et décousures. Si les deux arêtes fusionnées sont alignées, les deux arêtes de plongement correspondantes seront également fusionnées. Sinon le traitement est dépendant du plongement utilisé. Nous pouvons par exemple, si les 2-G-cartes de plongement possèdent un plongement arête, fusionner quand même les deux arêtes et ajouter le plongement du sommet supprimé dans celui-ci. Si elles ont uniquement un plongement sommet, nous n'effectuons alors pas la fusion.

5.5.2 L'algorithme d'extraction naïf

L'algorithme 21 calculant la carte de niveau n d'une image 3d segmentée en régions est simplement la transcription des définitions de nos cinq niveaux de simplification. De plus, cet algorithme est l'extension directe de l'algorithme naïf en dimension 2. Il prend en entrée une image 3d segmentée en régions, et le niveau de simplification souhaité (un entier entre 1 et 5), et retourne la carte de ce niveau représentant l'image.

Algorithme 21 Extraction « naïve » de la carte de niveau n en 3d

Entrée : Une image I de $n_1 \times n_2 \times n_3$ voxels
n le niveau de la carte que nous voulons extraire
Sortie : La carte de niveau n de l'image I.

```
Construire la carte de niveau 0 correspondant à I;
Fusionner tout couple de volumes adjacents appartenant à la même région;
si n ≥ 2 alors
    Fusionner tout couple de faces adjacentes, coplanaires, incidentes à une arête de degré
    un ou deux et n'entraînant pas de déconnexion de face;
    si n ≥ 3 alors
        Fusionner tout couple d'arêtes adjacentes et alignées a1 et a2 incidentes à un sommet
        s tel qu'il n'existe pas d'arête réelle différente de a1 et de a2 incidente à s;
        si n ≥ 4 alors
            Fusionner tout couple de faces adjacentes, incidentes à une arête de degré un
            ou deux et n'entraînant pas de déconnexion de face ni la suppression totale de
            la face;
            si n = 5 alors
                Fusionner tout couple d'arêtes adjacentes a1 et a2 n'étant pas des boucles
                et incidentes à un sommet s tel qu'il n'existe pas d'arête réelle différente
                de a1 et de a2 incidente à s;
Calculer l'arbre d'inclusion des régions;
```

La première ligne de cet algorithme commence par construire la carte de niveau 0 correspondant à l'image I, c'est-à-dire la carte complète représentant chaque voxel de cette image plus une face représentant la région infinie et fermant cette carte. Ensuite, chaque ligne de l'algorithme permet de passer au niveau de simplification suivant, par application directe de la définition correspondante.

La ligne 2 construit la carte de niveau 1 correspondant à I, en fusionnant chaque couple de volumes adjacents appartenant à la même région. C'est uniquement pour la construction de ce niveau que nous effectuons le lien entre l'image à représenter et la carte. Tous les autres niveaux vont ensuite utiliser uniquement le degré des cellules et des tests de déconnexion. Comme pour la dimension 2, cette étape peut être réalisée en complexité linéaire en le nombre de brins de la carte. En effet, nous parcourons chaque face de la carte, testons si les deux volumes incidents sont de même région, et lorsque c'est le cas effectuons la fusion. Nous parcourons chaque face exactement une seule fois, car lorsqu'une fusion n'est pas effectuée à un moment de l'algorithme, nous savons qu'elle ne pourra jamais être réalisée et n'avons donc pas besoin de tester à nouveau cette face. De plus, cette technique de construction permet de traiter les cas où nous devons fusionner un volume avec lui même, et les cas où nous devons fusionner plusieurs fois le même couple de volumes le long de différentes faces.

La ligne 3 construit la carte de niveau 2, en appliquant la définition 29 qui tient compte des problèmes de déconnexion de face et conserve éventuellement des arêtes fictives afin de les résoudre. Pour réaliser cette étape, nous parcourons chaque arête de la carte et regardons si elle vérifie les conditions de la définition de ce niveau. Pour cela, nous testons si cette fusion va entraîner la déconnexion de la face, en parcourant l'orbite β_1. Cette opération est en *O(nombre de brins de la face)*. De plus, contrairement au niveau précédent, une fusion peut ne pas être possible à un certain moment car elle entraîne une déconnexion, mais ne plus entraîner de déconnexion et donc être possible plus tard. Cela oblige, après une fusion de faces, à recommencer à tester toutes les arêtes de la carte. La complexité est alors quadratique en le nombre de brins de la carte. Lors de la construction de ce niveau, nous fusionnons uniquement des faces coplanaires, et donc pouvons effectuer exactement les mêmes fusions dans les 2-G-cartes de plongements. De plus, lorsqu'une fusion entraîne la déconnexion de la carte, nous ne l'effectuons pas afin de conserver une arête fictive. Dans ce cas, nous α_2-cousons les deux arêtes correspondantes dans la carte de plongement, ce qui crée une arête fictive dans celle-ci. Pour ce niveau, chaque face de la carte topologique possède une face de plongement identique.

Nous construisons ensuite la carte de niveau 3 à la ligne 4 de l'algorithme. Pour cela, nous fusionnons chaque couple d'arêtes adjacentes et alignées a_1 et a_2, incidentes à un sommet s tel qu'il n'existe pas d'arête réelle différente de a_1 et a_2 incidente à s. Nous parcourons chaque sommet de la carte, et cherchons s'il existe deux arêtes alignées incidentes au sommet courant s. Cette opération s'effectue en *O((nombre d'arêtes incidentes à s)^2)*. De plus, au cours de cette boucle autour du sommet, nous pouvons en même temps compter le nombre d'arêtes réelles incidentes à s[4]. À la fin de cette boucle, si nous avons trouvé deux arêtes réelles et que ce nombre est 2, nous pouvons effectuer la fusion. Mais si les deux arêtes sont fictives[5] et que ce nombre est 0, nous pouvons également effectuer la fusion. Dans ces deux cas, nous appelons tout d'abord l'al-

[4] Afin de tester en *O(1)* si une arête est réelle ou fictive, il suffit de marquer chaque arête fictive lors du calcul du niveau précédent ainsi que pour le calcul du niveau 4. En effet, c'est uniquement pour ces deux niveaux que nous pouvons créer des arêtes fictives.

[5] Rappelons que nous considérons que deux arêtes fictives sont toujours alignées.

gorithme décalant toutes les arêtes fictives incidentes à s, puis effectuons la fusion des deux arêtes de manière classique. Cet algorithme de décalage a une complexité en *O(nombre d'arêtes incidentes à s)*. La boucle principale qui parcours l'ensemble des sommets de la carte est linéaire en le nombre de brins de la carte. En effet, contrairement au niveau précédent, lorsque une fusion de deux arêtes n'est pas possible, elle ne le sera jamais, et nous n'avons donc pas besoin de tester à nouveau ce sommet. En majorant largement, nous obtenons une complexité quadratique en le nombre de brins de la carte. Nous n'avons pas besoin d'effectuer un calcul de complexité plus fin, étant donné que l'étape précédente est déjà en complexité quadratique. Les modifications du plongement sont, comme pour le niveau précédent, exactement les mêmes que celles de la carte. En effet, nous fusionnons les arêtes alignées et ces fusions peuvent donc être réalisées également dans les cartes de plongement.

La construction de la carte de niveau 4 (effectuée ligne 4 de l'algorithme) est très proche de celle du niveau 2. La seule différence entre ces deux niveaux porte sur le fait qu'au niveau 2 ces faces sont uniquement coplanaires alors que pour le niveau 4 non. Nous utilisons donc la même technique, avec les mêmes problèmes, et employons les mêmes solutions. Les deux complexités sont donc identiques. La seule différence se situe au niveau du plongement, où contrairement à la carte de niveau 3, les G-cartes de plongements ne sont pas fusionnées mais seulement α_2-cousues, afin de conserver chaque face de plongement plane.

Enfin, nous construisons la carte topologique à la ligne 6 de l'algorithme. Ici aussi, la construction de ce niveau est identique à celle du niveau 3, exception faite de la géométrie. Mais la géométrie n'entre en jeu ni pour la technique de construction, ni pour la complexité. Nous effectuons un test supplémentaire afin de déterminer si une arête est une boucle ou non, mais ce test peut s'effectuer simplement en *O(nombre d'arêtes incidentes à s)*. Nous obtenons alors exactement la même complexité que celle du calcul du niveau 3. Nous avons maintenant construit la carte de niveau n représentant l'image I. Nous devons construire l'arbre d'inclusion des régions qui permet de résoudre le problème de déconnexion de volumes.

5.5.3 Calcul de l'arbre d'inclusion

Cet algorithme 22 est quasiment le même que celui présenté en dimension 2 (section 4.4.3).

La seule différence entre ces deux algorithmes se situe lors du calcul de *parent*, la région contenant la composante connexe en cours de traitement. En effet, nous récupérons ici la région du brin β_3 cousu au brin représentant de la région en cours de traitement, alors que c'était le brin β_2 cousu en dimension 2. Le principe de l'algorithme est donc le même : parcourir les régions de l'image, dans l'ordre haut-bas, derrière-devant et droite-gauche, et chaque fois que nous trouvons une région non traitée parcourir la composante connexe incidente à cette région. Toutes les régions de cette composante connexe sont incluses dans la région du brin β_3 cousu au brin représentant de la région en cours de traitement, de par la contrainte que nous fixons sur ce brin représentant.

En effet, pour que cet algorithme fonctionne, nous devons vérifier trois contraintes, de manière similaire à la dimension 2 :

- chaque brin doit connaître sa région d'appartenance ;
- chaque région r connaît un des brins de la carte associée à cette région. Nous appelons un tel brin le *représentant* de sa région. Ce brin est soit le brin ayant comme plongement sommet

Algorithme 22 Calcul de l'arbre d'inclusion en 3d

Entrée : La carte C d'une image I segmentée en régions
La liste L des régions de l'image I
Sortie : L'arbre d'inclusion $\mathcal{A}$ correspondant.

```
Démarquer chaque région de L;
Ajouter un nœud associé à R_0 dans A;
1 pour chaque région R_i de L non marquée faire
    b_init ← représentant(R_i);
    parent ← le nœud associé à région(β_3(b_init));
2   pour chaque brin b de la composante connexe incidente à b_init faire
      si région(b) non marquée alors
        Ajouter un nœud associé à région(b) dans A;
        Ajouter ce nœud comme fils de parent dans A;
        Marquer région(b);
retourner A
```

le pointel le plus en haut à gauche et derrière de r, soit, si un tel brin n'existe pas, le brin ayant ce même pointel dans son plongement face ;

– nous connaissons la liste de toutes les régions de l'image, sans la région infinie. Cette liste doit être ordonnée de sorte qu'une région R_i est inférieure à une autre région R_j si et seulement si R_i se rencontre avant R_j lors du parcours de l'image de haut en bas, de derrière à devant et de gauche à droite.

Ces trois propriétés sont assurées par nos algorithmes d'extraction. La première lors de la construction de la carte de niveau 0, la deuxième également lors de cette construction, puis en nous assurant de la conserver lors de chaque fusion, et enfin la dernière par un parcours préalable de l'image, en classant les régions rencontrées selon cet ordre. Ce sont ces propriétés qui nous assurent que la région contenant la composante connexe en cours de traitement est bien la région *parent*. Comme pour la dimension 2, cet algorithme est linéaire en le nombre de brins de l'image. De plus, il fonctionne de manière identique, quel que soit le niveau de la carte lui étant fournie, à condition qu'elle respecte les trois contraintes citées.

Nous avons présenté de manière détaillée chaque étape de l'algorithme naïf d'extraction d'une carte à partir d'une image. La complexité globale de cet algorithme est finalement quadratique en le nombre de brins de la carte. Cela peut être gênant pour traiter de grosses images, qui vont être représentées par beaucoup de brins. Mais il est possible d'améliorer cette complexité en cherchant un ordre sur les fusions de faces, afin d'éviter de tester à nouveau chaque brin après chaque fusion. Nous verrons que ce point fait partie d'une de nos perspectives de recherche. Mais cette complexité importante n'est finalement pas très gênante, étant donné que cet algorithme est uniquement l'algorithme naïf, et que nous présentons maintenant l'algorithme optimal qui a une complexité linéaire.

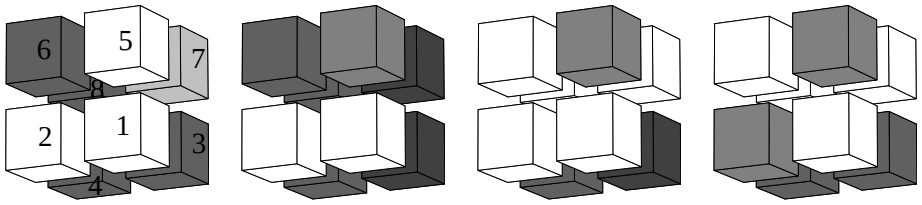


FIG. 5.49 – Quatre exemples de précodes en 3d.

5.6 Les précodes et l'algorithme optimal d'extraction

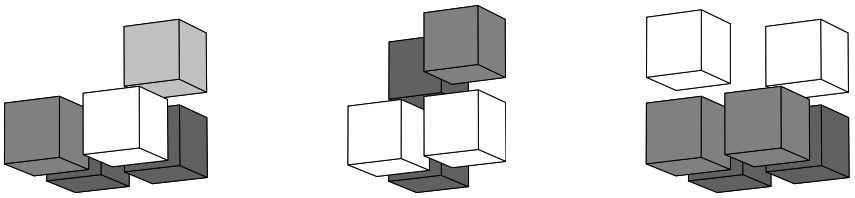
L'algorithme optimal d'extraction est l'extension à la dimension 3 de celui présenté en dimension 2. Il effectue un balayage de l'image de haut en bas, de derrière à devant et de gauche à droite avec une fenêtre de $2 \times 2 \times 2$ voxels, et exécute le traitement correspondant à la configuration de cette fenêtre. Nous étudions les configurations à traiter afin d'extraire chaque niveau de simplification. Pour chaque niveau, nous devons traiter des nouvelles configurations en plus de celles du niveau précédent. Cette manière de définir les précodes permet de factoriser les configurations se traitant de manière similaire. Nous revenons tout d'abord sur la notion de précode et de précode partiel en dimension 3.

5.6.1 Les précodes en 3d

Nous avons déjà présenté les notions de précode et précode partiel section 4.5.1, et ce en dimension n. En dimension 3, nous balayons l'image de haut en bas, de derrière à devant et de gauche à droite. Au cours de ce balayage, nous appelons *voxel courant* le voxel en cours de traitement (celui en bas, devant et à droite de la fenêtre courante).

Un *précode* en dimension 3 est une partition de l'ensemble des 8 voxels de l'hypercube de longueur 2. Nous pouvons voir figure 5.49 quatre exemples de précodes 3d. Comme pour la dimension 2, nous représentons un précode en affectant une couleur différente à chaque élément de la partition. Deux voxels de même couleur appartiennent à la même région et réciproquement. Si nous numérotons les voxels de la manière présentée figure 5.49.*a*, (le voxel courant a le numéro 1), ces quatre précodes sont équivalents respectivement aux partitions suivantes : $\{\{1,2,5\},\{3,4,6,8\},\{7\}\}$, $\{\{1,2\},\{3,7\},\{5\},\{4,6,8\}\}$, $\{\{1,2,6,7,8\},\{3\},\{4\},\{5\}\}$ et $\{\{1,6,7,8\},\{2,5\},\{3,4\}\}$.

Un précode 3d est *non variété* si et seulement s'il existe dans ce précode une partition n'étant pas 6-connexe. Les trois premiers précodes de la figure 5.49 sont des précodes variété, alors que le quatrième est un précode non variété. Un précode *partiel* est une partition d'un sous-ensemble des cellules d'un précode, chaque ensemble de la partition devant être un ensemble 6-connexe. Nous pouvons voir figure 5.50 deux exemples de précodes partiels en dimension 3 (*a* et *b*) ainsi qu'un contre-exemple de précode partiel (*c*). La partition associée au précode présenté figure 4.17.*c* n'est pas un précode partiel car un ensemble de voxels appartenant à la même région n'est pas 6-connexe.



a. b. c.

FIG. 5.50 – Deux exemples de précodes partiels (a et b) et un contre-exemple (c).

Nous avons vu lors de la présentation des précodes, section 4.5.1, que les nombres de Stirling et de Bell[6] permettent de calculer le nombre de précodes en dimension n. Cette formule nous a permis de calculer qu'il existait 15 précodes en dimension 2, ce qui est facilement vérifiable de manière exhaustive. Cette même formule nous donne que le nombre de précodes en dimension 3 est égal à B_8 qui vaut 4140. Il existe donc beaucoup trop de précodes pour en donner la liste exhaustive. Un autre nombre intéressant, est le nombre de précodes variétés. En dimension 2, nous l'avons calculé en regardant pour chacun des 15 précodes lesquels étaient des variétés. Mais cette solution n'est pas applicable ici. Nous avons dans un premier temps cherché une formule de dénombrement permettant de calculer ce nombre en dimension n, malheureusement sans succès pour le moment. Ce point fait partie de nos perspectives de recherches.

Afin de palier l'absence de formule, nous avons calculé ce nombre par programme. Nous avons pour cela généré les précodes de manière automatique, et testé ensuite chaque précode. Nous utilisons pour cela le graphe représentant l'hypercube de dimension n. Chaque sommet représente une n-cellule et deux sommets sont reliés par une arête si les deux n-cellules correspondantes sont $2n$-voisins. Chaque sommet est étiqueté par la région à laquelle il appartient. Il suffit ensuite de tester si chaque ensemble de sommets ayant la même étiquette est bien un ensemble connexe. Ce programme nous a permis de trouver qu'il existe 958 précodes variétés en dimension 3. Ce nombre est petit en regard des 4140 précodes existant au total. Mais il est encore trop important pour envisager l'écriture de chaque méthode de traitement correspondant. Nous allons voir que nous n'avons pas besoin de traiter l'ensemble de ces 958 cas, car un grand nombre d'entre eux se traitent de manière similaire. C'est ici que la définition de la carte topologique de manière hiérarchique apporte pleinement son intérêt.

5.6.2 L'algorithme optimal et général d'extraction

L'algorithme 23 général et optimal d'extraction est simple, comme en dimension 2, puisqu'il se résume par un balayage de l'image en exécutant le code associé au précode courant. Comme pour la dimension 2, cet algorithme fonctionne pour tout type d'image : segmentées ou non, étiquetage fort ou faible.

L'image I en entrée de cet algorithme doit être segmentée en régions. Elle est composée des voxels de coordonnées $1 \dots n_1$ en x, de $1 \dots n_2$ en y et de $1 \dots n_3$ en z. Elle est entourée d'une région infinie contenant l'ensemble des voxels n'appartenant pas à l'image. Le précode (i, j, k)

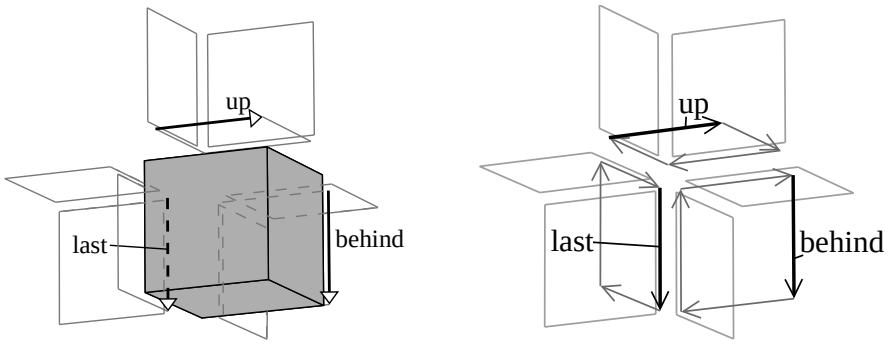
[6] Les nombres de Bell sont donnés par la formule de récurrence $B_l = \sum_{i=0}^{l-1} \frac{B_i . l!}{i!(l-i)!}$ avec $B_0 = B_1 = 1$.

Algorithme 23 Extraction optimale de la carte de niveau n en 3d

Entrée : Une image I segmentée en régions de $n_1 \times n_2 \times n_3$ voxels
n le niveau de la carte que l'on veut extraire

Sortie : La carte de niveau n de l'image I.

```
1 last ← Construire le bord supérieur de la carte;
2 pour k = 1 à n_3 + 1 faire
      pour j = 1 à n_2 + 1 faire
          pour i = 1 à n_1 + 1 faire
              last ← Exécuter le code associé au précode (i, j, k) du niveau n;
3 Calculer l'arbre d'inclusion des régions;
```



a. Avec le voxel courant. b. Et sans.

FIG. 5.51 – Les brins *last*, *up* et *behind* par rapport au voxel courant.

est le précode ayant comme voxel courant (celui en bas, devant et à droite de la fenêtre $2 \times 2 \times 2$) le voxel (i, j, k).

Avant de balayer l'image, nous commençons par créer le bord supérieur de la carte. En effet, comme pour la dimension 2, la carte correspondant aux voxels plus anciens pour l'ordre de parcours doit être déjà construite. Cet invariant garantit, à tout instant, qu'il existe une face à gauche, derrière et au-dessus du voxel courant. Ces trois faces permettent de raccrocher le voxel courant à la carte déjà construite. Afin de désigner ces faces, nous conservons un brin particulier sur chacune, que nous appelons respectivement *last*, *up* et *behind* (ou l, u et b en abrégé). Nous pouvons voir figure 5.51 la position de ces brins particuliers, tout d'abord par rapport au voxel courant, puis sans ce voxel afin de mieux voir leurs positions. Sur ces figures, nous n'avons pas représenté l'ensemble des orientations des brins afin de les rendre plus lisibles, mais elles peuvent se déduire aisément à partir des seules orientations représentées. La position de ces brins à l'intérieur des faces a été choisie de manière arbitraire. Nous devons simplement fixer cette position qui est primordiale lors des algorithmes associés aux précodes.

Avant de parcourir l'image, le voxel courant (qui est le prochain voxel à traiter) est celui ayant comme coordonnées $(1, 1, 1)$. Afin que notre invariant soit conservé, nous devons avant de traiter ce voxel, avoir construit la carte des voxels plus anciens, c'est-à-dire le voxel à gauche $(0, 1, 1)$,

les voxels derrière $(i, 0, 1)$, $\forall i$ et les voxels dessus $(i, j, 0)$, $\forall i, j$. C'est cette carte qui est appelée bord supérieur de l'image. De plus, comme pour la dimension 2, nous créons ce bord de manière particulière afin de nous affranchir des problèmes liés aux précodes de bord de l'image. Nous verrons section 5.6.8 comment ce bord est construit afin de régler ces problèmes, et comment il intervient lors de l'extraction d'une carte.

Lors du balayage de l'image, nous conservons uniquement le brin *last*, car les brins *up* et *behind* peuvent se retrouver aisément à partir de ce brin à l'aide des algorithmes 24 et 25.

Algorithme 24 Calcul du brin *up* en 3d.

Entrée : *last* le brin à gauche du voxel courant
Sortie : *up* le brin au-dessus du voxel courant.

$up \leftarrow \beta_{02}(last)$;
tant que up *est* β_3*-cousu* **faire**
  $up \leftarrow \beta_{32}(up))$
$up \leftarrow \beta_1(up)$;
retourner up;

Algorithme 25 Calcul du brin *behind* en 3d.

Entrée : *last* le brin à gauche du voxel courant
Sortie : *behind* le brin derrière le voxel courant.

$behind \leftarrow \beta_2(last)$;
tant que behind *est* β_3*-cousu* **faire**
  $behind \leftarrow \beta_{32}(up))$
$behind \leftarrow \beta_{11}(behind)$;
retourner behind;

Ces deux algorithmes fonctionnent suivant le même principe. L'algorithme 24 commence par initialiser le brin *up* par $\beta_{02}(last)$. Ce brin appartient à l'arête incidente aux deux faces contenant *last* et *up*. Nous tournons ensuite autour de cette arête, en utilisant β_{32} tant que le brin courant est β_3-cousu. En effet, toutes les faces entre la face contenant *last* et celle contenant *up* sont β_3-cousus. Lorsque le brin courant n'est pas β_3-cousu, c'est qu'il appartient à la face contenant *up*, et il suffit alors d'appliquer β_1 pour trouver le brin *up*. L'algorithme 25 effectue le même traitement mais pour un brin initial différent, ainsi que pour le « déplacement » final.

La complexité de ces deux algorithmes est linéaire en le nombre de faces incidentes à l'arête entre la face contenant *last* et celle contenant *up* pour le premier, et la face contenant *last* et celle contenant *behind* pour le deuxième. Ce nombre de faces est toujours égal à 0, 1 ou 2, étant donné que chaque face représente un surfel. Nous pouvons voir figure 5.52 les quatre seules configurations possibles des faces entre celle contenant *last* et celle contenant *up*. Sur cette figure, nous avons dessiné deux brins de même couleur lorsqu'ils appartiennent à la même région. Nous n'avons pas représenté tous les brins afin de ne pas surcharger les schémas. Nous voyons donc que, dans le pire des cas présenté figure 5.52.*a*, l'algorithme calculant le brin *up* va effectuer uniquement 2 itérations. Dans le meilleur des cas présenté figure 5.52.*d*, ce même algorithme ne va pas entrer dans cette boucle, car dès l'initialisation le brin *up* est β_3-libre. Il en est exactement de

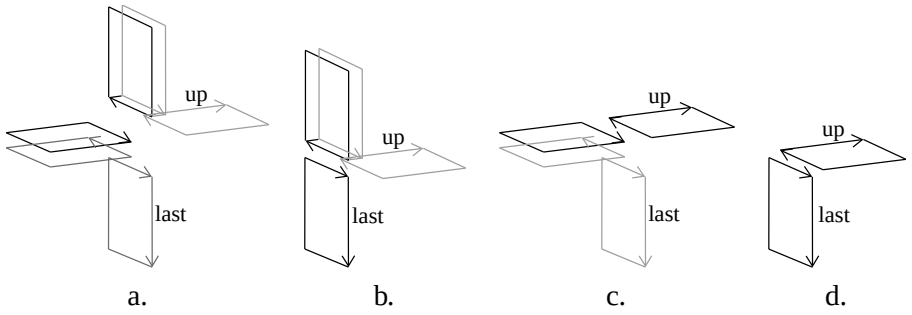


FIG. 5.52 – Les quatre seules configurations possible des faces entre celle contenant *last* et celle contenant *up*.

même pour le calcul du brin *behind* et les configurations possibles de faces entre celle contenant *last* et celle contenant *behind*.

Après avoir créé le bord supérieur de l'image dans l'algorithme 23, le brin *last* est initialisé afin d'être sur la face de gauche par rapport au voxel $(1, 1, 1)$. Ensuite, l'algorithme effectue le balayage de l'image en exécutant le traitement correspondant au précode courant. Chacun de ces précodes modifie localement la carte, en utilisant les trois brins distingués *last*, *up* et *behind*, et en retournant le prochain brin *last* pour le traitement du précode suivant. Nous verrons section 5.6.8 que de par la manière particulière dont le bord est construit, aucun traitement particulier n'est nécessaire pour les précodes se trouvant au bord de l'image, ce qui simplifie beaucoup l'algorithme.

Enfin, après le balayage de l'image, nous avons construit la carte de niveau n représentant l'image I. Il ne reste plus qu'à calculer l'arbre d'inclusion des régions afin de positionner les éventuelles différentes composantes connexes de la carte les unes par rapport aux autres. Pour cela, nous utilisons l'algorithme 22 déjà présenté section 5.5.3. Comme pour la dimension 2, le parcours préalable de l'image afin d'obtenir la liste des régions triées selon leur ordre d'apparition n'est plus nécessaire pour ce second algorithme d'extraction. En effet, cette liste peut maintenant être créée durant le même parcours que celui construisant la carte. Cet algorithme optimal d'extraction est générique car il permet d'extraire n'importe quel niveau de simplification. Pour cela, il faut définir pour chacun de ces niveaux les précodes à distinguer et la manière de les traiter.

5.6.3 Les précodes pour la carte lignel

Afin de calculer la carte lignel, nous effectuons la fusion de chaque couple de volumes adjacents et appartenant à la même région. De par notre ordre de parcours, il est uniquement possible de fusionner le voxel courant avec ses voisins gauche, derrière et dessus. Les différents cas à traiter sont simplement l'ensemble des combinaisons de fusion ou non du voxel courant avec ces trois voisins. Il y a donc 8 précodes différents à traiter, présentés figure 5.53 (que nous appelons l_i pour précode lignel numéro i). Le précode l_1 représente le cas où le voxel courant n'est fusionné avec aucun de ses voisins (C_3^0), les précodes l_2, l_3 et l_4 les cas où il est fusionné avec un seul de ses voisins (C_3^1), les précodes l_5, l_6 et l_7 les cas où il est fusionné avec deux de ses voisins (C_3^2) et enfin le précode l_8 le cas où il est fusionné avec ses trois voisins (C_3^3).

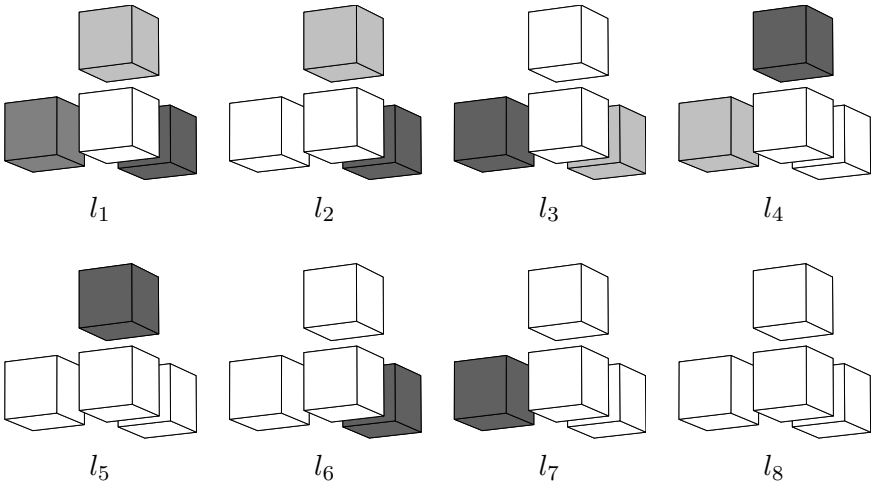


FIG. 5.53 – Les huit précodes à traiter afin de calculer la carte lignel en 3d.

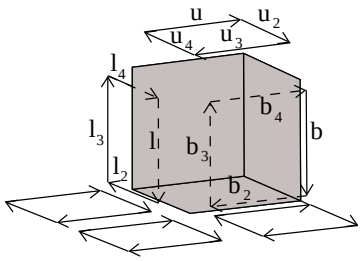
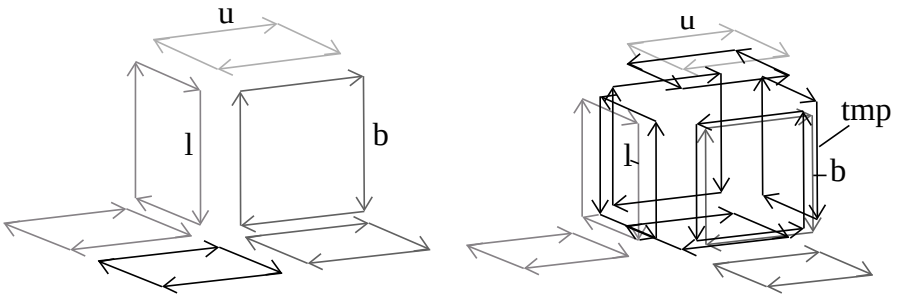


FIG. 5.54 – Les brins obligatoirement présents avant de traiter un précode de niveau 1.

La configuration de la carte avant le traitement de chacun de ces précodes est la même, et est présentée figure 5.54. En effet, nous savons qu'il existe les trois faces autour du voxel courant de par notre invariant. Ce sont les faces contenant les brins *last*, *up* et *behind*. Nous ne connaissons aucune autre face, présente obligatoirement avant le traitement d'un précode, car ces huit précodes sont des précodes partiels et nous ne connaissons pas la région des voxels libres. Nous avons également représenté trois autres faces entre les voxels (i, j, k) et $(i, j, k+1)$, mais ces faces n'entrent pas en compte pour les différents traitements, elles permettent seulement de mieux visualiser la figure. Afin de simplifier la désignation des brins, nous notons $l_2 = \beta_1(l)$, $l_3 = \beta_1(l_2)$ et $l_4 = \beta_1(l_3)$, les trois autres brins de la face contenant le brin *last*. Nous notons de manière similaire les autres brins appartenant aux faces contenant les brins *up* et *behind*.

Comme pour la dimension 2, le traitement d'un précode consiste à donner la suite d'opérations transformant la carte courante avant de traiter ce précode en la carte à obtenir après ce traitement. Ces opérations simulent les fusions correspondant à ce précode. De par le nombre important de précodes différents, nous ne donnons pas ici, contrairement à la dimension 2, chaque algorithme de traitement. La figure 5.55.*a* montre la carte courante avant de traiter le précode l_1, et la figure 5.55.*b* celle à obtenir. L'algorithme 26 permettant de passer de la première carte à la seconde est simple,



a. Carte avant le traitement. b. Le résultat.

FIG. 5.55 – Le précode l_1 avant et après son traitement.

étant donné qu'il consiste simplement à créer un cube, et à β_3-coudre sa face de gauche à la face contenant *last*, sa face de dessus à la face contenant *up* et sa face de derrière à la face contenant *behind*.

Algorithme 26 Code associé au précode l_1 en 3d

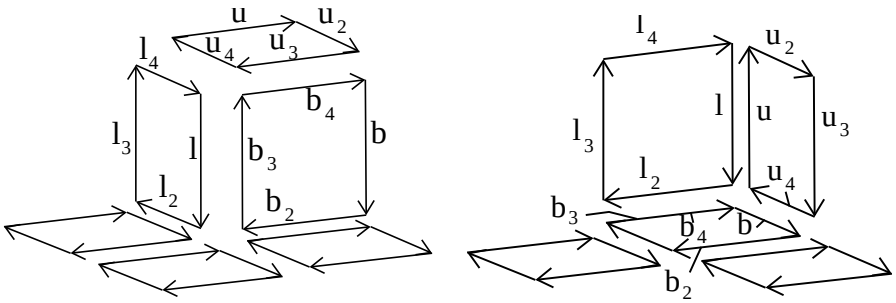
Entrée : *last*, *up* et *behind*
(x, y, z) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$tmp \leftarrow$ créer un cube topologique ;
β_3-coudre la face contenant *last* et celle contenant $\beta_{2112}(tmp)$;
β_3-coudre la face contenant *up* et celle contenant $\beta_{021}(tmp)$;
β_3-coudre la face contenant *behind* et celle contenant $\beta_2(tmp)$;
Le plongement du sommet incident à $\beta_{11}(tmp) \leftarrow (x, y, z)$;
retourner tmp;

Pour le plongement, nous associons simplement les coordonnées du nouveau sommet au plongement sommet du brin $\beta_{11}(tmp)$. En effet, c'est le seul sommet n'ayant pas de plongement, tout les autres sommets appartiennent à une orbite possédant déjà un plongement dans la carte avant le traitement. Suivant le plongement utilisé cette mise à jour du plongement se fera de manière différente. Mais la majorité des mises à jour de plongement sont faites lors des opérations de couture. Par exemple pour un plongement face, lors d'une β_3-couture de deux faces entre elles, celle n'ayant pas de plongement pointera désormais sur le plongement de la face qui en possède un.

La figure 5.56.*a* montre la configuration de la carte avant de traiter le précode l_8 et la figure 5.56.*b* celle à obtenir. L'algorithme 27 effectue la transformation de la première carte en la seconde. Cet algorithme présente une manière d'effectuer cette transformation, mais il est possible d'effectuer le même traitement de manière différente. Nous avons ici privilégié la conservation des brins de la carte initiale, afin d'éviter des allocations et désallocations mémoire successives coûteuses en temps d'exécution. De plus, nous avons essayé d'être minimal en nombre d'opérations. Par exemple, nous conservons les brins l_3, b_2 et u_2 à leurs places initiales, ce qui évite une β_2-décousure et une β_2-couture pour chacun de ces brins.

La première modification effectuée par cet algorithme (ligne 1) permet de traiter la configuration présentée figure 5.57. Ce cas se produit lorsque, comme nous l'avons représenté figure 5.57.*a*,



a. Carte avant le traitement. b. Le résultat.

FIG. 5.56 – Le précode l_8 avant et après son traitement.

Algorithme 27 Code associé au précode l_8 en 3d

Entrée : *last, up* et *behind*
(x, y, z) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

1 $t_1 \leftarrow \beta_2(l_4)$; $t_2 \leftarrow \beta_2(u_4)$; β_2-découdre(l_4) ;
si $t_1 \neq u_4$ **alors**
  β_2-découdre(u_4) ; β_2-coudre(t_1, t_2) ;

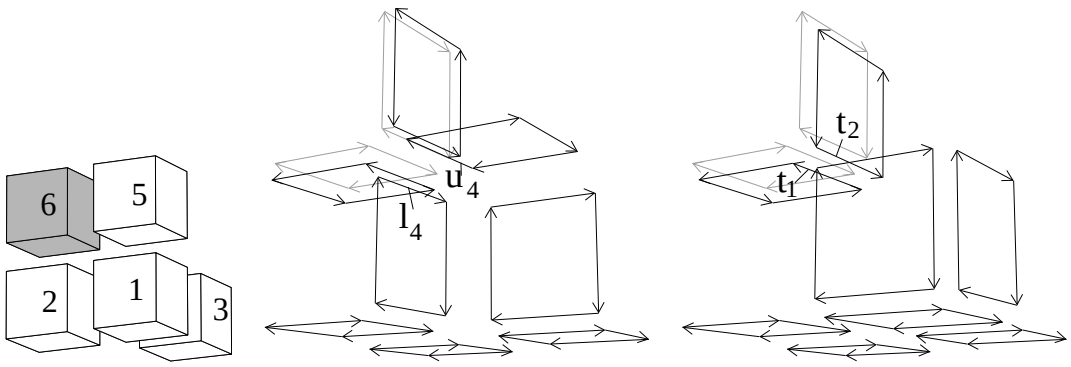
2 $t_1 \leftarrow \beta_2(l)$; $t_2 \leftarrow \beta_2(b_3)$; β_2-découdre(l) ;
si $t_1 \neq b_3$ **alors**
  β_2-découdre(b_3) ; β_2-coudre(t_1, t_2) ;

3 $t_1 \leftarrow \beta_2(u_3)$; $t_2 \leftarrow \beta_2(b_4)$; β_2-découdre(u_3) ;
si $t_1 \neq b_4$ **alors**
  β_2-découdre(b_4) ; β_2-coudre(t_1, t_2) ;

4 $t_1 \leftarrow \beta_2(l_2)$; β_2-découdre(l_2) ; β_2-coudre(t_1, b_3);
5 $t_1 \leftarrow \beta_2(b)$; β_2-découdre(b) ; β_2-coudre(t_1, u_3);
6 $t_1 \leftarrow \beta_2(u)$; β_2-découdre(u) ; β_2-coudre(t_1, l_4);
7 β_2-coudre(l, u) ; β_2-coudre(l_2, b_4) ; β_2-coudre(u_4, b) ;
Le plongement du sommet incident à $u \leftarrow (x, y, z)$;
retourner u_3;

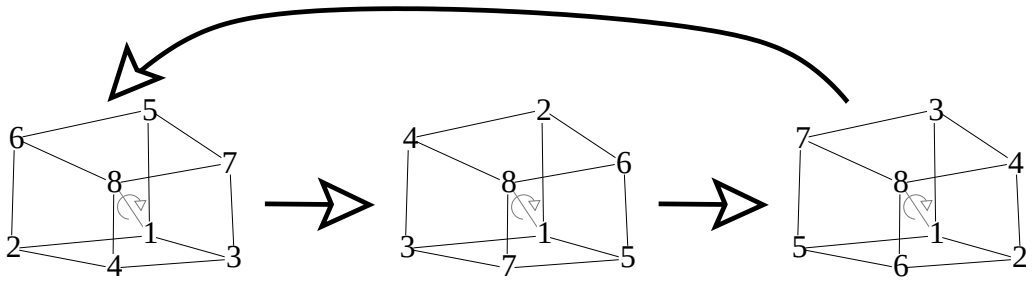
le voxel numéro 6 n'est pas de la même région que les voxels 1, 2 et 5. La carte courante avant de traiter ce précode est présentée figure 5.57.*b*, où nous pouvons noter la présence d'une face entre les voxels 2 et 6, et une autre entre les voxels 5 et 6. Dans ce cas, $\beta_2(l_4) \neq u_4$ et nous devons donc β_2-découdre les deux brins l_4 et u_4 puis β_2-coudre les brins qui étaient β_2-cousus à eux (que nous avons conservé dans t_1 et t_2). Par contre, si $\beta_2(l_4) = u_4$, alors c'est que le voxel 6 est de la même région que les voxels 1, 2 et 5, et dans ce cas nous devons simplement β_2-découdre l_4. Ce traitement est similaire pour traiter les deux cas où les voxels numéro 4 et 7 ne sont pas de la même région que leurs trois voisins (lignes 2 et 3 de l'algorithme).

Après avoir traité ces trois cas possibles, les lignes 4, 5 et 6 effectuent le remplacement respectivement du brin l_2 par b_3, du brin b par u_3 et du brin u par l_4. Enfin, la ligne 7 effectue les β_2-coutures qui n'ont pas encore été effectuées. Notons que tous les brins traités ici ont déjà été β_2-décousus. Les plongements divers vont être mis à jour de manière transparente lors des cou-



a. Le cas à traiter. b. Carte avant le traitement. c. Le résultat.

FIG. 5.57 – Le cas traité par la ligne 1 de l'algorithme 27 lorsque le voxel 6 n'est pas de la même région que ses trois voisins dans le précode l_8.



a. Un précode p_1. b. p_2 =rotation(p_1). c. p_3 =rotation(p_2).

FIG. 5.58 – La permutation *rotation* qui transforme un précode.

tures et décousures ; le seul que nous devons affecter explicitement est celui du sommet incident au brin u. En effet, c'est le seul sommet topologique qui n'existait pas dans la carte courante avant de traiter ce précode.

Nous pouvons donner pour les précodes l_2 à l_7 chaque algorithme de traitement correspondant. Ces algorithmes vont ressembler aux deux que nous venons de présenter. Mais il n'est pas nécessaire de définir un algorithme par précode. Nous regroupons ces traitements pour les précodes *isomorphes par rotation*, ce qui réduit le nombre d'algorithmes à écrire. Afin de savoir si deux précodes sont isomorphes par rotation, nous définissons pour cela la permutation appelée *rotation* sur l'ensemble $1, \ldots, 8$. Cette fonction associe à un numéro de voxel son image par rotation, de la manière présentée figure 5.58. La figure 5.58.*a* représente un précode en numérotant chacun de ses huit voxels, ainsi qu'en symbolisant les relations de 6-voisinage afin de faciliter la visualisation. La figure 5.58.*b* montre l'image de ce précode par application de *rotation*, et la figure 5.58.*c* l'image de ce deuxième précode. Nous définissons explicitement cette permutation en donnant l'image de chaque élément au moyen du tableau 5.1. Nous pouvons remarquer que lorsque nous appliquons trois fois cette fonction sur n'importe quel précode, nous obtenons toujours le précode initial. Remarquons également que les deux voxels numéro 1 et 8 sont invariants par rotation. En effet, cette

TAB. 5.1 – La permutation *rotation*.

x	1	2	3	4	5	6	7	8
rotation(x)	1	5	2	6	3	7	4	8

rotation peut être vue comme la rotation de 120 degrés autour de l'axe allant du voxel 1 au voxel 8, et il est donc normal que ces deux voxels soient invariants. Deux précodes sont *isomorphes par rotation* si à partir d'un précode nous pouvons obtenir le deuxième en appliquant *rotation* autant de fois que nécessaire.

L'ensemble des précodes isomorphes par rotation peuvent être traités avec un seul algorithme appelé avec des brins initiaux différents. Cela nous permet de diminuer le nombre de traitements à écrire. Par exemple pour les précodes lignels, les précodes l_2, l_3 et l_4 sont isomorphes par rotation. Pour s'en convaincre, il suffit d'écrire la partition correspondant au précode l_2 et d'appliquer ensuite *rotation*. La partition correspondant au précode l_2 est $\{\{1,2\},\{3\},\{5\}\}$. Le résultat de *rotation* sur ce précode est la partition $\{\{1,5\},\{2\},\{3\}\}$, obtenue en remplaçant chaque élément par son image. Cette partition est celle correspondant au précode l_3. Le résultat d'une nouvelle application de *rotation* est la partition $\{\{1,3\},\{5\},\{2\}\}$ correspondant au précode l_4. Nous pouvons donc définir un seul algorithme de traitement pour ces trois précodes, qui est présenté algorithme 28.

Algorithme 28 UneFusionVolumes3d

Entrée : d_1, f_1 et f_2 trois brins
(x,y,z) les coordonnées du nouveau sommet

Sortie : le « prochain » *last*.

$d_2 \leftarrow \beta_1(d_1)$; $d_3 \leftarrow \beta_1(d_2)$; $d_4 \leftarrow \beta_1(d_3)$;
β_1-découdre(d_1) ; β_1-découdre(d_2) ; β_1-découdre(d_3) ; β_1-découdre(d_4) ;

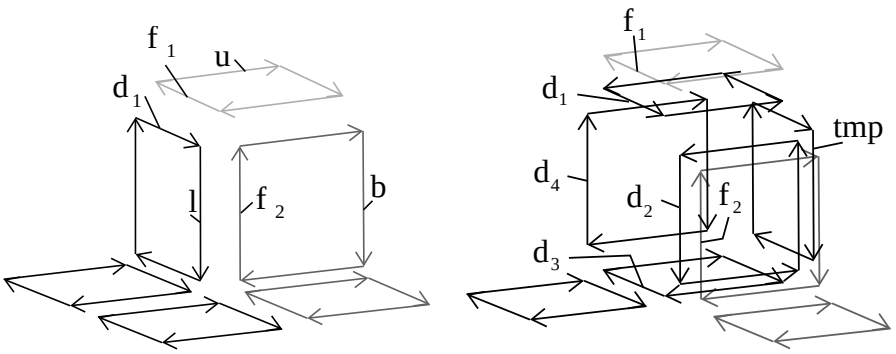
créer une face carrée autour de d_1 ; créer une face carrée autour de d_2 ;
créer une face carrée autour de d_3 ; créer une face carrée autour de d_4 ;
$tmp \leftarrow$ créer une face carrée topologique ;

β_2-coudre$(\beta_1(d_1),\beta_0(d_2))$; β_2-coudre$(\beta_{11}(d_1),\beta_0(tmp))$;
β_2-coudre$(\beta_0(d_1),\beta_1(d_4))$; β_2-coudre$(\beta_1(d_3),\beta_0(d_4))$;
β_2-coudre$(\beta_1 1(d_3),\beta_1(tmp))$; β_2-coudre$(\beta_0(d_3),\beta_1(d_2))$;
β_2-coudre$(tmp,\beta_{11}(d_2))$; β_2-coudre$(\beta_{11}(tmp),\beta_{11}(d_4))$;

β_3-coudre la face contenant f_1 et celle contenant d_1 ;
β_3-coudre la face contenant f_2 et celle contenant d_2 ;

Le plongement du sommet incident à $\beta_{11}(tmp) \leftarrow (x,y,z)$;
retourner tmp;

Cet algorithme prend en entrée trois brins d_1, f_1 et f_2 et modifie la carte courante en fusionnant le voxel courant avec celui incident à d_1 et en le β_3-cousant aux deux faces f_1 et f_2. Nous commençons par nommer les trois autres brins de la face incidente à d_1, puis β_1-décousons ces quatre brins. Nous créons ensuite quatre faces carrées autour de chacun de ces brins. Nous créons également une nouvelle face carrée supplémentaire. Le traitement suivant consiste simplement à β_2-coudre ensemble chacune de ces faces. À part les brins d_1 à d_4 qui étaient initialement β_2-



a. Carte avant le traitement. b. Le résultat.

FIG. 5.59 – La carte avant et après le traitement du précode l_2.

cousus, et dont nous conservons les coutures, les autres brins viennent tous d'être créés et sont donc β_2-libres. Enfin, le dernier traitement de cet algorithme consiste à β_3-coudre les deux faces incidentes aux deux brins f_1 et f_3 avec leurs faces respectives en vis-à-vis. Nous devons affecter le plongement du sommet incident au brin $\beta_{11}(tmp)$ car il appartient au nouveau sommet topologique. Les autres plongements sont mis à jour de manière automatique, principalement lors des deux β_3-coutures. Le résultat de cet algorithme pour les trois traitements des précodes l_2, l_3 et l_4 est donné respectivement sur les figures 5.59, 5.60 et 5.61.

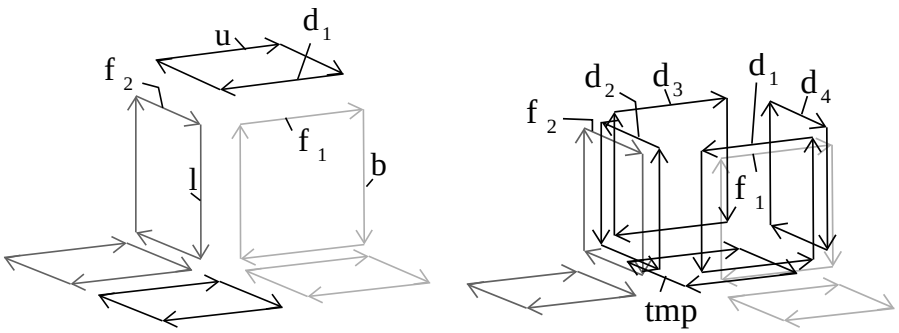
Algorithme 29 Code associé au précode l_2 en 3d

Entrée : *last*, *up* et *behind*
$c = (x, y, z)$ les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$d_1 \leftarrow \beta_0(last)$; $f_1 \leftarrow \beta_0(up)$; $f_2 \leftarrow \beta_{11}(behind)$;
$tmp \leftarrow$ UneFusionVolumes3d(d_1, f_1, f_2, c);
retourner tmp;

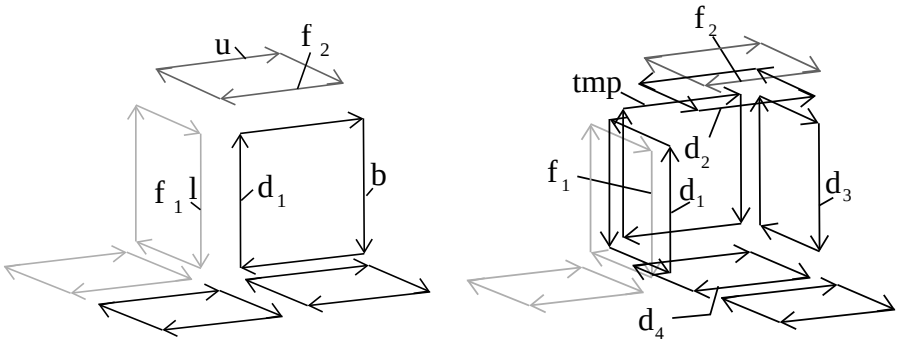
Maintenant que nous avons défini cet algorithme, nous devons l'appeler avec les brins initiaux correspondant au précode effectif. L'algorithme 29 montre comment cet algorithme est appelé pour le traitement du précode l_2. 29. Cet algorithme consiste simplement à appeler *UneFusionVolumes3d* et à retourner le prochain brin *last* pour le traitement du précode suivant. La figure 5.59.*a* montre la position des brins initiaux passés à l'algorithme *UneFusionVolumes3d*. La carte obtenue après cet algorithme est présentée figure 5.59.*b*.

Le traitement du précode l_3 est identique, à l'exception de la position des brins initiaux et du brin retourné. Nous pouvons voir figure 5.60.*a* la position de ces brins avant le traitement de ce précode, et figure 5.60.*b* le résultat de ce traitement, avec la position du brin retourné par l'algorithme *UneFusionVolumes3d*. Nous avons $d_1 = \beta_{11}(up)$, $f_1 = \beta_0(behind)$ et $f_2 = \beta_0(last)$. Le brin retourné par cet algorithme est maintenant $\beta_{1120}(tmp)$. Le traitement du précode l_4 est également identique à celui du précode l_2, comme nous pouvons le vérifier figure 5.61. Pour ce traitement, nous avons $d_1 = \beta_{11}(behind)$, $f_1 = last$ et $f_2 = \beta_{11}(up)$. Le brin retourné est $\beta_{1211}(tmp)$.



a. Carte avant le traitement. b. Le résultat.

FIG. 5.60 – La carte avant et après le traitement du précode l_3.

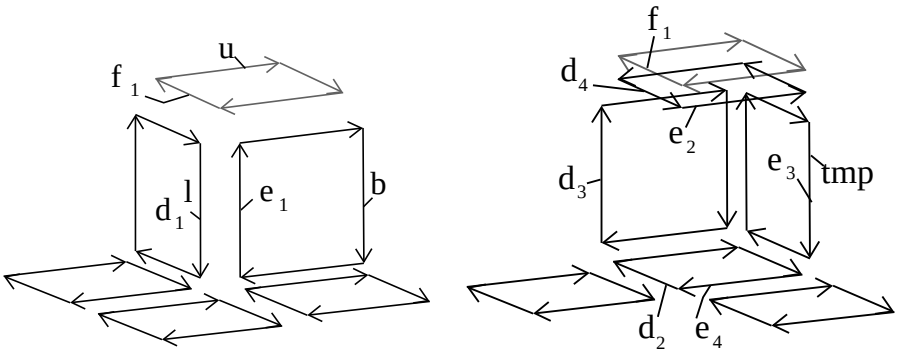


a. Carte avant le traitement. b. Le résultat.

FIG. 5.61 – La carte avant et après le traitement du précode l_4.

Remarquons que pour chacun de ces algorithmes, nous avons un brin sur chaque face incidente à *last*, *up* et *behind*. En effet, ces brins désignent les faces supprimées par une fusion de volumes ou conservées et β_3-cousues avec une face du volume représentant le voxel courant. De plus, afin de fixer la position de ces brins, il suffit de la choisir pour le premier précode que nous traitons, ici l_2, et d'écrire l'algorithme *UneFusionVolumes3d* pour ce précode. L'écriture des traitements des autres précodes se fait alors simplement en retrouvant l'image de chaque brin utilisé par le premier traitement par *rotation*.

Les trois précodes restant, l_5, l_6 et l_7, sont isomorphes par rotation, et nous pouvons donc également définir un seul traitement pour les trois. Cet algorithme, que nous appelons *DeuxFusionsVolumes3d* car le voxel courant est fusionné avec deux de ses voisins, prend en paramètre deux brins d_1 et e_1 appartenant aux deux voxels à fusionner avec le voxel courant, et un brin f_1 appartenant au voxel qui ne va pas être fusionné. Remarquons que pour cet algorithme, il faut un traitement similaire à celui effectué en début de l'algorithme 27 traitant le précode l_8 afin de coudre les éventuelles faces se trouvant entre d_1 et e_1. Nous ne donnons pas ici cet algorithme, qui peut s'écrire simplement en regardant la carte courante avant le traitement et celle à obtenir.



a. Carte avant le traitement. b. Le résultat.

FIG. 5.62 – La carte avant et après le traitement du précode l_5.

Nous allons maintenant voir comment cet algorithme est appelé pour traiter les trois précodes l_5, l_6 et l_7. Cela permet de bien comprendre les traitements de précodes isomorphes par rotation. De plus, nous montrons pour chacun de ces précodes la carte courante avant et après le traitement, avec la position de chaque brin, ce qui permet ensuite pour les personnes désireuses d'implanter l'algorithme optimal d'extraction d'écrire facilement la fonction correspondant à *DeuxFusionsVolumes3d*.

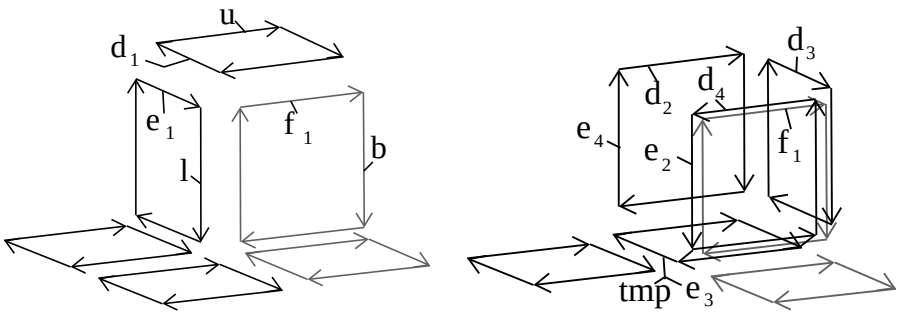
Algorithme 30 Code associé au précode l_5 en 3d

Entrée : *last*, *up* et *behind*
$c = (x, y, z)$ les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

$d_1 \leftarrow last$; $e_1 \leftarrow \beta_{11}(behind)$; $f_1 \leftarrow \beta_0(up)$;
$tmp \leftarrow$ DeuxFusionsVolumes3d(d_1, e_1, f_1, c);
retourner tmp;

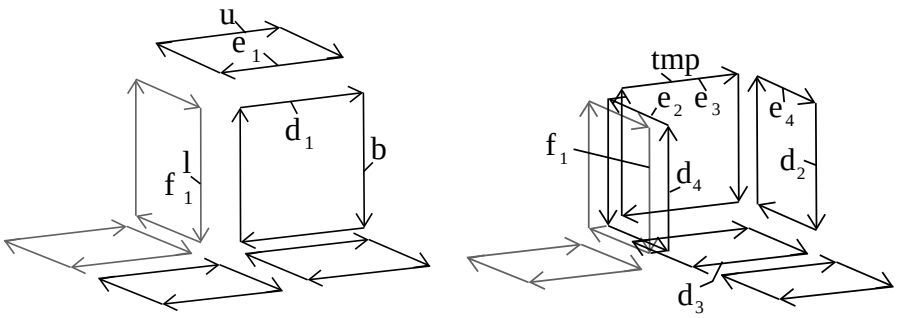
Nous présentons tout d'abord l'algorithme 30 traitant le précode l_5, et figure 5.62 la carte courante avant de traiter ce précode, et celle à obtenir après son traitement. Sur cette figure, nous pouvons observer la position des brins passés en paramètre à l'algorithme *DeuxFusionsVolumes3d*. d_1 et e_1 appartiennent aux deux faces qui doivent être fusionnées avec le voxel courant. Afin de faciliter la désignation d'un brin particulier, nous appelons d_2, d_3 et d_4 les brins appartenant à la même face que d_1, dans l'ordre donné par β_1, et de manière similaire les brins de la face incidente à e_1. Nous pouvons voir figure 5.62.*b* que nous avons conservé certains de ces brins afin d'éviter des destructions et des allocations de mémoire. De plus, nous conservons ces brins à leur place initiale, ils n'ont donc pas besoin d'être β_2-décousus puis β_2-cousus à un autre endroit. La face incidente à f est β_3-cousue à celle incidente à d_4, ce qui entraîne des mises à jour des plongements. Enfin, le seul sommet topologique qui n'existait pas dans la carte avant le traitement de ce précode est celui incident à $\beta_0(e_4)$ et nous lui affectons donc les coordonnées du nouveau sommet passé en paramètre à l'algorithme. Enfin le brin retourné est tmp qui est le brin retourné par l'algorithme *DeuxFusionsVolumes3d*.

Le traitement du précode l_6 est identique à celui du précode l_5, à l'exception de la position des brins initiaux et du brin retourné. Comme nous pouvons voir figure 5.63, nous avons $d_1 =$



a. Carte avant le traitement. b. Le résultat.

FIG. 5.63 – La carte avant et après le traitement du précode l_6.



a. Carte avant le traitement. b. Le résultat.

FIG. 5.64 – La carte avant et après le traitement du précode l_7.

$\beta_0(up)$, $e_1 = \beta_0(last)$, $f_1 = \beta_0(behind)$ et le brin retrourné qui est $\beta_{1120}(tmp)$. De même pour le traitement du précode l_7, où comme nous pouvons vérifier figure 5.64 nous avons $d_1 = \beta_0(behind)$, $e_1 = \beta_{11}(up)$, $f_1 = last$ et le brin retourné qui est $\beta_{1211}(tmp)$.

Nous avons présenté le traitement de chaque précode lignel. L'algorithme optimal d'extraction va simplement tester quel est parmi ces huit précodes, le précode courant, et exécuter le traitement correspondant qui modifie localement la carte. Cet algorithme est linéaire, car chaque traitement est en *O(1)*, et chaque voxel de l'image est parcouru exactement une fois. De plus, nous avons vu pour ces divers traitements que nous créons exactement le nombre de brins nécessaires, sans effectuer de destruction ou d'allocation inutile.

Nous pouvons montrer que ces huit précodes lignels couvrent bien l'ensemble des précodes existant en dimension 3. Mais contrairement à la dimension 2, nous ne pouvons pas donner ici pour chaque précode la liste des précodes qu'il couvre, étant donné leur nombre beaucoup trop important. Nous allons dénombrer les configurations couvertes par chaque précode, et vérifier que c'est bien le nombre de configurations possibles. Une configuration (notion présentée section 4.5.1) s'obtient à partir d'un précode en affectant à chaque élément de la partition un identifiant de région, que nous appelons ici abusivement couleur. Il est plus facile de compter les configurations que les précodes, étant donné que deux configurations sont différentes si elles n'ont pas exactement

les mêmes couleurs pour les mêmes voxels, ce qui n'est pas le cas des précodes qui tiennent compte de l'appartenance ou non à la même région des voxels voisins. Nous considérons que nous avons huit régions différentes. En effet, un précode 3d est composé de huit voxels, et peut donc dans le pire des cas avoir huit couleurs différentes. Nous pouvons sans problème effectuer les mêmes calculs en considérant que nous avons k régions différentes, mais cela les complique pour arriver exactement au même résultat.

Le nombre de configurations possibles est simplement 8^8. En effet, nous pouvons avoir n'importe quelle configuration dans l'image. La contrainte sur la segmentation en régions, imposant que chaque région soit 6-connexe, n'entraîne en effet aucune configuration interdite localement. Étant donné qu'une configuration est de taille $2 \times 2 \times 2$, nous avons donc huit voxels différents, et chacun peut avoir n'importe laquelle des huit couleurs. Il existe donc $16\,777\,216$ configurations différentes.

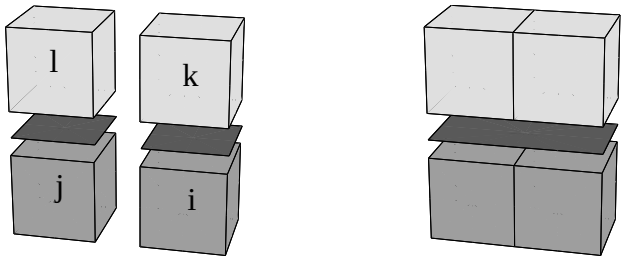
Nous comptons maintenant le nombre de configurations couvertes par chaque précode lignel. Le précode l_1 en couvre $8 \times 7 \times 7 \times 7 \times 8^4 = 11\,239\,424$. En effet, le voxel numéro 1 peut avoir n'importe quelle couleur parmi les 8, ses trois voisins ne sont pas de la même couleur donc peuvent avoir une des 7 couleurs restantes, et les 4 voxels libres peuvent avoir n'importe quelle couleur. Le précode l_2 couvre $8 \times 1 \times 7 \times 7 \times 8^4 = 1\,605\,632$ configurations, car le voxel 1 peut avoir n'importe laquelle des 8 couleurs, son voisin de gauche est forcément de la même couleur que lui, ses deux autres voisins ne sont pas de cette couleur, donc peuvent avoir une des 7 couleurs restantes et enfin les 4 voxels libres peuvent être de n'importe quelle couleur. Ce nombre de configurations couvertes est identique pour les précodes l_3 et l_4 car ces trois précodes sont isomorphes par rotation. Nous pouvons d'ailleurs facilement le vérifier en utilisant la même démarche que pour le précode l_2. Le précode l_5 couvre $8 \times 1 \times 1 \times 7 \times 8^4 = 229\,376$ configurations, car deux voisins du voxel 1 sont de la même couleur que lui. Les précodes l_6 et l_7 couvrent chacun ce même nombre de configurations. Enfin, le précode l_8 couvre $8 \times 1 \times 1 \times 1 \times 8^4 = 32\,768$ configurations.

Nous obtenons finalement que ces huit précodes lignels couvrent $11\,239\,424 + (3 \times 1\,605\,632) + (3 \times 229\,376) + 32\,768 = 16\,777\,216$, qui est bien le nombre de configurations possibles. Cela prouve que les 8 précodes lignels couvrent bien l'ensemble des précodes possibles.

5.6.4 Les précodes pour la carte de niveau 2

Pour ce niveau de simplification, nous fusionnons chaque couple de faces adjacentes, coplanaires, et incidentes à une arête de degré un ou deux (cf. définition 25). Ce type de fusion est possible uniquement si deux voxels adjacents i et j sont de même région et qu'il existe un autre voxel k adjacent à i étant d'une région différente, tel que le voxel adjacent à la fois à k et à j soit de la même région que k. Cette formulation compliquée est illustrée figure 5.65. Nous devons effectuer ce type de fusion uniquement lorsqu'un des quatre voxels i, j, k ou l est le voxel courant. En effet, le cas échéant nous avons déjà effectué ce traitement lors d'un précode précédent.

De manière un peu plus intuitive, nous pouvons effectuer une fusion de faces coplanaires s'il existe deux voxels adjacents de même région, et deux autres voxels adjacents d'une autre région « parallèle » aux deux premiers. Nous avons représenté figure 5.65.*a* ces quatre voxels, et schématisé les deux faces coplanaires qui vont être fusionnées. Si l'un des quatre voxels représenté sur cette figure appartient à une autre région, la fusion de faces est alors impossible. En effet, nous



a. Avant la fusion. b. Et après.

FIG. 5.65 – Une représentation partielle d'un précode où l'on peut fusionner deux faces coplanaires.

aurions alors une face supplémentaire entre ce voxel et un de ses voisins. De ce fait, l'arête au centre des quatre voxels ne serait plus de degré deux mais de degré trois.

Afin de trouver tous les précodes pour lesquels nous pouvons effectuer une fusion de faces coplanaires, il suffit d'étudier les différentes cas où une telle configuration peut se produire, ainsi que toutes les combinaisons possibles entre ces cas. Nous pouvons ainsi trouver de manière exhaustive chacun de ces précodes. Nous obtenons les 18 précodes présentés figures 5.66 et 5.67 (que nous appelons fc_i pour précode faces coplanaires numéro i).

Sur ces deux figures, nous avons représenté les précodes que nous devons traiter afin d'extraire la carte de niveau 2, en les classant afin de faciliter la lecture. Sur chaque ligne, nous avons représenté un premier précode, puis son image par l'application de la fonction *rotation* puis l'image de ce deuxième précode par cette même fonction. Chaque ligne représente donc trois précodes isomorphes par rotation. De ce fait, nous obtenons immédiatement un deuxième classement en colonne. La première colonne de la figure 5.66 montre les sous-cas du précode l_2, la deuxième les sous-cas du précode l_3 et la troisième ceux du précode l_4. De manière similaire, la première colonne de la figure 5.67 montre les précodes sous-cas du précode l_5, la deuxième ceux du précode l_6 et la troisième ceux du précode l_7.

Certains de ces précodes partiels sont composés de 5 voxels, et d'autres de 6. Par exemple, le précode fc_7 fixe une contrainte sur le voxel numéro 6 alors que le précode fc_1 non. Ce voxel peut donc, *a priori*, appartenir à n'importe quelle région. Mais en fait il ne peut pas appartenir à la même région que le voxel numéro 5 (son voisin droite). En effet, dans ce cas nous obtenons le précode fc_7. Ce voxel peut donc être de la même région que son voisin du bas, le voxel numéro 2, mais pas de la même région que son voisin de gauche. Il en est de même pour chaque précode étant composé seulement de 5 voxels. Remarquons que dans ces deux cas nous effectuons exactement le même traitement, pour le précode fc_1 une fusion de volumes et une fusion des deux faces coplanaires situées entre les voxels 1 et 3 pour la première, et entre les voxels 2 et 4 pour la deuxième.

En utilisant le même principe que pour les précodes lignels, nous pouvons définir un seul traitement pour chaque ensemble de précodes isomorphes par rotation. Nous devons donc définir six traitements différents, qui sont appelés sur des brins initiaux différents pour des précodes isomorphes par rotation. Ces traitements sont similaires et peuvent se trouver simplement en étudiant la carte de départ et celle que nous voulons obtenir. Nous donnons donc uniquement le traitement générique des trois premiers précodes, fc_1, fc_2 et fc_3, présenté algorithme 31. Nous pouvons

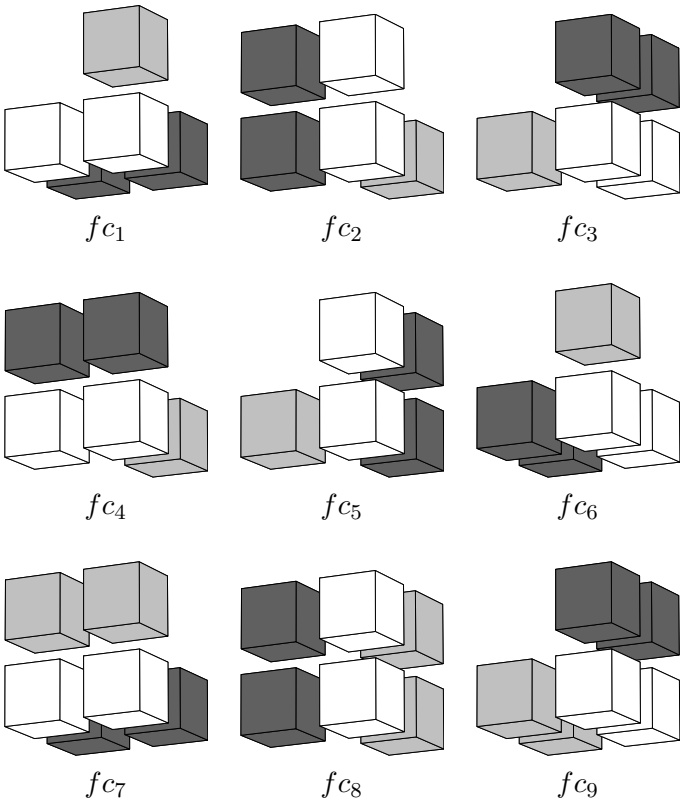


FIG. 5.66 – Les 9 premiers des 18 précodes à traiter afin de calculer la carte de niveau 2 en 3d, sous-cas des précodes l_2, l_3 et l_4.

vérifier pour ces trois précodes, que cet algorithme effectue bien la modification de la carte correspondant au précode traité, en étudiant son déroulement sur les figures 5.68, 5.69 et 5.70.

Cet algorithme prend trois brins en paramètres : b_1 qui appartient à la face du voxel qui va être fusionné avec le voxel courant, f_1 qui appartient à la face du voxel qui ne va pas être fusionné, mais pour lequel nous effectuons la fusion des deux faces incidentes à ce brin, et e_1 pour lequel nous n'effectuons aucune fusion. Il est assez proche de l'algorithme traitant les précodes l_2, l_3 et l_4. La principale différence concerne les traitements numérotés 1, 2 et 3, qui effectuent la fusion des deux faces incidentes au brin f_1. Cette fusion est effectuée de manière locale, autour du brin f_1, puis de manière un peu différente pour la face en vis à vis étant donné qu'il y a une seule demi-face présente. Au vu des modifications locales effectuées, nous avons ensuite seulement besoin de β_3-coudre les trois brins qui étaient β_3-libres, les autres brins conservant leur coutures. La suite de cet algorithme est très proche de l'algorithme pour les précodes lignels. La seule différence se situe au niveau de la création de la première face carrée, effectuée autour des brins d_1 et d_2. Cela revient à dire que nous créons deux nouveaux brins, et β_1-cousons les quatre brins pour former une face carrée. Nous récupérons ici le brin d_2, qui n'a plus d'utilité, ce qui évite une allocation mémoire suivie d'une destruction. La plupart des modifications de plongement sont faites lors de opérations de couture et décousures. Nous affectons seulement les coordonnées du nouveau sommet au sommet topologique créé.

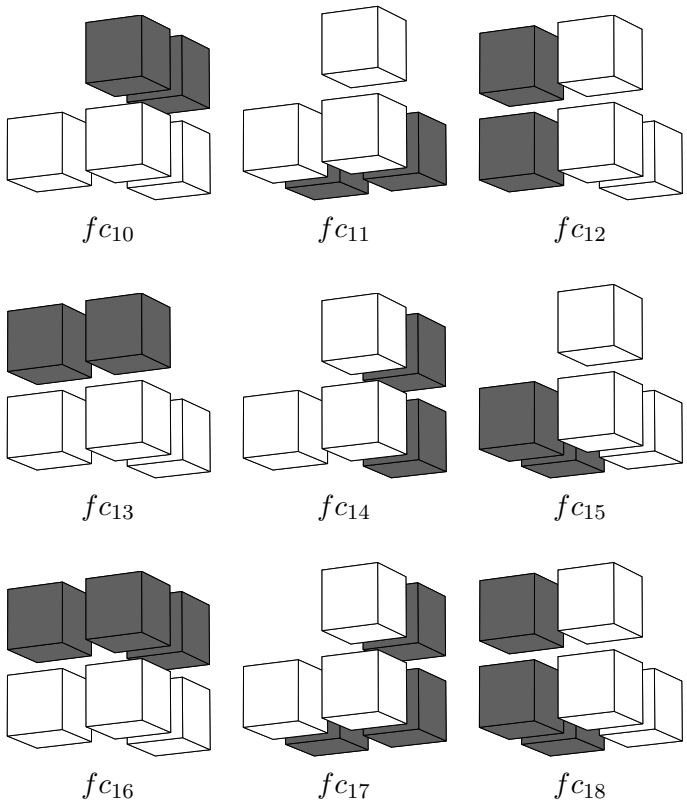
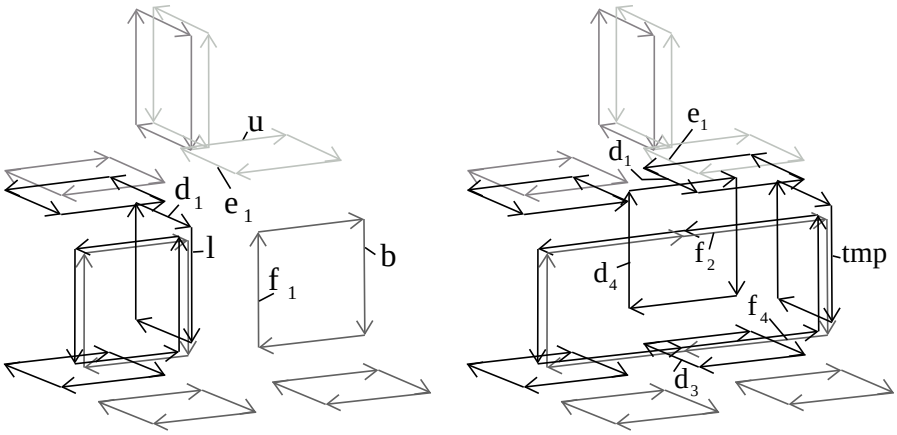


FIG. 5.67 – Les 9 derniers des 18 précodes à traiter afin de calculer la carte de niveau 2 en 3d, sous-cas des précodes l_5, l_6 et l_7.



a. Carte avant le traitement. b. Le résultat.

FIG. 5.68 – La carte avant et après le traitement du précode fc_1.

Algorithme 31 UneFusionVolumesUneFusionFacesCoplanaires3d

Entrée : d_1, f_1 et e_1 trois brins
(x, y, z) les coordonnées du nouveau sommet

Sortie : le « prochain » *last*.

$d_2 \leftarrow \beta_1(d_1)$; $d_3 \leftarrow \beta_1(d_2)$; $d_4 \leftarrow \beta_1(d_3)$;
β_1-découdre(d_1) ; β_1-découdre(d_2) ; β_1-découdre(d_3) ; β_1-découdre(d_4) ;

1 $t_1 \leftarrow \beta_{20}(f_1)$; $t_2 \leftarrow \beta_1(f_1)$;
β_1-découdre(t_1) ; β_0-découdre(t_2) ; β_1-coudre(t_1, t_2) ;

2 $t_1 \leftarrow \beta_0(f_1)$; $t_2 \leftarrow \beta_{21}(f_1)$;
β_1-découdre(t_1) ; β_0-découdre(t_2) ; β_1-coudre(t_1, t_2) ;

3 $a_1 \leftarrow \beta_2(d_2)$; $a_2 \leftarrow \beta_2(f_1)$; $a_3 \leftarrow \beta_1(a_1)$;
β_1-découdre(a_1) ; β_2-découdre(a_2) ;
β_1-coudre(a_1, a_2) ; β_1-coudre(a_2, f_1) ; β_1-coudre(f_1, a_3) ;

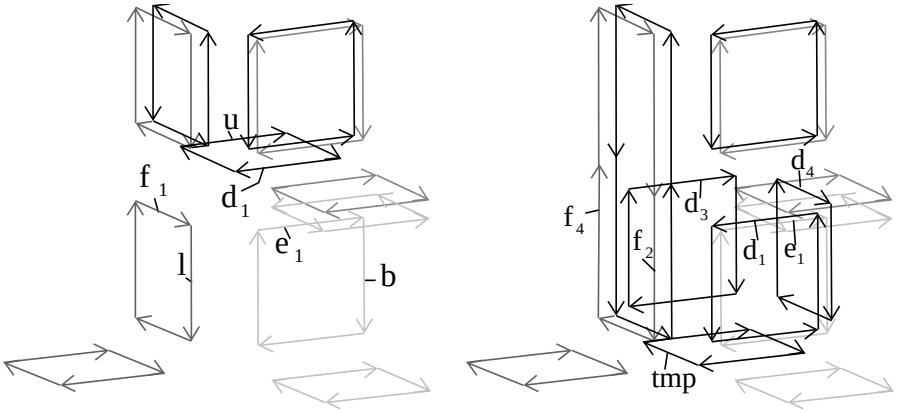
β_3-coudre(a_1, t_1) ; β_3-coudre$(a_2, \beta_0(t_1))$; β_3-coudre$(f_1, \beta_{00}(t_1))$;

créer une face carrée autour de d_1 et de d_2 ; créer une face carrée autour de d_3 ;
créer une face carrée autour de d_4 ; $tmp \leftarrow$ créer une face carrée topologique ;

β_2-coudre$(\beta_1(d_1), f_1)$; β_2-coudre$(\beta_{11}(d_1), \beta_0(tmp))$;
β_2-coudre$(\beta_0(d_1), \beta_1(d_4))$; β_2-coudre$(\beta_1(d_3), \beta_0(d_4))$;
β_2-coudre$(\beta_{11}(d_3), \beta_1(tmp))$; β_2-coudre$(\beta_0(d_3), a_1)$;
β_2-coudre(tmp, a_2) ; β_2-coudre$(\beta_{11}(tmp), \beta_{11}(d_4))$;

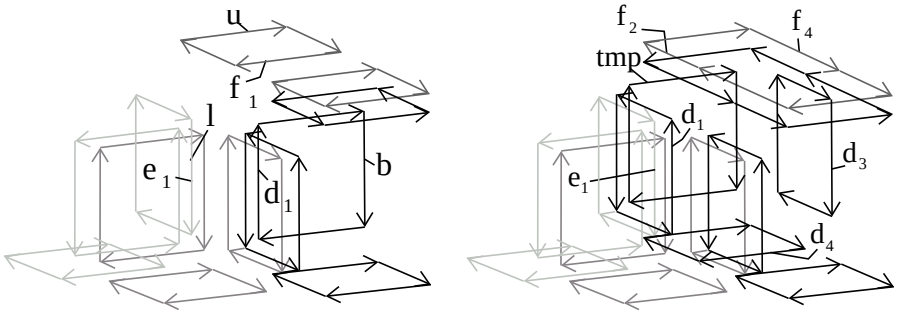
β_3-coudre la face contenant e_1 et celle contenant d_1 ;

Le plongement du sommet incident à $\beta_{11}(tmp) \leftarrow (x, y, z)$;
retourner tmp;



a. Carte avant le traitement. b. Le résultat.

FIG. 5.69 – La carte avant et après le traitement du précode fc_2.



a. Carte avant le traitement. b. Le résultat.

FIG. 5.70 – La carte avant et après le traitement du précode fc_3.

Nous pouvons voir sur les figures 5.68, 5.69 et 5.70, la carte courante avant de traiter les précodes, fc_1, fc_2 et fc_3, ainsi que la position des différents brins passés en paramètres à l'algorithme, puis le résultat de ce traitement. Nous ne détaillons pas ici les trois algorithmes de ces précodes, qui consistent uniquement, comme pour les précodes lignels, à initialiser les brins de départ, à appeler l'algorithme général, puis à retourner le prochain brin *last* pour le précode suivant. Les autres algorithmes généraux de traitement sont très proches de celui que nous venons de présenter. Suivant les précodes traités, il faut effectuer une ou deux fusions de volumes, et une ou deux fusions de faces coplanaires. Nous ne détaillons pas plus ces traitements parce que cela s'avèrerait long et fastidieux, alors qu'ils ne sont pas trop difficiles à retrouver en dessinant la carte courante et celle à obtenir.

Nous avons vu lors de la définition de la carte de niveau 2, (définition 29) les problèmes de déconnexion de faces qui pouvaient se produire, et la solution consistant à conserver des arêtes fictives. Lorsque nous avons défini l'algorithme d'extraction naïf, il nous a suffit de ne pas faire les fusions de faces qui entraînaient des déconnexions. Maintenant que nous balayons l'image et construisons directement la carte de niveau quelconque avec les précodes, ce n'est plus exactement pareil. Nous pouvons remarquer qu'une déconnexion de face peut se produire uniquement pour les précodes fc_{16}, fc_{17} et fc_{18}. En effet, c'est seulement dans ces cas que nous pouvons déconnecter le bord extérieur de la face de son bord intérieur, étant donné que nous effectuons deux fusions de faces coplanaires.

Ces trois précodes étant isomorphes par rotation, ils sont donc traités par le même algorithme qui effectue deux fusions de volumes et deux fusions de faces coplanaires. Cet algorithme va commencer par tester si la fusion des deux faces entraîne une déconnexion. Pour cela, il parcourt l'orbite $< \beta_1 >$ d'un brin d'une des deux faces. Si durant ce parcours un brin de la deuxième face est rencontré, c'est que la fusion entraîne une déconnexion, à condition que le deux brins ne soient pas cousus entre eux par β_0 et β_1. Dans ce cas, une seule des deux fusions est effectuée afin de conserver une arête fictive. Pour cela, il suffit d'exécuter le traitement de l'algorithme effectuant deux fusions de volumes et une seule fusion de faces, traitant les précodes fc_{10}, fc_{11} et fc_{12} ou celui traitant les précodes fc_{13}, fc_{14} et fc_{15}. En effet, le choix de la fusion de faces à effectuer n'est pas important, étant donné que la position de l'arête fictive conservée peut être quelconque.

Avec ce simple test et éventuellement un traitement différent, nous conservons les arêtes fictives permettant de régler les problèmes de déconnexion de faces. Cela se fait quand même au détriment de la complexité, puisque le traitement des précodes fc_{16}, fc_{17} et fc_{18} n'est plus effectué en *O(1)* étant donné que nous parcourons l'ensemble des brins d'une des deux faces. Nous conservons pour ce niveau de simplification éventuellement plusieurs arêtes fictives, étant donné que chaque arête représente un lignel de l'image.

Afin de nous assurer que ces 18 précodes couvrent bien toutes les possibilités de fusion de faces coplanaires, nous utilisons la même démarche que pour le niveau précédent. Nous dénombrons le nombre de cas existants où il faut effectuer des fusions de faces coplanaires, et comparons ensuite ce nombre aux cas couverts pas les 18 précodes.

Afin d'effectuer une fusion de faces coplanaires, le voxel courant et un de ses voisins doivent être de même région, et deux voxels parallèles doivent appartenir à une autre région. Remarquons que lorsque nous choisissons un des trois voisins du voxel courant, il y a seulement deux possibilités pour choisir les voxels parallèles, étant donné qu'il « reste » seulement deux directions possibles. Le nombre de cas possibles pour une de ces deux directions, pour un des trois voisins du voxel courant est de $8 \times 1 \times 7 \times 1 \times 8^4 = 229\,376$ cas. En effet, il y a 8 couleurs pour le voxel courant, le voisin choisi est de la même couleur, le voxel dans la direction choisi est d'une autre couleur (il y a donc 7 possibilités), et son voisin est de la même couleur. Enfin, les 4 voxels restant peuvent être de n'importe quelle couleur. Nous comptons maintenant le nombre de cas si les deux pixels choisis sont dans la deuxième direction possible. Ils sont au nombre de $8 \times 1 \times 7 \times 1 \times 8 \times 7 \times 8^2 = 200\,704$ cas. En effet, c'est identique pour les deux couples de voxels choisis, mais par contre le deuxième couple dans la première direction choisie ne doit pas être de la même région car ce cas a déjà été compté dans la première formule. Enfin les 2 voxels restants peuvent être de n'importe quelle couleur.

Pour un voisin du voxel courant, nous avons donc $229\,376 + 200\,704 = 430\,080$ cas possible, qu'il faut multiplier par trois étant donné que le voxel courant à trois voisins. Nous obtenons donc au total $1\,290\,240$ cas possibles pour lesquels nous devons effectuer une ou plusieurs fusions de faces coplanaires.

Nous dénombrons maintenant le nombre de cas couverts par chaque précode fc_1 à fc_{18}. Nous savons que le nombre de cas couverts par deux précodes isomorphes par rotation est identique, ce qui nous laisse seulement 6 cas à étudier.

Pour le précode fc_1, nous avons $8 \times 7 \times 7 \times 7 \times 8^2 = 175\,616$ cas. Le voxel courant est de la même couleur que son voisin gauche, pas de la même que les deux voxels derrière, ni que celle de son voisin du dessus. Ce dernier voxel n'est pas de la même couleur que son voisin gauche, mais peut-être de la même couleur que le voxel courant. Les deux voxels restants peuvent être de n'importe quelle couleur. Pour le précode fc_4 nous obtenons le même résultat par application d'un raisonnement similaire. Le précode fc_7 couvre $8 \times 7 \times 7 \times 8^2 = 25\,088$ cas, étant donné que deux couples de voxels sont de même région. Pour les précodes f_{10} et f_{13}, nous obtenons le même nombre de cas couverts : $8 \times 7 \times 7 \times 8^2 = 25\,088$ cas. Enfin, le précode f_{16} couvre $8 \times 7 \times 8^2 = 3\,584$ cas différents.

Nous obtenons finalement $430\,080$ cas différents qu'il faut multiplier par trois car nous avons compté uniquement les cas couverts par un précode de chaque classe d'équivalence. Nous obtenons bien finalement $1\,290\,240$, le même nombre que le nombre de cas possibles pour la fusion

de faces coplanaires. Comme pour les précodes lignels, cela prouve que les 18 précodes couvrent bien tous les cas pour lesquels nous devons effectuer une ou plusieurs fusions de faces coplanaires.

5.6.5 Les précodes pour la carte des frontières

Pour extraire ce niveau de simplification, nous devons maintenant fusionner chaque couple d'arêtes adjacentes, alignées et incidentes à un sommet de degré deux. Cette définition n'est plus valable lorsque nous sommes en présence d'arêtes fictives. Mais nous commençons par mettre de côté la gestion de ces arêtes, nous y reviendrons après avoir défini les précodes concernés par cette fusion.

Afin d'effectuer ce type de fusion, il est nécessaire que chaque couple de voxels dans une même direction soit composé de deux voxels de même région. En effet, afin que le sommet au centre d'un précode soit de degré deux, il faut qu'il y ait seulement deux arêtes qui lui soient incidentes. Pour trouver tous les précodes à traiter, il suffit d'étudier tous les cas où une telle configuration peut se produire. Ces cas étant encore peu nombreux, nous pouvons les définir de manière exhaustive et obtenir les 27 précodes présentés figures 5.71, 5.72 et 5.73.

Nous avons découpé ces précodes en trois parties, afin de pouvoir représenter sur chaque ligne les trois précodes isomorphes par rotation. Pour la figure 5.71, nous avons par exemple $rotation(f_1) = f_2$, $rotation(f_2) = f_3$ et $rotation(f_3) = f_1$. De plus, cette figure montre sur sa première colonne, les sous-cas du précode l_2, sur sa deuxième colonne ceux du précode l_3 et sur sa dernière colonne ceux du précode l_4. La figure 5.72 montre respectivement sur ses trois colonnes les sous-cas des précodes l_5, l_6 et l_7. Enfin la figure 5.73 montre les trois sous-cas du précode l_8. Remarquons que les deux précodes présentés figure 5.74 (et leurs rotations) ne sont pas des précodes à traiter pour l'obtention de la carte des frontières. En effet, après les fusions de faces coplanaires, il n'y a plus d'arête au centre du précode, et donc pas de fusion d'arêtes alignées à effectuer.

Comme pour les précodes du niveau précédent, nous donnons seulement l'algorithme 32 correspondant au traitement des précodes f_1, f_2 et f_3. Nous pouvons vérifier sur les trois figures 5.75, 5.78 et 5.79 le résultat de ce traitement pour ces trois précodes, ainsi que la position des brins initiaux passés à cet algorithme.

Nous avons découpé cet algorithme en 5 parties que nous expliquons maintenant pour le traitement du précode f_1. Nous notons d_2, d_3 et d_4 les trois brins de la face incidente à d_1 suivant l'ordre donné par β_1, et de manière similaire les brins des faces incidentes à e_1 et à f_1. La carte 5.75.*a* montre la configuration courante avant de traiter ce précode, ainsi que la position des trois brins passés en paramètre à l'algorithme 32. Afin d'expliquer ces traitements, nous déroulons chaque partie de cet algorithme. Pour simplifier la visualisation de la carte, nous représentons uniquement la partie concernée par le traitement en cours. Le premier traitement modifie les deux faces incidentes à f_1 et $\beta_2(f_1)$ présentées figure 5.76.*a*. La première ligne effectue la fusion de ces deux faces du côté où les arêtes ne doivent pas être fusionnées. Nous obtenons la carte de la figure 5.76.*b*. La deuxième ligne effectue la fusion de l'autre côté. Afin d'effectuer en même temps la fusion d'arêtes, nous β_1-cousons t_1 à f_3. Nous enlevons donc le brin f_2 de la deuxième arête et obtenons la carte présentée figure 5.76.*c*. Les brins $\beta_2(f_1)$, f_1 et f_2 sont maintenant inutiles. Ils sont décousus afin d'être utilisés plus tard, au lieu d'être détruits et de créer ensuite de nouveaux brins.

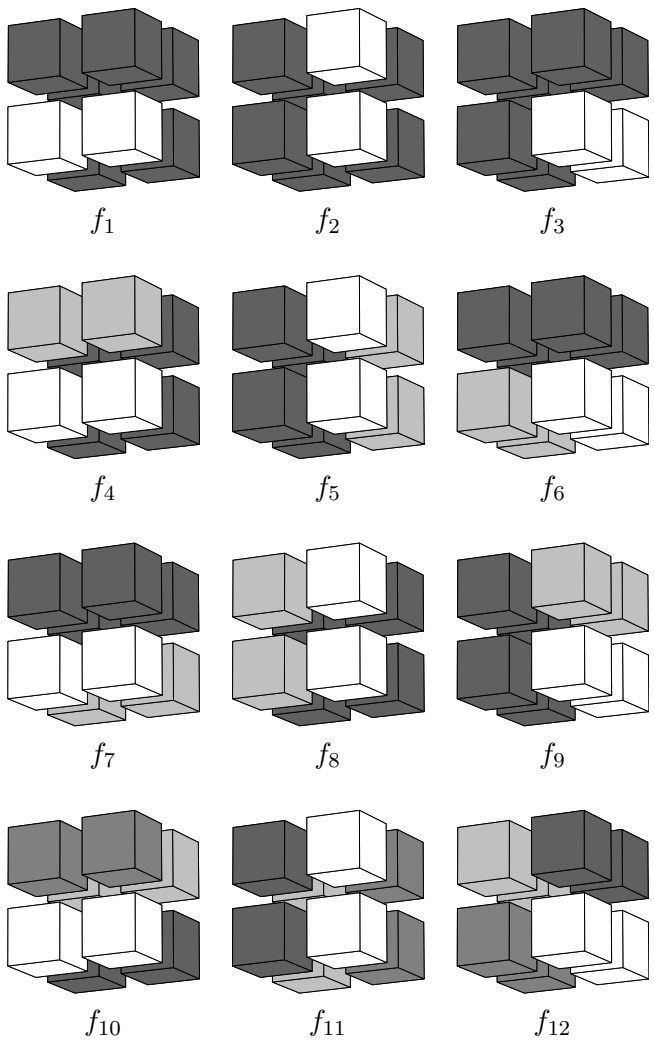


FIG. 5.71 – Les 12 premiers des 27 précodes à traiter afin de calculer la carte des frontières en 3d, sous-cas des précodes l_2, l_3 et l_4.

La deuxième partie de l'algorithme 32 effectue un traitement similaire pour les deux faces incidentes à e_1 et $\beta_2(e_1)$. La troisième partie traite la face incidente à $\beta_2(d_1)$. Elle insère un nouveau brin n_1 sur cette face et β_3-coud ce brin ainsi que $\beta_2(d_1)$ respectivement à f_2 et à f_3. Les autres brins de cette face conservent leurs β_3-coutures. La quatrième partie de l'algorithme effectue un traitement similaire pour la face incidente à $\beta_2(d_2)$. Enfin, la dernière partie de l'algorithme crée les trois faces restantes, en utilisant pour cela les brins libérés par les opérations précédentes. Il ne reste plus qu'à β_2-coudre correctement ces brins, et à affecter les coordonnées du nouveau sommet au plongement du sommet topologique créé lors de cet algorithme.

Nous pouvons vérifier que cet algorithme, appliqué sur la carte de la figure 5.75.*a*, produit bien celle de la figure 5.75.*b*. De même pour le précode f_2 où nous montrons figure 5.78.*a* la carte

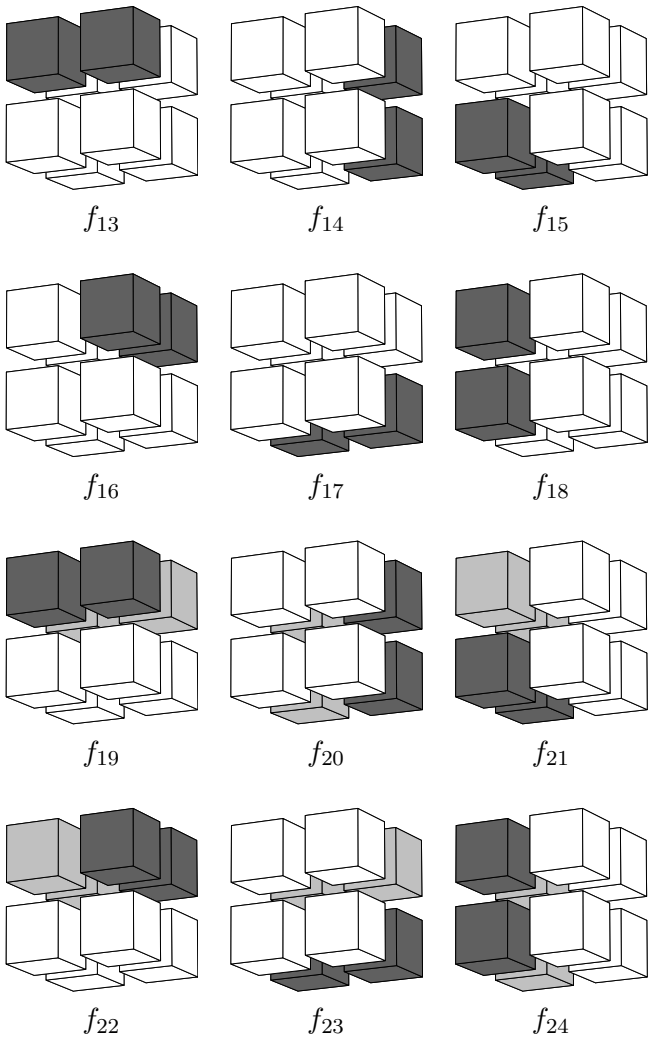


FIG. 5.72 – Les 12 précodes suivants des 27 précodes à traiter afin de calculer la carte des frontières en 3d, sous-cas des précodes l_5, l_6 et l_7.

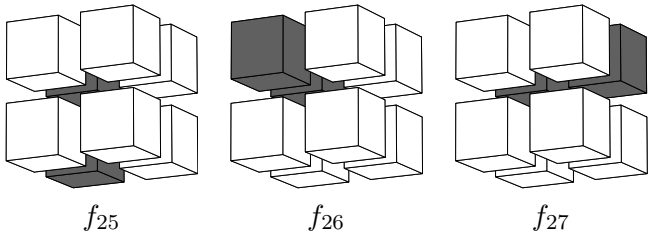


FIG. 5.73 – Les 3 derniers précodes des 27 précodes à traiter afin de calculer la carte des frontières en 3d, sous-cas du précode l_8.

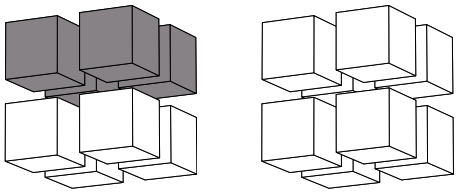


FIG. 5.74 – Deux précodes à ne pas traiter pour la construction de la carte des frontières.

Algorithme 32 Traitement_f_1_f_2_f_3_3d

Entrée : d_1, f_1 et e_1 trois brins
(x, y, z) les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

1 $t_1 \leftarrow \beta_{20}(f_1)$; β_1-découdre(t_1) ; β_0-découdre(f_2) ; β_1-coudre(t_1, f_2) ;
$t_1 \leftarrow \beta_{21}(f_1)$; β_0-découdre(t_1) ; β_1-découdre(f_3) ; β_1-coudre(f_3, t_1) ;
$m_1 \leftarrow \beta_2(f_1)$; β_2-découdre(f_1) ; β_0-découdre(f_1) ; β_3-découdre(m_1) ;

2 $t_1 \leftarrow \beta_{21}(e_1)$; β_0-découdre(t_1) ; β_1-découdre(e_4) ; β_1-coudre(e_4, t_1) ;
$t_1 \leftarrow \beta_{20}(e_1)$; β_1-découdre(t_1) ; β_0-découdre(e_3) ; β_0-coudre(e_3, t_1) ;
$n_1 \leftarrow \beta_2(e_1)$; β_2-découdre(n_1) ; β_1-découdre(e_1) ;
β_2-découdre(e_2) ; β_3-découdre(n_1) ;

3 $t_1 \leftarrow \beta_{21}(d_1)$; β_0-découdre(t_1) ; β_0-coudre(t_1, n_1) ; β_1-coudre($\beta_2(d_1), n_1$) ;
β_3-coudre(f_2, n_1) ; β_3-coudre($f_3, \beta_2(d_1)$) ;

4 $t_1 \leftarrow \beta_{20}(d_2)$; β_1-découdre(t_1) ; β_1-coudre(t_1, m_1) ; β_1-coudre($m_1, \beta_2(d_2)$) ;
β_3-coudre(e_4, m_1) ; β_3-coudre($e_3, \beta_2(d_2)$) ;

5 β_1-découdre(d_2) ; β_1-découdre(d_3) ; β_1-découdre(d_4) ;
créer une face carrée autour de d_1 et de d_2 ;
créer une face carrée autour de d_3, f_1 et f_4 ;
créer une face carrée autour de d_4, e_1 et e_2 ;

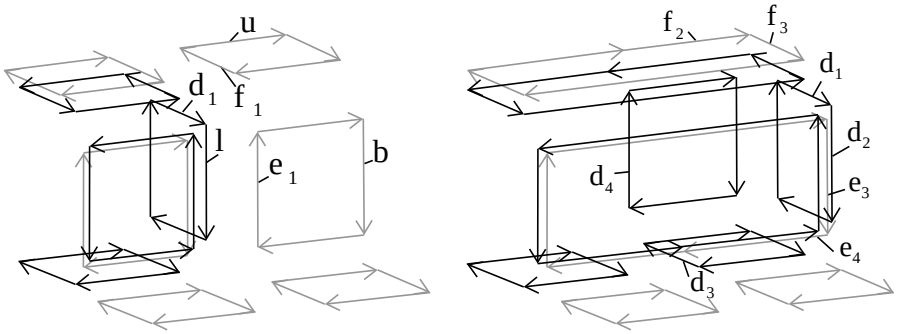
β_2-coudre($m_1, \beta_0(d_3)$) ; β_2-coudre($f_4, \beta_1(d_2)$) ; β_2-coudre($f_1, \beta_0(d_4)$) ;
β_2-coudre($e_2, \beta_0(d_1)$) ; β_2-coudre(n_1, e_1) ;

Le plongement du sommet incident à $f_4 \leftarrow (x, y, z)$;
retourner d_2;

avant le traitement et figure 5.78.*b* le résultat obtenu, et pour le précode f_3 de manière identique sur la figure 5.79.

Nous ne donnons pas les algorithmes correspondant à ces trois précodes, qui consistent simplement à appeler l'algorithme général sur les brins de départ particuliers, présentés sur chaque figure, et à retourner ensuite le prochain brin *last* pour le traitement du précode suivant. Nous ne détaillons pas les traitements des autres précodes, assez proches de ceux déjà présentés. De plus, certains traitements peuvent être regroupés afin de ne pas développer l'ensemble des 9 algorithmes.

En effet, de nombreux précodes effectuent les mêmes fusions de volumes et de faces coplanaires, comme par exemple les précodes f_1, f_4, f_7 et f_{10}. Pour ces quatre précodes, la seule différence porte sur le nombre de brins des deux arêtes alignées à fusionner, où sur la position de ces brins. Nous pouvons définir un seul traitement pour ces précodes, qui effectue la fusion



a. Carte avant le traitement. b. Le résultat.

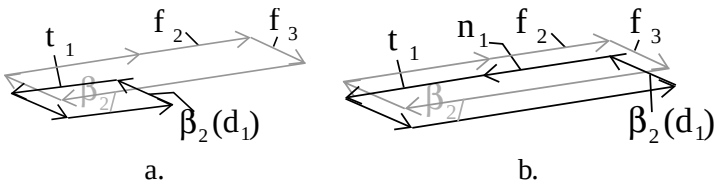
FIG. 5.75 – La carte avant et après le traitement du précode f_1.

a. b. c. d.

FIG. 5.76 – Le premier traitement de l'algorithme 32 appliqué sur le précode f_1.

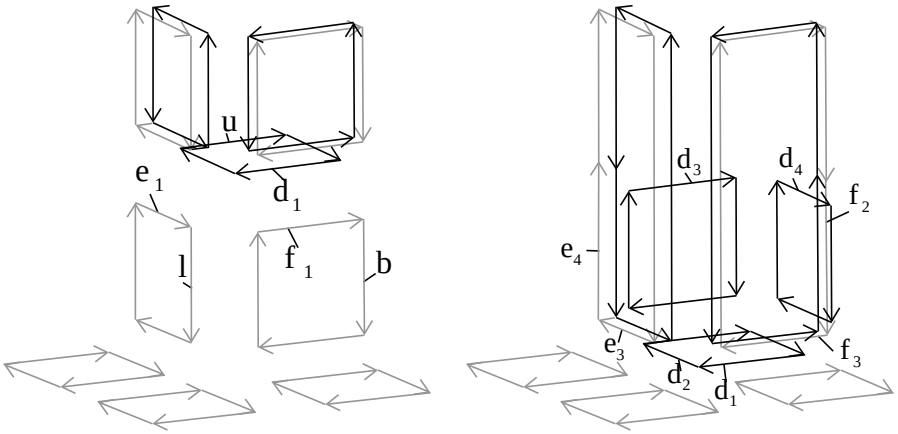
de volumes et des faces coplanaires de manière optimisée, puis la fusion des deux arêtes alignées au moyen de l'algorithme 20. Cette solution permet d'écrire un seul traitement pour les précodes $f_1 \dots f_9$, un autre pour les précodes f_{13}, f_{14}, f_{15}, f_{19}, f_{20} et f_{21}, un autre pour f_{16}, f_{17}, f_{18}, f_{22}, f_{23} et f_{24}, et enfin un dernier pour les trois précodes restants. Seulement 4 algorithmes différents sont à écrire au lieu de 9 si nous écrivons chaque méthode de manière optimale. Bien sûr, cette solution entraîne une légère perte en complexité, étant donné que nous ne connaissons pas exactement la configuration locale, et effectuons donc des tests, ou des boucles, ce qui n'est pas le cas sinon. Mais ces deux solutions permettent de choisir entre temps de développement plus important et complexité.

Afin d'extraire directement la carte des frontières d'une image au moyen de l'algorithme optimal, il suffit de tester quel est le précode courant parmi les 53 possibles (8 pour le niveau 1, 18 pour le niveau 2 et 27 pour le niveau 3) et simplement exécuter le traitement correspondant à ce



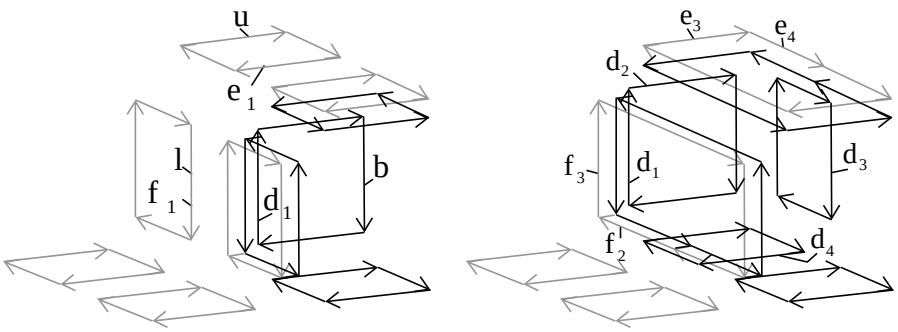
a. b.

FIG. 5.77 – Le troisième traitement de l'algorithme 32 appliqué sur le précode f_1.



a. Carte avant le traitement. b. Le résultat.

FIG. 5.78 – La carte avant et après le traitement du précode f_2.

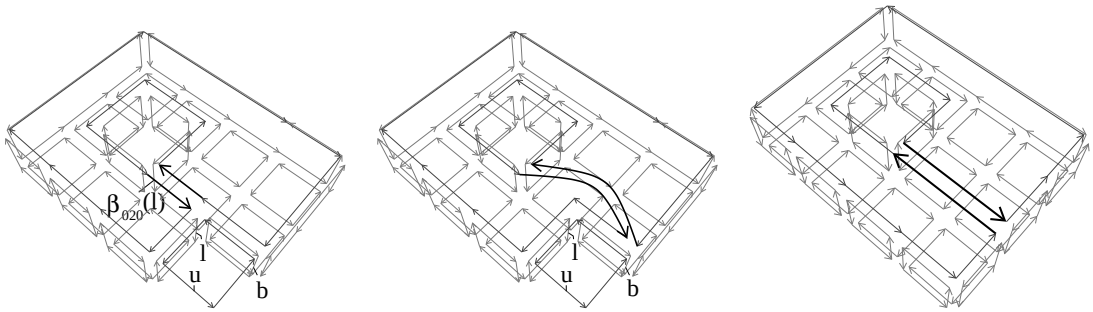


a. Carte avant le traitement. b. Le résultat.

FIG. 5.79 – La carte avant et après le traitement du précode f_3.

précode. Mais nous n'avons pas encore étudié le problème des arêtes fictives. Nous avons vu section 5.3.3 quels problèmes ces arêtes pouvaient engendrer lors de la fusion d'arêtes et comment nous devions les résoudre. Nous devons maintenant voir comment cette gestion intervient dans l'algorithme optimal basé sur les précodes.

Nous avons vu pour le niveau précédent, que la conservation de ces arêtes fictives ne posaient aucun problème. Pour les 27 précodes de fusion d'arêtes, nous utilisons l'algorithme 17 décalant toutes les arêtes fictives incidentes à un sommet. En effet, chacun de ces précodes fusionne les deux arêtes incidentes au sommet central du précode. L'appel de cet algorithme avant chacun de ces précodes décale les éventuelles arêtes fictives sur un autre sommet, et le traitement du précode pourra ensuite se dérouler de manière normale. Mais un autre problème n'est pas résolu par cette solution : lorsque deux bords d'une face sont reliés par un chemin composé de plusieurs arêtes fictives. Nous voulons dans ce cas obtenir une seule arête fictive dans la carte des frontières. Ce problème ne peut pas être résolu uniquement pour les précodes frontières, étant donné que les



a. b. c.

FIG. 5.80 – Un exemple du traitement particulier des précodes fc_{16}, fc_{17} et fc_{18} pour gérer les arêtes fictives.

arêtes fictives sont conservées par les précodes faces coplanaires. De ce fait, nous devons modifier le traitement des trois précodes fc_{16}, fc_{17} et fc_{18}.

Nous utilisons le pré-traitement présenté algorithme 33 pour le précode fc_{16}, afin de savoir quelle opération nous devons effectuer.

Algorithme 33 Pré-traitement du précode fc_{16} en 3d

Entrée : *last, up* et *behind*
$c = (x, y, z)$ les coordonnées du nouveau sommet
Sortie : le « prochain » *last*.

si $\beta_{020}(last)$ *est un brin d'une arête fictive* **alors**
  Décaler_arête($\beta_{0202}(last), \beta_{02}(behind)$);
  Exécuter le traitement du précode fc_{16};
sinon
  si $\beta_{020}(last) \neq \beta_{02}(behind)$ ***et*** $\beta_{02}(last) \in < \beta_1 > (\beta_{02}(behind))$ **alors**
    // Les deux fusions de faces entraînerait déconnexion
    Marquer l'arête incidente à $\beta_{02}(behind)$ comme étant une arête fictive ;
    Exécuter le traitement du précode fc_{13};
  sinon
    // Le cas normal, il n'y a pas d'arête fictive ni de déconnexion
    Exécuter le traitement du précode fc_{16};

Nous distinguons trois cas différents. Le premier lorsque le brin $\beta_{020}(last)$ appartient à une arête fictive. Ce cas est présenté figure 5.80.*a*. C'est seulement dans ce cas (et les deux cas isomorphes par rotation) que nous créons des « chemins » d'arêtes fictives. Pour éviter ces chemins, nous décalons l'arête fictive entre le brin $\beta_{02}(behind)$ et $\beta_{020}(behind)$. Ce décalage est effectué à l'aide de l'algorithme 18 déjà présenté lors de la définition de la carte topologique. Nous obtenons alors la carte présentée figure 5.80.*b* et nous pouvons maintenant exécuter le traitement normal du précode fc_{16} qui effectue les deux fusions de faces coplanaires et donne la carte présentée figure 5.80.*c*. Si le précode suivant est à nouveau fc_{16}, nous allons rencontrer ce même cas parti-

culier, et recommencer à « pousser » l'arête fictive. Nous obtenons au final une seule arête fictive de longueur quelconque, qui relie deux bords distincts de la face.

Le deuxième cas de ce pré-traitement teste si la fusion des deux faces coplanaires entraîne une déconnexion. Si c'est le cas, nous appliquons simplement le traitement du précode fc_{13}, qui effectue les mêmes fusions que celles du précode fc_{16}, à l'exception d'une fusion de faces coplanaires. Cela nous permet de conserver une arête fictive, que nous marquons afin de tester en *O(1)* si un brin est incident à une de ces arêtes. Enfin, le dernier cas est le cas « normal » où nous exécutons le traitement du précode fc_{16}. Ce principe est appliqué de manière similaire pour les deux autres précodes fc_{17} et fc_{18}. Il faut alors tester d'autres brins que ceux du pré-traitement du précode fc_{16} afin de savoir dans quel cas l'on se trouve, et changer le précode appelé dans le deuxième cas.

Nous montrons maintenant que nous avons bien traité tous les cas de fusion d'arêtes alignées. Pour cela, nous utilisons le même principe que pour les deux niveaux précédents, en comptant le nombre de cas où une telle fusion est possible, et en comparant ce nombre au nombre de cas couverts par les 27 précodes. Une manière simple de compter le nombre de cas où une fusion d'arêtes alignées est possible, est de faire le lien avec la dimension 2. En effet, une telle fusion est possible si chaque couple de voxels d'un précode dans une même direction est composé de deux voxels de même région. Cela revient à prendre un précode en 2d, à le placer dans une des trois directions possibles d'un précode 3d, et à le dupliquer pour les 4 voxels restants. Nous pouvons vérifier que cette technique de construction nous permet bien d'obtenir chacun des 27 précodes. Mais il y a trois précodes 2d avec lesquels cette technique n'est pas valide. Pour le précode 2d composé de 4 pixels de même couleur, nous obtenons le précode 3d composé de 8 voxels de même couleur. Dans ce cas, il n'y a pas d'arêtes à fusionner, étant donné qu'il n'y a pas d'arêtes au milieu de ce précode. Les deux autres précodes 2d pour lesquels cette technique n'est pas valide sont les précodes ayant 2 pixels voisins de même couleur, et les 2 autres pixels d'une autre couleur. En effet, ces deux précodes étendus en 3d donnent un précode similaire à celui présenté figure 5.74. Ce précode n'a pas d'arête en son centre, et il n'y a donc pas de fusion d'arêtes à effectuer.

Finalement, le nombre de précodes 3d pour lesquels une fusion d'arêtes alignées est nécessaire est donc de $3 \times (8^4 - 8 - 2 \times 8 \times 7)$. 8^4 est le nombre de cas en 2d, moins 8 pour enlever les cas où un précode a une seule couleur, et moins $2 \times 8 \times 7$ pour enlever les deux précodes en 2d ayant deux pixels voisins de même couleur et les deux autres pixels d'une autre couleur. Il faut multiplier par trois le nombre obtenu, car il y a trois directions possibles pour placer chaque précode 2d. Nous obtenons donc $11\,928$ cas différents.

Nous dénombrons maintenant les cas couverts par chaque précode de niveau 3. Pour les 12 précodes ayant seulement deux régions différentes, il est de $8 \times 7 = 56$. Pour les 12 précodes ayant trois régions différentes, il est de $8 \times 7 \times 6 = 336$. Et pour les 3 précodes ayant quatre régions différentes, il est de $8 \times 7 \times 6 \times 6 + 8 \times 7 \times 7 = 2408$. Le premier terme de cette somme compte les cas de variétés, et le deuxième les cas de non variétés. Nous obtenons finalement que le nombre de cas couverts par les 27 précodes est de $12 \times 56 + 12 \times 336 + 3 \times 2408 = 11\,928$: nous couvrons bien tous les cas possibles de fusion d'arêtes alignées.

Nous présentons figure 5.81 les arbres de couverture pour les précodes des trois premiers niveaux de simplification. Ces arbres sont au nombre de 8, chacun ayant pour racine un précode lignel. Un précode p_1 est fils d'un autre précode p_2 lorsque p_1 est couvert par p_2. Le précode l_1 ne possède pas de fils, étant donné qu'il n'y a aucun précode couvert par lui. En effet, pour ce

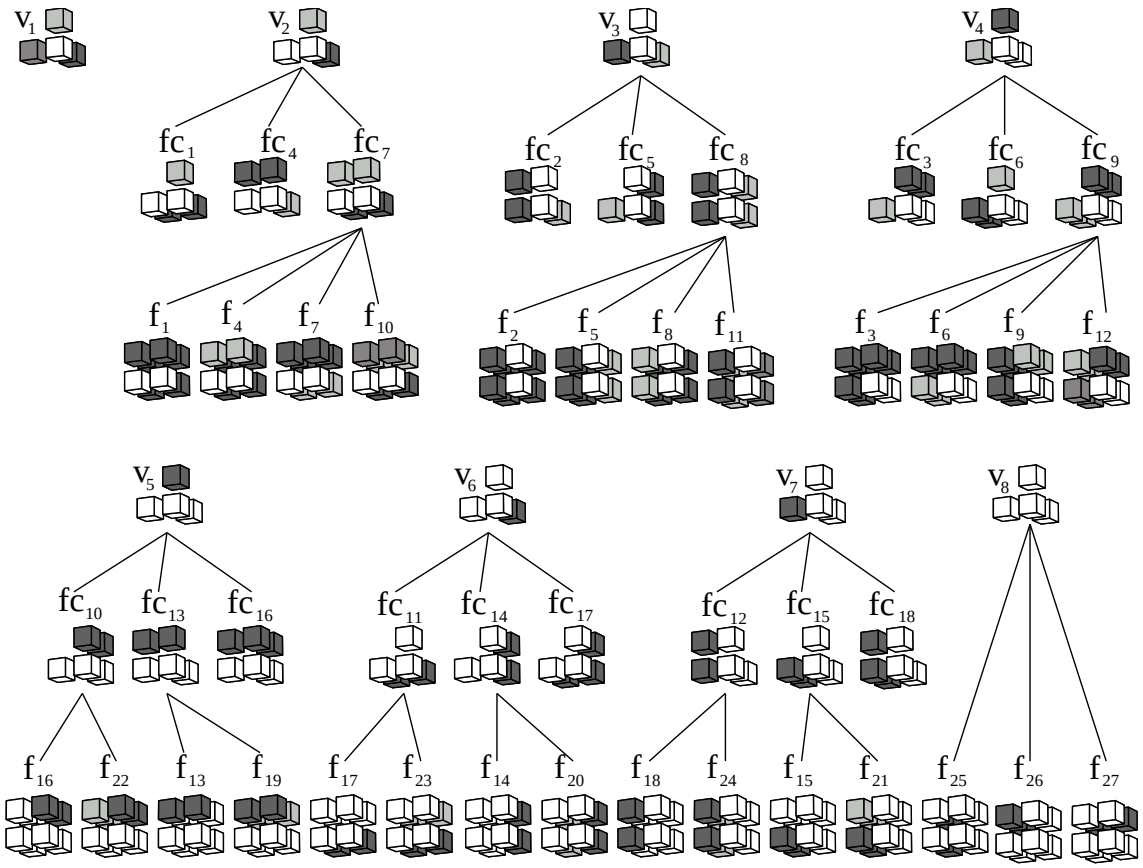


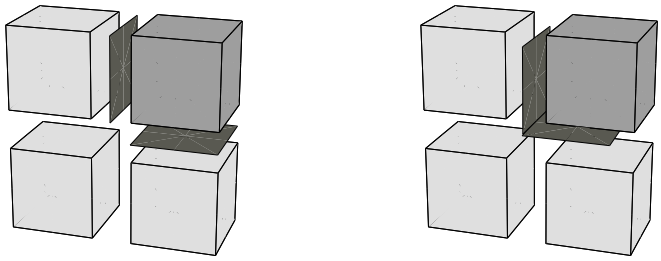
FIG. 5.81 – Les arbres de couverture pour les trois premiers niveaux de simplification.

précode, le voxel courant est différent de ses trois voisins, et il ne peut donc pas y avoir ni fusion de faces coplanaires, ni fusion d'arêtes alignées. Le précode l_8 n'a pas de fils pour la carte de niveau 2, mais en a trois pour la carte des frontières.

5.6.6 Les précodes pour la carte de niveau 4

Nous devons maintenant fusionner les faces non coplanaires adjacentes et incidentes à une arête de degré deux. Ce type de fusion se produit lorsque trois voxels d'un même plan sont de la même région, et que le quatrième est d'une autre région, comme nous pouvons le voir figure 5.82. Nous devons effectuer en même temps les fusions de faces coplanaires, et d'arêtes alignées, et traiter toutes les combinaisons possibles.

Nous pouvons à nouveau trouver tous les précodes à traiter de manière exhaustive, en faisant une étude de cas à partir des 8 précodes lignels. Par exemple si nous cherchons les conditions pour le précode l_1 pour lesquelles nous devons fusionner deux faces non coplanaires, nous obtenons trois possibilités. La première lorsque les voxels 2, 5 et 6 sont de la même région et le voxel 1 d'une autre, les deux autres obtenues par rotation. En effet, étant donné que nous sommes partis



a. Avant la fusion. b. Et après.

FIG. 5.82 – Une représentation partielle d'un précode où l'on peut fusionner deux faces non coplanaires.

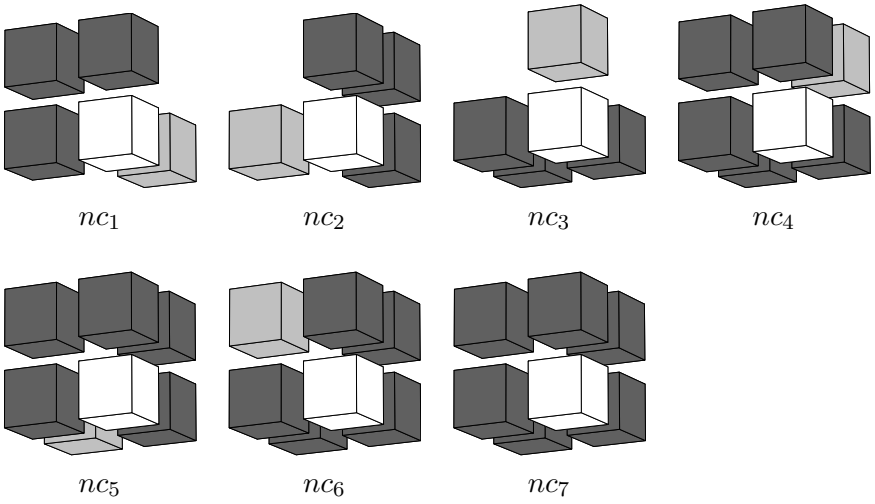


FIG. 5.83 – Les sept sous-cas du précode l_1 pour calculer la carte de niveau 4 en 3d.

du précode l_1, nous savons que le voxel courant n'est pas de la même région que ses trois voisins, et il n'existe pas d'autre cas possible. Nous devons alors étudier toutes les combinaisons possibles avec ces trois cas. Soit un seul de ces trois cas est vrai, ce qui donne alors 3 précodes, soit deux sont vrais, cela donne également 3 précodes, et enfin un précode où les trois sont vrais en même temps. Remarquons qu'il ne peut pas y avoir de fusion de faces coplanaires, étant donné que pour cela il faut que le voxel courant soit de la même région qu'au moins un de ses voisins. Pour le précode l_1, nous obtenons donc les 7 précodes de niveau 4 présentés figure 5.83 (que nous appelons précode nc_i pour précode non coplanaire numéro i). Les trois premiers de ces précodes sont les cas où une seule fusion est à faire. Remarquons que ces trois cas sont isomorphes par rotation. Les précodes nc_4, nc_5 et nc_6 sont les trois précodes où nous devons fusionner deux couples de faces non coplanaires, et enfin le dernier précode est le cas où nous devons fusionner les trois couples de faces non coplanaires.

Nous pouvons utiliser le même principe afin de trouver les sous-cas du précode l_2. C'est un peu plus complexe car nous devons maintenant tenir compte des faces coplanaires et de toutes

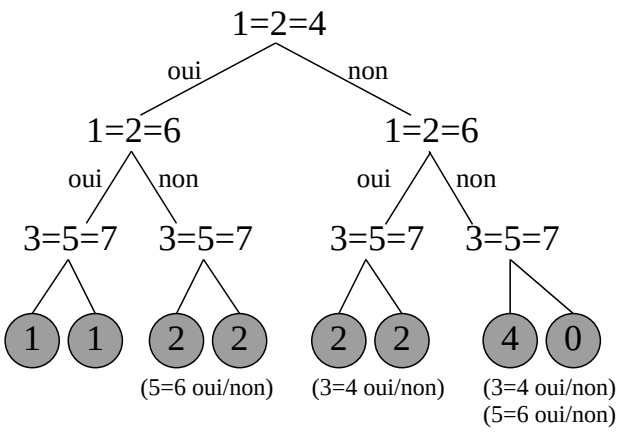


FIG. 5.84 – L'arbre de décision pour trouver les précodes de niveau 4 sous-cas du précode l_2.

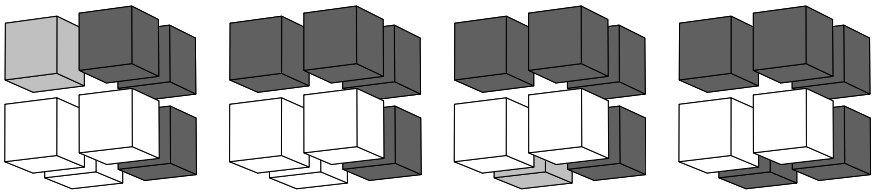


FIG. 5.85 – Quatre précodes de niveau 4 en 3d, sous-cas du précode l_2.

les combinaisons possibles entre ces deux types de fusions. Afin de ne pas oublier un cas, nous effectuons ce raisonnement à l'aide d'un arbre de décision présenté figure 5.84. Pour le précode l_2, nous savons que le voxel 1 est de la même région que le voxel 2, et qu'il est d'une région différente des voxels 3 et 5. Nous étudions donc toutes les possibilités pour les voxels restants, en tenant compte de ces contraintes. Nous écrivons $1 = 2 = 4$ pour dire que les voxels 1, 2 et 4 sont de la même région et le voxel 3 d'une région différente. Il existe les trois possibilités isomorphes par rotation. Remarquons que si aucune n'est vraie, il n'y a alors pas de fusion de faces non coplanaires à faire, nous sommes dans la branche à droite de l'arbre. Il n'y a alors pas de précode de niveau 4 correspondant. Pour les cas intermédiaires, il peut exister des fusions de faces coplanaires. Enfin, pour le cas où seul les trois voxels 3, 5 et 7 sont de même région, il existe 4 précodes différents, qui sont toutes les combinaisons possibles si les voxels 3 et 4 sont de la même région ou pas, et 5 et 6 de même région ou pas. Nous obtenons au final 14 cas différents, sous-cas du précode l_2. Nous obtenons le même nombre de cas pour les précodes l_3 et l_4, en effectuant un raisonnement identique. Nous présentons quatre de ces précodes figure 5.85. Nous pouvons obtenir l'ensemble des précodes sous-cas du précode l_2 en parcourant l'arbre de décision et en affectant les voxels correspondant suivant les cas. Par exemple le précode présenté figure 5.85.*a* correspond au cas où $1 = 2 = 4$, non $1 = 2 = 6$, $3 = 5 = 7$ et non $5 = 6$. Le précode de la figure 5.85.*a* correspond au même cas, sauf que ici $5 = 6$. La différence de traitement entre

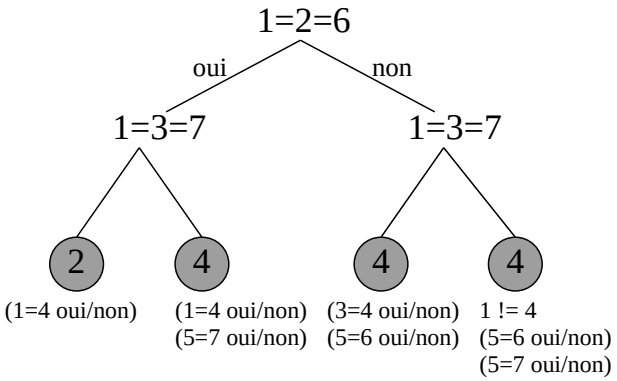


FIG. 5.86 – L'arbre de décision pour trouver les précodes de niveau 4 sous-cas du précode l_5.

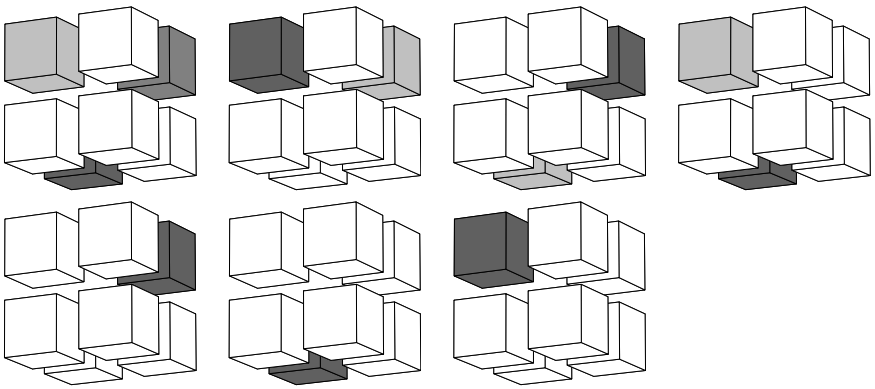


FIG. 5.87 – Les sept sous-cas du précode l_8 pour calculer la carte de niveau 4 en 3d.

ces deux précodes est que pour le deuxième il existe une fusion de faces coplanaires qui n'est pas effectuée pour le premier.

Pour le précode l_5, nous effectuons le même raisonnement à l'aide de l'arbre de décision présenté figure 5.86. La feuille la plus à droite de cet arbre regroupe 4 précodes différents pour lesquels le voxel 1 n'est pas de la même région que le voxel 4, puis les 4 combinaisons possibles suivant si les voxels $5 = 6$ ou non et les voxels $5 = 7$ ou non. Nous obtenons donc 14 précodes supplémentaires sous-cas du précode l_4. Nous obtenons les mêmes nombres pour les sous-cas des précodes l_6 et l_7 en appliquant la même démarche.

Il ne reste plus qu'à trouver les sous-cas du précode l_8. Pour ce précode, nous avons $1 = 2 = 3 = 5$. Les trois cas de fusion de faces non coplanaires sont lorsque le voxel 4 ou le voxel 6 ou le voxel 7 n'appartiennent pas à la même région que les voxels 1, 2, 3 et 5. Il existe donc trois précodes où une seule de ces conditions est vérifiée, trois précodes où deux sont vérifiées, et un où les trois sont vérifiées. Nous obtenons donc les 7 précodes présentés figure 5.87.

Au total nous avons donc 98 précodes afin d'extraire la carte de niveau 4, présentés annexe A. Tous ces précodes comprennent au moins une fusion de faces non coplanaires, et éventuellement

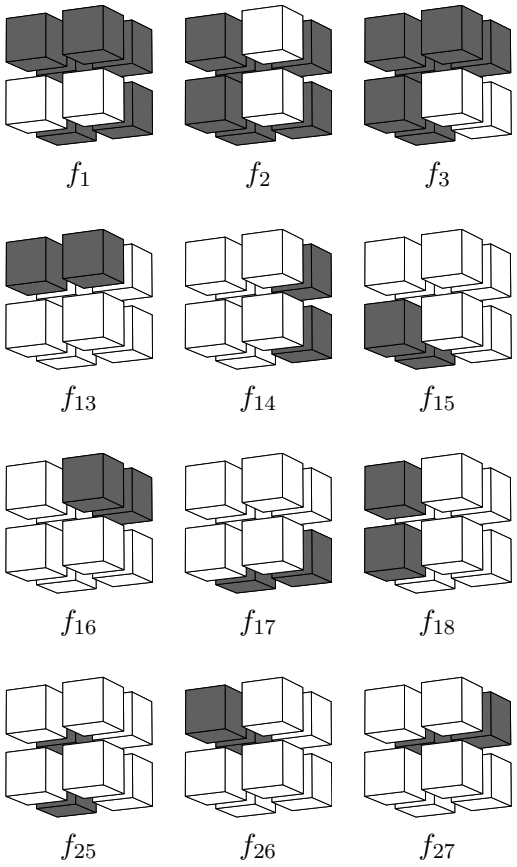
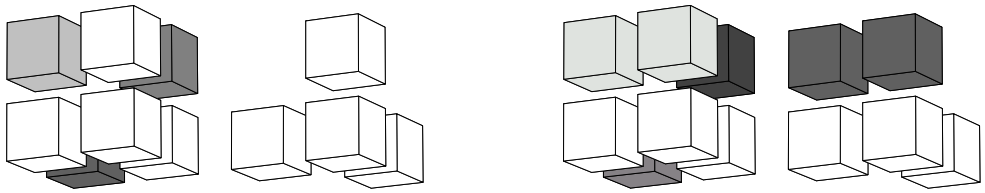


FIG. 5.88 – Les 12 précodes frontières pour lesquels nous effectuons une fusion de faces non coplanaires afin de calculer la carte de niveau 4.

des fusions de faces coplanaires. Mais il n'y a pas de cas de fusion d'arêtes alignées. En effet, si nous regardons les 27 précodes frontières pour lesquels nous effectuons une telle fusion, nous pouvons noter que pour 12 d'entre eux il faut fusionner des faces non coplanaires. Ce sont les 12 précodes présentés figure 5.88. Mais chacun de ces précodes est déjà couvert par un des 98 précodes de niveau 4. Par exemple le précode f_1 est couvert par le précode figure 5.85.*d*. Le traitement correspondant à ce précode effectue la fusion de faces non coplanaires. Après cette fusion, il n'y a plus d'arête au centre du précode, et donc pas de fusion d'arêtes alignées possible. En pratique, cela revient à dire que lorsque nous calculons la carte de niveau 4, nous testons en priorité les précodes de ce niveau avant les précodes de niveau 3. Si le précode courant est un précode de niveau 4, nous exécutons son traitement. Le cas échéant, nous testons si le précode courant est un de niveau 3.

Nous devons comme pour le niveau 2, ne pas effectuer de fusion de faces lorsque cela entraîne une déconnexion. Nous utilisons la même solution que celle du niveau 2, en exécutant un pré-traitement pour les précodes susceptibles d'entraîner une déconnexion. Nous présentons figure 5.89 deux précodes de niveau 4 pouvant entraîner une déconnexion de face. Le premier précode (figure 5.89.*a*) est un sous-cas du précode l_8. Ce précode doit fusionner la face entre les



a. b. l_8. c. d. fc_{13}.

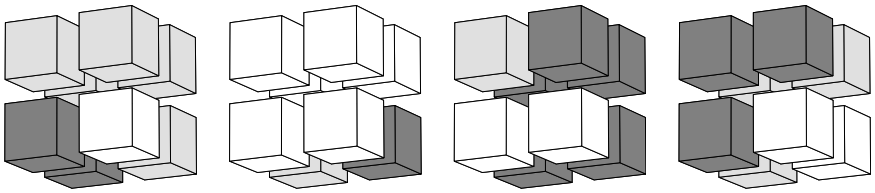
FIG. 5.89 – Deux précodes de niveau 4 pour lesquels une déconnexion de faces peut se produire, et leurs précodes de « remplacement ».

voxels 2 et 6 et celle entre les voxels 5 et 6. Nous testons si ces faces sont différentes ou non. Lorsque c'est une seule face, la fusion va entraîner une déconnexion. Dans ce cas, nous exécutons simplement le traitement du précode l_8, étant donné qu'il n'y a pas d'autre fusion à effectuer. Le précode présenté figure 5.89.*c* peut également entraîner une déconnexion. Ce précode effectue une fusion de faces non coplanaires ainsi qu'une fusion de faces coplanaires. Cette deuxième fusion ne peut pas entraîner de déconnexion. Si la première fusion en entraîne une, nous exécutons le traitement du précode fc_{13} qui est le précode effectuant uniquement la fusion des deux faces coplanaires. Dans ces deux cas, nous n'effectuons pas la fusion qui entraînerait une déconnexion, et créons donc une arête fictive.

Nous ne détaillons pas l'écriture des algorithmes correspondant à ces précodes de niveau 4. La technique utilisée est toujours la même : dessiner la carte courante avant le traitement d'un précode, la carte à obtenir, et écrire la suite des opérations transformant la première carte en la seconde. Notons que nous pouvons avoir désormais des faces non coplanaires, et que nous devons donc conserver un plongement face afin de garder la géométrie des régions représentées. Cela n'était pas obligatoire jusqu'à ce niveau, étant donné que nous avions uniquement des arêtes topologiques représentant des segments de droite. Il existe pour ce niveau 98 précodes différents, mais « seulement » 34 classes de précodes, obtenues en regroupant les précodes isomorphes par rotation. En effet, nous avons un précode sous-cas de l_1 (cf. figure 5.83) et un précode sous-cas de l_8 (cf. figure 5.87) qui sont stables par rotation. Nous obtenons donc $(98-2)/3+2=34$ classes d'équivalences. Ce nombre de méthodes reste raisonnable et permet d'envisager d'implanter le calcul de ce niveau de manière optimale, d'autant plus que des compromis peuvent être faits entre efficacité et nombre de méthodes à développer comme pour le niveau 3.

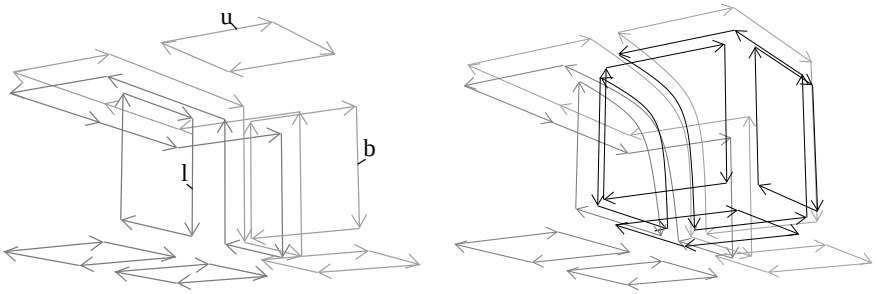
5.6.7 Les précodes pour la carte topologique

Pour ce dernier niveau nous devons fusionner les arêtes non alignées incidentes à des sommets de degré deux, si nous mettons momentanément de côté la gestion des arêtes fictives. Nous présentons figure 5.90 quatre exemples de précodes pour lesquels nous devons effectuer une telle fusion. Comme nous définissons ici les précodes de notre dernier niveau de simplification, nous devons également effectuer toutes les autres fusions possibles par rapport au voxel courant (volumes, faces coplanaires, arêtes alignées et faces non coplanaires). Par exemple pour le précode 5.90.*a* une fusion de faces non coplanaires et une fusion d'arêtes non alignées. Pour le précode 5.90.*c* il faut effectuer une fusion de volumes, une de faces coplanaires et une de faces non coplanaires.



a. b. c. d.

FIG. 5.90 – Quatre exemples de précodes de niveau 5 en 3d.



a. Carte avant le traitement. b. Le résultat.

FIG. 5.91 – Un exemple de précode où l'on fusionne deux arêtes non alignées en 3d.

Nous pouvons voir figure 5.91.*a* la carte correspondant au précode de la figure 5.90.*a*, avant son traitement, et figure 5.91.*b* la carte à obtenir. Nous observons sur cette carte que deux arêtes non alignées ont été fusionnées. Nous pouvons également vérifier que cette fusion a bien été réalisée autour d'un sommet de degré 2, ce qui est assez difficile à voir sur le précode correspondant.

Afin de traiter les arêtes fictives, nous utilisons le même principe que pour les précodes de niveau 3. Avant une fusion d'arêtes, nous décalons toutes les arêtes fictives incidentes au sommet supprimé lors de cette fusion. De plus, nous devons également, pour chaque précode de niveau 4 susceptible d'entraîner une déconnexion de faces, effectuer un pré-traitement similaire à celui présenté pour les précodes de niveau 3. Ce pré-traitement permet, suivant la configuration locale, de ne pas effectuer une fusion si elle entraîne une déconnexion, ce qui crée une arête fictive, où d'étirer une arête fictive afin d'éviter de créer des chemins d'arêtes fictives. Nous ne revenons pas ici sur ce pré-traitement qui va être très proche de celui déjà présenté section 5.6.5.

Nous avons calculé par programme le nombre de précodes supplémentaires à traiter pour ce dernier niveau : 228. Ces précodes sont présentés annexe B. Il existe « seulement » 76 classes d'équivalences de précodes isomorphes par rotation, ce qui diminue de manière conséquente le nombre de méthodes différentes à écrire. De plus, si nous effectuons uniquement les fusions de volumes et de faces de manière optimisée, nous devons seulement écrire 50 méthodes différentes. En effet, par cette technique, nous regroupons tous les précodes qui effectuent les mêmes fusions, mais où la fusion d'arêtes diffère uniquement par le nombre de brins de ces arêtes. Ces fusions d'arêtes étant, pour cette solution, effectuées à l'aide de l'algorithme générique, le traitement sera

TAB. 5.2 – Récapitulatif du nombre de précodes pour chaque niveau de simplification.

Niveau	1	2	3	4	5
Précodes pour ce niveau	8	18	27	98	228
Nombre total de précodes	8	26	53	151	379
Classes d'équivalences pour ce niveau	4	6	9	34	76
Nombre total de classes d'équivalences	4	10	19	53	129

alors le même pour ces différents cas. Cette solution permet d'envisager des implantations intermédiaires, plus ou moins optimisées, mais plus ou moins longues en temps de développement.

Le tableau 5.2 récapitule le nombre de précodes supplémentaires et total pour chaque niveau de simplification. Nous obtenons donc au total 379 précodes à traiter pour extraire la carte topologique en une seule passe de l'image. Ce nombre est important, mais pas en regard des 4140 précodes existant en dimension 3, ni même en regard des 958 précodes variétés. En effet, nous avons regroupé 579 de ces précodes variétés, ce qui est de beaucoup supérieur à nos prévisions initiales, estimées après avoir défini la carte topologique en dimension 2. De plus, nous avons seulement 129 classes d'équivalences de précodes isomorphes par rotation.

La complexité globale de l'algorithme optimal d'extraction est linéaire en le nombre de brins de l'image, multiplié par le nombre moyen de brins par face. Ce nombre est difficilement calculable étant donné qu'il dépend fortement du type d'image traitée. Mais nous verrons section 5.8 sur quelques images de tests, qu'il est relativement faible, et en moyenne égale à 7.

5.6.8 Le bord de l'image

Comme en dimension 2, c'est la définition du bord initial de l'image qui permet de s'affranchir des problèmes qui lui sont inhérents, et évite des cas particuliers et des traitements dupliqués. L'image en entrée de l'algorithme optimal est composée des voxels de $\{1 \dots n_1\} \times \{1 \dots n_2\} \times \{1 \dots n_3\}$, les autres voxels appartiennent tous à la région infinie. Nous plaquons cette image sur un tore, afin que le traitement d'un précode sur le bord droit de l'image crée la face de gauche du premier précode de la ligne suivante, et que le traitement d'un précode de la dernière ligne d'une plaque de voxels crée la face de derrière du voxel de la plaque suivante. Pour cela, nous identifions les voxels de coordonnées $(n_1 + 1, y, z)$ aux voxels $(0, y + 1, z)$ (le dernier voxel d'une ligne et le premier voxel de la ligne suivante en y), ainsi que les voxels de coordonnées $(x, y + 1, z)$ aux voxels $(x, 0, z + 1)$ (le dernier voxel d'une colonne, et le premier voxel de la même colonne de la plaque suivante).

Nous présentons figure 5.92 ce bord initial pour une image de taille $3 \times 2 \times n_3$. Ce bord initial est constitué des faces supérieures de l'image, représentant le bord inférieur de la région infinie. Ces faces vont de $1 \dots n_1 + 1$ en x et $1 \dots n_2 + 1$ en y. En effet, lors de notre balayage de l'image, nous « débordons » d'un voxel par rapport à l'image dans les 3 directions de l'espace, afin de tenir compte des frontières avec la région infinie. Ce bord contient également les faces derrière la première ligne de voxels de l'image, plus la face de gauche du premier voxel. De ce fait, le premier voxel possède bien les trois faces à sa gauche, derrière et au-dessus de lui. Les autres voxels de

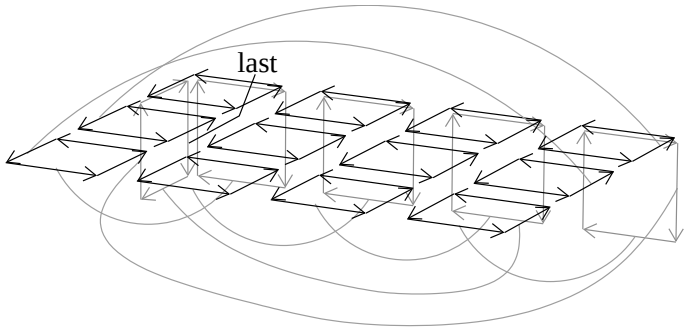
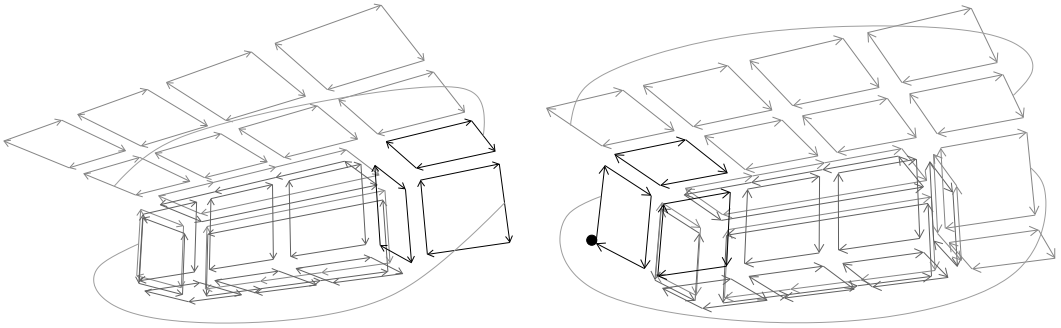


FIG. 5.92 – Le bord créé avant de commencer le traitement de l'image. Ici sur une image de $3 \times 2 \times n_3$.



a. Avant de traiter ce précode. b. Après traitement du précode.

FIG. 5.93 – Configuration d'une carte en cours de construction, avant et après le traitement du dernier précode de la première ligne.

cette première ligne possèdent une face au-dessus et derrière, la face de gauche étant à chaque fois créée par le traitement du voxel précédent.

Nous pouvons voir sur la figure 5.92 comment les faces aux extrémités du bord sont cousues afin d'identifier les voxels de la manière présentée ci-dessus. Sur cette figure, nous avons représenté les β_2-coutures de ces brins particuliers avec les courbes grises. Nous pouvons remarquer que cette carte initiale est 2-fermée. Nous β_2-cousons la dernière face d'une ligne avec la première face de la ligne suivante. De ce fait, le traitement du dernier précode d'une ligne crée la face de gauche du premier voxel de la ligne suivante, comme nous pouvons le vérifier figure 5.93. Cette figure montre la configuration d'une carte en cours de construction, avant de traiter le dernier précode de la première ligne. L'image traitée est ici de taille $3 \times 2 \times n_3$, et nous sommes en train d'extraire la carte de niveau 3. En effet, nous pouvons vérifier figure 5.93.*a* que nous avons déjà effectué des fusions de faces coplanaires et d'arêtes alignées. Le dernier précode de la première ligne est toujours le même, quelle que soit l'image, étant donné qu'un seul voxel appartient à l'image, tous les autres appartiennent à la région infinie. Nous pouvons vérifier figure 5.94 qu'il est impossible d'avoir n'importe quel précode sur les bords de l'image. Pour les 8 angles, nous

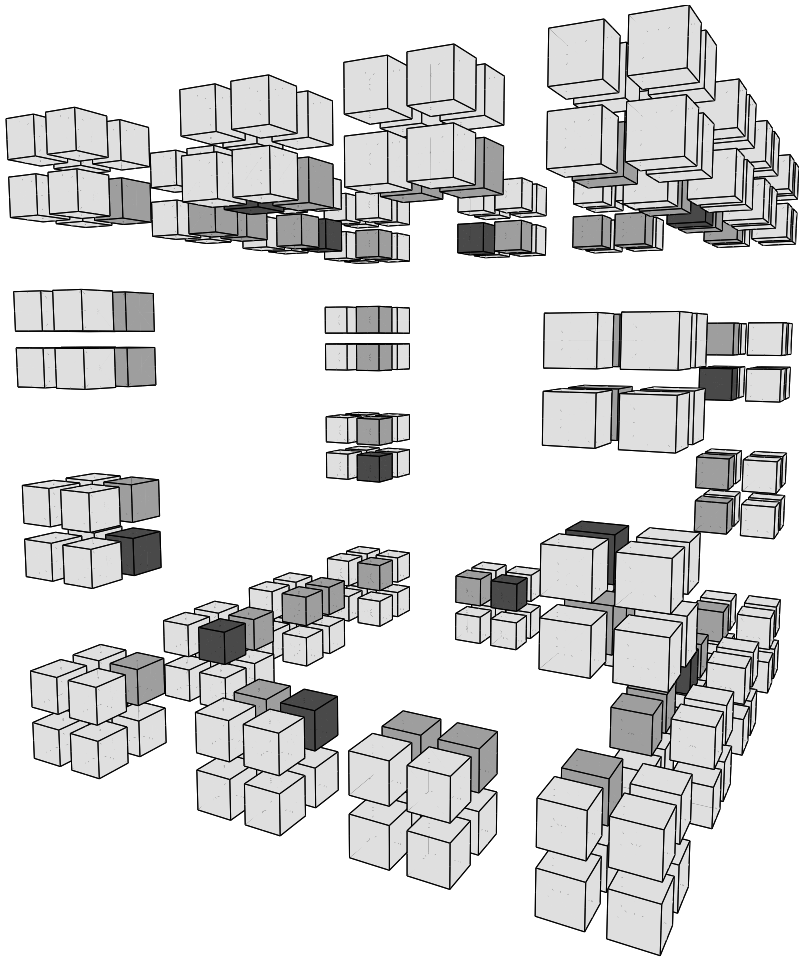


FIG. 5.94 – Les précodes pouvant se trouver sur les bords de l'image 3d.

avons toujours un seul précode possible. Pour les « arêtes » de l'image, il existe deux précodes possibles étant donné que deux voxels appartiennent à l'image, ils peuvent donc être de même région ou non. Enfin, nous n'avons pas représenté sur cette figure les précodes au centre des faces, mais il en existe 15 possibles par face. En effet, seuls 4 voxels de ces précodes appartiennent à l'image, et nous retrouvons donc les 15 précodes existant en 2d.

Pour l'exemple de la figure 5.93, nous traitons le dernier précode de la première ligne et nous fusionnons donc le voxel courant avec ses voisins de derrière et de dessus (précode l_7). Le calcul du nouveau sommet, passé en paramètre à chaque précode, est calculé à l'aide d'un modulo afin de simuler le tore. Lorsque nous traitons le voxel (i, j, k), ces coordonnées sont simplement $(i$ modulo $(n_1 + 1), j + (i$ div $(n_1 + 1), k)$. Ce nouveau sommet aura donc, pour ce dernier précode où le voxel courant est $(4, 1, 1)$, pour coordonnées $(0, 2, 1)$. Ce sommet est représenté par le point noir sur la figure 5.93.*b*. De par la construction initiale du bord et ce calcul des coordonnées, la face de droite du voxel courant se retrouve à gauche du premier voxel de la ligne suivante. De plus, le brin *last* est positionné correctement sur cette face, et nous pouvons donc traiter le prochain

précode sans cas particulier. Il en est de même pour les précodes de la dernière ligne de l'image, qui vont créer de manière similaire les faces derrière le premier voxel de la même colonne de la plaque suivante. Pour cela, nous devons effectuer le calcul des coordonnées du nouveau sommet modulo la deuxième coordonnées : $(i$ modulo (n_1+1), $(j+(i$ div $(n_1+1)))$ modulo (n_2+1), $k+(j+(i$ div $(n_1+1)))$ div $(n_2+1))$. Intuitivement, ce deuxième modulo revient à effectuer le deuxième repliement de la surface pour en faire un tore. Le traitement du dernier voxel d'une plaque va créer la face de derrière du premier voxel de la même colonne et de la plaque suivante, ainsi que la face de gauche du premier voxel de la plaque suivante.

Chaque précode fonctionne donc de manière identique qu'il soit au bord de l'image ou pas. De plus, la construction du bord initial ne dépend pas de la hauteur de l'image, étant donné que ce sont les précodes de bord qui créent les faces de bords des plaques suivantes. Enfin, de manière similaire à la dimension 2, après avoir fini le balayage de l'image, nous retrouvons le bord initial en-dessous et déconnecté de celle-ci. Il est alors inutile et peut être détruit.

5.7 Un algorithme d'extraction intermédiaire

Nous proposons ici un dernier algorithme d'extraction qui est une solution intermédiaire intéressante entre les algorithmes d'extractions naïf et optimal. En effet, l'algorithme naïf est en complexité quadratique en le nombre de brins de l'image. Il est donc difficilement utilisable sur de grosses images. L'algorithme optimal est bien meilleur en complexité, mais demande beaucoup de travail de développement. Même en regroupant les précodes isomorphes par rotations, nous devons quand même développer 129 méthodes différentes afin de calculer la carte topologique. C'est envisageable mais reste un travail important. C'est pour cette raison que nous proposons l'algorithme 34 qui est une troisième approche permettant d'implanter rapidement un algorithme d'extraction ayant une complexitée linéaire, mais n'étant pas optimal.

Cet algorithme reprend le principe de l'algorithme optimal. Il commence par construire le bord de l'image de manière identique à cet algorithme, puis effectue un balayage de l'image selon le même ordre. Nous conservons le même invariant que pour l'algorithme optimal : le voxel courant possède toujours une face à sa gauche, dessus et derrière, désignées par les brins *last*, *up* et *behind*.

Au cours de ce balayage, cet algorithme crée un cube topologique, (ligne 1) correspondant au voxel courant, et le β_3-coud correctement à ces trois faces. Ce traitement correspond au précode l_1 pour lequel aucune fusion n'est effectuée. Afin de faciliter la lecture de l'algorithme, nous nommons ensuite les différentes cellules incidentes à ces trois faces, et susceptibles d'être supprimées par une fusion. Les traitements suivants sont comparables aux cinq traitements effectués pour l'algorithme d'extraction naïf. La différence est que ces traitements sont ici effectués uniquement pour les cellules incidentes au cube qui vient d'être créé. Par exemple, pour le niveau 1, nous testons si les trois volumes v_1, v_2 et v_3 (incidents aux brins *last*, *up* et *behind*) sont de la même région que le voxel courant, et lorsque c'est le cas nous effectuons la fusion de ces volumes avec le voxel courant. Les traitements sont ensuite similaires pour les autres niveaux. Remarquons que pour la fusion de faces, nous avons seulement trois arêtes le long desquelles nous pouvons effectuer une fusion, et pour la fusion d'arêtes un seul sommet.

Nous utilisons les mêmes principes que ceux déjà présentés lors de l'algorithme optimal afin de gérer correctement les arêtes fictives. Pour une fusion de deux faces, nous testons si cela entraîne

Algorithme 34 Extraction « intermédiaire » de la carte de niveau n en 3d

Entrée : Une image I segmentée en régions de $n_1 \times n_2 \times n_3$ voxels
n le niveau de la carte que l'on veut extraire

Sortie : La carte de niveau n de l'image I.

```
1  last ← Construire le bord supérieur de la carte;
2  pour k = 1 à n_3 + 1 faire
       pour j = 1 à n_2 + 1 faire
           pour i = 1 à n_1 + 1 faire
3              tmp ← créer un cube topologique ;
               β_3-coudre la face contenant last et celle contenant β_2112(tmp) ;
               β_3-coudre la face contenant up et celle contenant β_021(tmp) ;
               β_3-coudre la face contenant behind et celle contenant β_2(tmp) ;
               x ← i modulo (n_1 + 1);
               y ← (j + (i div (n_1 + 1))) modulo (n_2 + 1);
               z ← k + (j + (i div (n_1 + 1))) div (n_2 + 1);
               Le plongement du sommet incident à β_11(tmp) ← (x, y, z);
4              // Notons v_1, v_2 et v_3 les volumes incidents à last, up et behind;
               // Notons f_1, f_2 et f_3 les faces incidentes à last, up et behind;
               // Notons a_1, a_2 et a_3 les arêtes incidentes à β_0(last), β_0(up) et β_11(behind);
               // Notons s le sommet incident à last;
5              si v_1 (resp. v_2, v_3) est de la même région que le voxel courant alors
                   Fusionner v_1 (resp. v_2, v_3) avec le voxel courant le long de f_1 (resp. f_2, f_3);
               si n ≥ 2 alors
6                  si a_1 (resp. a_2, a_3) est de degré un ou deux, que les deux faces incidentes à cette
                   arêtes sont coplanaires et que la fusion de ces deux faces n'entraîne pas déconnexion alors
                       Fusionner les deux faces incidentes à a_1 (resp. a_2, a_3);
                   si n ≥ 3 alors
7                      si s est de degré deux et que les deux arêtes incidentes à ce sommet sont
                       alignées alors
                           Fusionner les deux arêtes incidentes à s;
                       si n ≥ 4 alors
8                          si a_1 (resp. a_2, a_3) est de degré un ou deux et que la fusion de ces deux
                           faces n'entraîne pas déconnexion alors
                               Fusionner les deux faces incidentes à a_1 (resp. a_2, a_3);
                           si n ≥ 5 alors
9                              si s est de degré deux alors
                                   Fusionner les deux arêtes incidentes à s;
               last ← tmp;
10 Calculer l'arbre d'inclusion des régions;
```

une déconnexion, et lorsque c'est le cas nous testons s'il existe une arête fictive pouvant être étirée afin de ne pas créer des chemins d'arêtes fictives. Lors d'une fusion d'arêtes, nous appelons préalablement l'algorithme décalant l'ensemble des arêtes fictives incidentes à un sommet.

Cet algorithme est intermédiaire entre les deux algorithmes précédents. Il utilise le principe de balayage de l'image de l'algorithme optimal afin d'éviter de créer initialement toute la carte de niveau 0, donc énormément de brins, pour ensuite en supprimer beaucoup lors des différentes fusions. Mais il utilise le principe des fusions génériques de l'algorithme naïf, afin d'éviter de définir les traitements des précodes. La perte en complexité par rapport à l'algorithme optimal est relativement faible. En effet, nous créons ici des brins pour le cube initial, pour les détruire éventuellement ensuite lors des fusions, ce qui entraîne une perte en temps d'exécution. De plus, les fusions sont effectuées de manière générique, et nous allons donc effectuer des boucles et des tests, ce qui n'est pas le cas pour l'algorithme optimal. Mais l'ordre de complexité général reste le même que celui de l'algorithme optimal, et l'espace mémoire nécessaire est de l'ordre de l'espace mémoire du niveau calculé. Ce n'est pas le cas pour l'algorithme naïf, qui requiert beaucoup d'espace mémoire pour la construction du niveau 0.

Enfin, cette solution peut également être améliorée en calculant certains niveaux de simplification au moyen des précodes, et en effectuant les autres fusions de manière classique. Nous pouvons par exemple envisager de traiter les 8 précodes lignels, ce qui évite la construction des brins pour les faces entre deux voxels de même région, et permet d'avoir à définir seulement les 4 traitements correspondant au niveau 1.

5.8 Expérimentations et analyse

Nous avons implanté l'algorithme optimal d'extraction à base de précodes pour les trois premiers niveaux de simplification. En effet, le plongement utilisé pour ces trois premiers niveaux est un plongement sommet, où nous associons à chaque sommet topologique les coordonnées d'un point 3d. Ce plongement n'est plus valable pour les niveaux 4 et 5 étant donné que les faces ne sont plus forcément planes. Pour ces niveaux, nous avons utilisé le plongement face, tout d'abord implanté avec des 2-cartes. Mais les premiers résultats obtenus ont été catastrophiques en temps d'exécution.

À titre de comparaison, pour une image de taille $64 \times 64 \times 64$, le calcul de la carte de niveau 4 ayant environ $55\,000$ brins prend plus de 40 minutes avec le plongement 2-cartes, et 1.6 secondes avec le plongement 2-G-cartes. Cette image a environ $210\,000$ brins dans les 2-cartes de plongements, et il y a plus de 400 millions retournements de ces brins. Cela explique le temps d'exécution important. En effet, nous retournons une carte de plongement lorsqu'elle n'a pas la même orientation qu'une autre carte, et qu'elles doivent être fusionnées. Mais la carte résultante de cette fusion pourra être à nouveau retournée lors d'autres fusions.

Nous avons donc abandonné l'idée de représenter les surfaces avec des 2-cartes au profit des 2-G-cartes. En effet, même si l'espace mémoire occupé par ces dernières est deux fois plus important que celui occupé par les 2-cartes, le gain en temps d'exécution justifie complètement cette perte d'espace mémoire. Nous avons implanté le plongement face (présenté section 5.4) car il est plus simple que le plongement face ouverte, arête ouverte et sommet. Mais ce plongement n'est pas optimal en espace mémoire, étant donné qu'il représente plusieurs fois le plongement

TAB. 5.3 – Cœur, $64 \times 64 \times 64$, 204 régions.

	Niveau 1	Niveau 2	Niveau 3	Niveau 4	Niveau 5
Nombre de brins	429 552	88 140	38 382	23 370	15 934
Nombre de sommets géométriques	107 756	37 562	12 183	12 928	12 928
Nombre de brins par face	4	12	5,2	9,8	6,7
Nombre d'arêtes fictives	0	55	4	209	85
Espace mémoire en méga-octets	12,3	2,67	1,12	1,64	1,37
Temps de calcul en secondes	1,73	1,57	1,54	1,6	1,52

des sommets et des arêtes de la carte topologique. De ce fait, mais également car nous utilisons les 2-G-cartes pour représenter les surfaces, l'espace mémoire occupé par la carte topologique est généralement un peu plus important que celui occupé par la carte de niveau 3. Il faudrait effectuer les mêmes comparaisons en implantant le plongement face ouverte, arête ouverte et sommet afin de comparer les espaces mémoires « optimaux » de ces niveaux de simplification. De plus, il est possible d'utiliser les 2-G-cartes lors de la construction de la carte topologique, afin de ne pas avoir de nombreux retournements à effectuer, puis de les transformer en 2-cartes afin de diminuer l'espace mémoire utilisé. Si les modifications effectuées par la suite ne demandent pas trop de retournements, la perte en temps d'exécution sera minime et le gain en espace mémoire important.

Ces 2-G-cartes représentent le plongement de chaque face frontière de la carte topologique. Elles sont implantées de manière similaire aux cartes topologiques 2d, en fusionnant les arêtes incidentes à des sommets de degré deux. Elles sont plongées arêtes ouvertes et sommets, le plongement des arêtes étant effectué avec des 1-G-cartes. Nous obtenons un modèle hiérarchique, qui a pour avantage de pouvoir facilement s'étendre en dimension supérieure. De plus, ce plongement réutilise le travail effectué sur la carte topologique 2d ainsi que certaines fonctions, comme par exemple les fusions de faces et d'arêtes en 2d. Par manque de temps, nous n'avons pas implanté l'algorithme optimal d'extraction pour les niveaux 4 et 5. Nous avons utilisé l'algorithme intermédiaire, en effectuant les traitements des 8 précodes lignels de manière optimisée, puis les autres fusions de manière classique. Cela montre l'avantage de cet algorithme, et permet de voir que la perte en temps d'exécution n'est pas trop importante.

Nous présentons les résultats obtenus pour 5 images différentes. Le manque de temps et d'images 3d segmentées ont en effet limité le nombre de tests effectués. Ces images sont en majorité des images médicales, étant donné que c'est le domaine d'application utilisant le plus les images 3d. Le tableau 5.3 présente les résultats de l'extraction des 5 niveaux de simplification pour une image IRM de cœur de taille $64 \times 64 \times 64$ et composée de 204 régions. Les tableaux 5.4, 5.5 et 5.6 présentent les résultats pour trois IRM de cerveaux. Le tableau 5.7 présente les résultats pour une image provenant de l'institut français du pétrole représentant un bloc géologique afin d'étudier les failles sismiques. Cette image est composée de seulement deux régions, car le but de la segmentation est de différencier les voxels appartenant au bord des failles des autres voxels.

La première ligne de ces tableaux donne le nombre de brins nécessaires à la représentation de chaque niveau de simplification. Nous voyons que ce nombre diminue de manière impressionnante, mais ce n'est pas étonnant en regard de tous les éléments fusionnés. Ces nombres montrent également que le temps des traitements concernant uniquement la partie topologique vont être grandement améliorés étant donné le nombre d'éléments beaucoup moins important à traiter. Nous

TAB. 5.4 – Cerveau1, $256 \times 256 \times 69$, 52909 régions.

	Niveau 1	Niveau 2	Niveau 3	Niveau 4	Niveau 5
Nombre de brins	10 688 776	6 892 772	6 001 862	3 246 718	2 493 866
Nombre de sommets géométriques	2 753 856	2 286 360	1 840 905	1 882 640	1 882 640
Nombre de brins par face	4	5,5	4,8	8,9	6,9
Nombre d'arêtes fictives	0	1 557	477	103 729	58 824
Espace mémoire en méga-octets	308	203	175	244	217
Temps de calcul en secondes	577	563	562	1097	890

TAB. 5.5 – Cerveau2, $256 \times 256 \times 72$, 14627 régions.

	Niveau 1	Niveau 2	Niveau 3	Niveau 4	Niveau 5
Nombre de brins	6 222 152	2 899 120	2 437 356	1 003 178	566 782
Nombre de sommets géométriques	1 584 064	984 246	753 364	658 659	658 659
Nombre de brins par face	4	5,8	4,9	11,3	6,4
Nombre d'arêtes fictives	0	3 230	468	30 291	13 286
Espace mémoire en méga-octets	178	85	71	87	71
Temps de calcul en secondes	104	94	95	197	138

TAB. 5.6 – Cerveau3, $256 \times 256 \times 83$, 17995 régions.

	Niveau 1	Niveau 2	Niveau 3	Niveau 4	Niveau 5
Nombre de brins	7 758 816	4 218 088	3 549 748	1 604 840	1 041 050
Nombre de sommets géométriques	1 961 598	1 427 914	1 093 744	1 006 657	1 006 657
Nombre de brins par face	4	5,9	5	13,2	8,6
Nombre d'arêtes fictives	0	1 072	404	74 146	38 068
Espace mémoire en méga-octets	222	124	103	133	113
Temps de calcul en secondes	125	121	121	623	510

TAB. 5.7 – Failles, $512 \times 45 \times 121$, 2 régions.

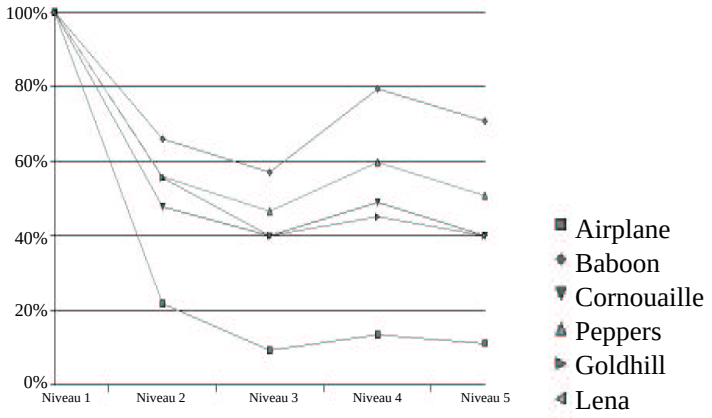
	Niveau 1	Niveau 2	Niveau 3	Niveau 4	Niveau 5
Nombre de brins	8 738 672	4 710 036	3 434 058	1 073 686	706 442
Nombre de sommets géométriques	2 218 060	1 734 460	1 096 471	836 695	836 695
Nombre de brins par face	4	9,6	6,9	12,4	8,2
Nombre d'arêtes fictives	0	28 387	28 387	165 131	93 595
Espace mémoire en méga-octets	250	139	100	112	99
Temps de calcul en secondes	65	47	49	532	421

présentons sur la deuxième ligne de ces tableaux le nombre de sommets géométriques représentés, c'est-à-dire les sommets du plongement de la carte. Cela permet d'étudier pour les trois premiers niveaux de carte le nombre de sommets topologiques, car pour ces niveaux chaque sommet topologique correspond à un sommet géométrique. Mais cela permet également de vérifier que les niveaux 4 et 5 représentent certains de ces sommets de manière redondante. Par exemple, pour l'image du cœur, nous voyons que le niveau 3 représente $12\,183$ sommets géométriques alors que les niveaux 4 et 5 en représentent chacun $12\,928$. Ce nombre devrait normalement rester constant pour ces trois niveaux étant donné qu'ils représentent la même géométrie. Cela provient du plongement face utilisé, où les sommets au bord de ces faces sont représentés de manière redondante. Mais nous voyons que ce nombre de sommets représenté « en trop » n'est pas très important et la perte en espace mémoire est finalement assez faible.

Les deux lignes suivantes donnent le nombre moyen de brins par face topologique, et le nombre d'arêtes fictives. Ces nombres sont intéressants afin d'estimer la complexité de l'algorithme optimal qui parcourt une face afin de savoir si sa fusion va entraîner une déconnexion ou non. Pour le niveau 1, chaque face est forcément composée de 4 brins étant donné qu'aucune fusion de faces n'a encore été effectuée. Ce nombre augmente entre les niveaux 1 et 2 mais aussi entre les niveaux 3 et 4, étant donné que le passage entre ces niveaux effectue des fusions de faces ce qui augmente le nombre moyen de brins par face. Par contre, le passage entre les niveaux 2 et 3 et entre les niveaux 4 et 5 diminue ce nombre étant donné que certaines arêtes sont fusionnées. Pour les cinq images, nous obtenons environ 7 brins par face, ce qui est relativement faible. De ce fait, le test afin de savoir si une fusion va entraîner une déconnexion est effectué en parcourant en moyenne 7 brins, ce qui limite le surcoût dû à cette opération.

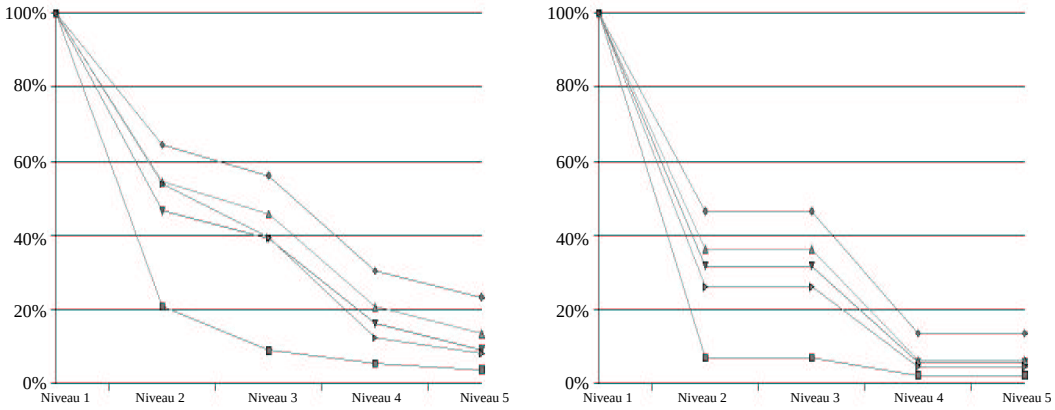
L'avant-dernière ligne donne l'espace mémoire nécessaire à chaque niveau, en tenant compte du plongement ainsi que de l'arbre d'inclusion. Nous constatons une perte en espace mémoire entre les niveaux 3 et 4, dû au changement de plongement. Nous utilisons en effet un plongement 2-G-cartes, qui est relativement coûteux. De plus, ce plongement duplique le plongement des sommets et des arêtes du bord des surfaces. Malgré ça, la carte de niveau 5 ne requiert pas beaucoup d'espace mémoire supplémentaire par rapport à la carte de niveau 3. Enfin le temps d'exécution, donné en dernière ligne, augmente de manière non négligeable entre les niveaux 3 et 4, car nous changeons alors d'algorithme d'extraction. Le niveau 3 est calculé directement de manière optimale, à l'aide des 53 précodes, alors que les niveaux 4 et 5 sont calculés à l'aide de l'algorithme intermédiaire, en utilisant seulement les 8 précodes lignels. De nombreux brins sont donc créés et détruits, ce qui augmente le temps d'exécution, mais nous effectuons également le test de déconnexion de face avant chaque fusion, contrairement à l'algorithme optimal du niveau 3 où il est effectué uniquement pour les trois précodes fc_{16}, fc_{17} et fc_{18}.

Nous récapitulons et regroupons ces résultats sur des schémas qui permettent de mieux visualiser l'évolution de ces caractéristiques en fonction du niveau de simplification. La figure 5.95 montre l'évolution de l'espace mémoire en pourcentage du niveau 1. Nous pouvons vérifier que l'espace mémoire augmente entre les niveaux 3 et 4, à cause du changement de plongement. Mais l'espace mémoire occupé par le niveau 5 est en moyenne du même ordre que celui du niveau 3, malgré le plongement 2-G-carte et les sommets représentés de manière redondante. La figure 5.96.*a* montre l'évolution du nombre de brins en pourcentage du niveau 1 et la figure 5.96.*b* l'évolution du nombre de faces, toujours en pourcentage du niveau 1. Nous remarquons sur cette figure que le nombre de faces reste constant entre les niveaux 2 et 3 et entre les niveaux 4 et 5, étant donné que seules des arêtes sont fusionnées entre ces niveaux. La figure 5.97.*a* présente l'évolution



a. Espace mémoire. b. Légende.

FIG. 5.95 – Évolution de l'espace mémoire en pourcentage du niveau 1.

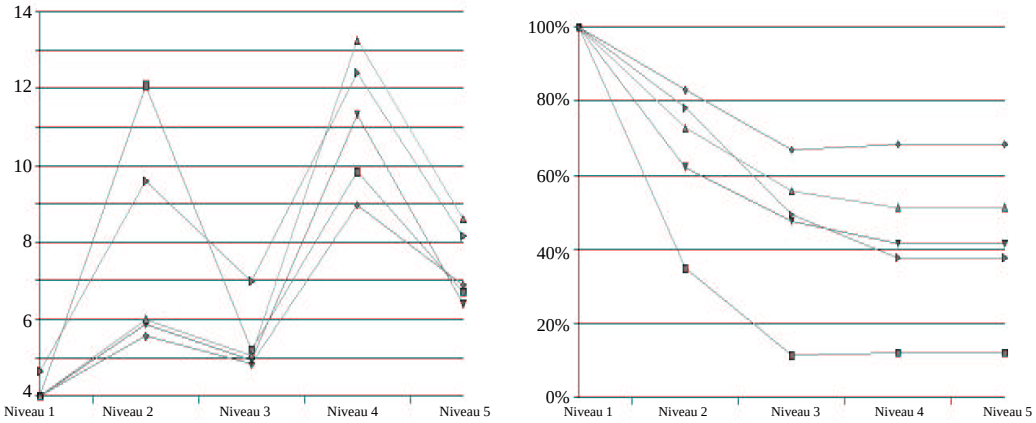


a. Nombre de brins. b. Nombre de faces.

FIG. 5.96 – Évolution du nombre de brins et du nombre de faces en pourcentage du niveau 1.

du nombre moyen de brins par face. Ce nombre est le plus élevé pour le niveau 4, environ de 11 brins par face, mais diminue ensuite pour le niveau 5, pour être en moyenne d'environ 7 brins par face. La figure 5.97.*b* présente l'évolution du nombre de sommets géométriques en pourcentage du niveau 1. Nous vérifions que ce nombre augmente légèrement entre les niveaux 3 et 4 étant donné que nous changeons de plongement, et représentons les sommets se trouvant au bord des faces de manière redondante. Mais cette augmentation est assez faible, en regard du nombre total de sommets représentés.

Nous avons également étudié le nombre de précodes utilisés lors de l'extraction des trois premiers niveaux de simplification. Cela permet de voir quels sont les précodes les plus fréquents. Ce sont ces précodes qu'il va falloir optimiser au maximum afin d'améliorer le temps d'extrac-



a. Nombre moyen de brins par face. b. Nombre de sommets géométriques.

FIG. 5.97 – Évolution du nombre moyen de brins par face et du nombre de sommets géométriques.

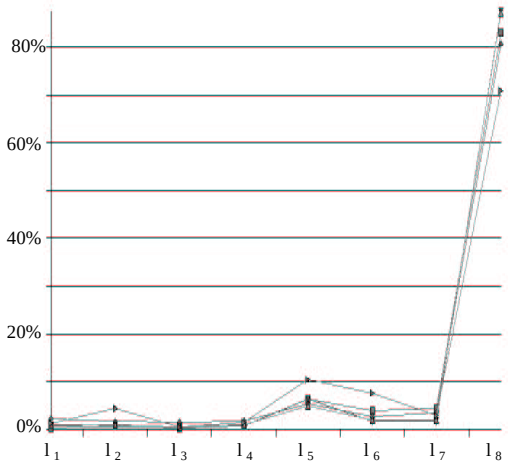
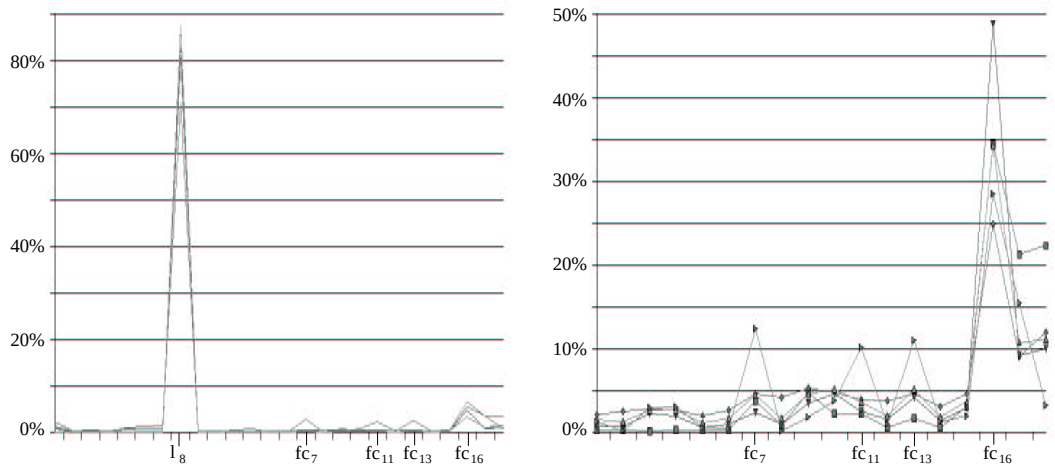


FIG. 5.98 – Nombre de précodes lignels utilisés lors de l'extraction de la carte de niveau 1, en pourcentage du nombre total de précodes.

tion. Nous avons effectué cette étude uniquement pour les trois premiers niveaux étant donné que les deux derniers sont calculés à l'aide de l'algorithme intermédiaire. La figure 5.98 montre le nombre de précodes utilisés pour l'extraction de la carte lignel, en pourcentage du nombre total de précodes dans l'image. Nous voyons que, pour les 5 images utilisées pour nos tests, le précode l_8 apparaît dans environ 80% des précodes totaux de l'image. En effet, les images étant segmentées en régions, la majorité des précodes se trouvent au milieu d'une région, et sont donc traités par le précode l_8. Cela explique la perte de temps d'exécution pour les niveaux 4 et 5, étant donné que le traitement du précode l_8 effectue des tests afin de savoir si des faces ou des arêtes doivent être fusionnées, ce qui n'est pas le cas pour l'algorithme optimal.



a. En pourcentage du nombre total de précodes.

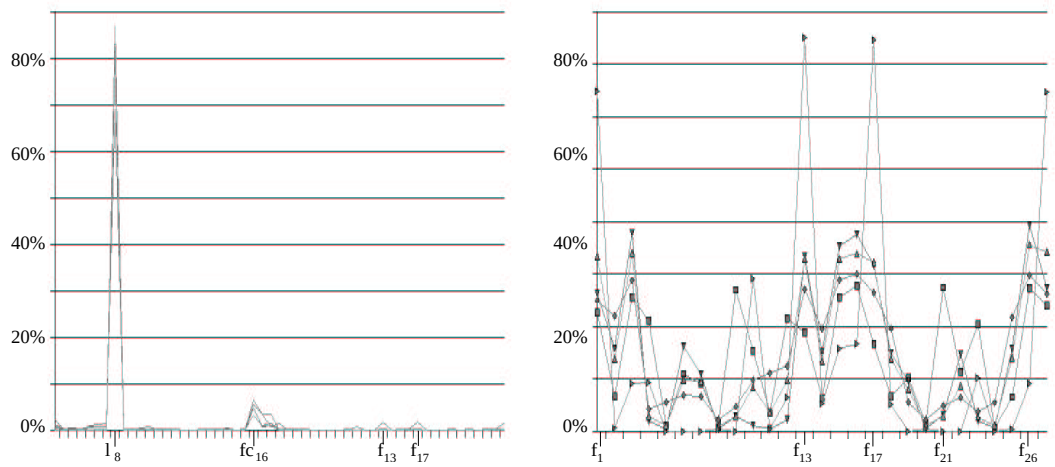
b. En pourcentage du nombre de précodes faces coplanaires.

FIG. 5.99 – Nombre de précodes faces coplanaires utilisés lors de l'extraction de la carte de niveau 2.

La figure 5.99.*a* montre le nombre de précodes utilisés pour l'extraction de la carte lignel, en pourcentage du nombre total de précodes dans l'image. Comme pour le niveau 1, le précode l_8 apparaît dans environ 80% des cas. Les autres pourcentages sont de ce fait tous assez faibles. Les seuls qui ressortent sur les cinq images sont les précodes fc_{16}, fc_{17} et fc_{18}. Ce sont les trois précodes pour lesquels il faut tester la déconnexion de face. Ces précodes apparaissent plus que les autres car ce sont ceux ayant le plus de voxels de même région. Il est donc normal de les retrouver en plus grand nombre que les précodes ayant beaucoup de voxels différents. Afin de mieux voir les précodes utilisés pour extraire la carte de niveau 2, la figure 5.99.*b* montre le pourcentage d'apparition de ces précodes par rapport au nombre total de précodes faces coplanaires. Cette figure confirme l'apparition plus fréquente des précodes fc_{16}, fc_{17} et fc_{18}. Les précodes fc_7, fc_{11} et fc_{13} ont des pourcentages supérieurs aux autres, mais uniquement pour l'image *failles*. Cela n'est donc pas vraiment représentatif. Il faudrait effectuer des tests plus nombreux afin d'obtenir une moyenne sur ces pourcentages d'apparition qui soit représentative.

La figure 5.100.*a* montre le nombre de précodes utilisés pour l'extraction de la carte de niveau 3. Le précode l_8 apparaît toujours dans 80% des cas. Les seuls précodes un peu plus utilisés sont les précodes fc_{16}, fc_{17} et fc_{18} pour la fusion de faces coplanaires, et les précodes f_{13}, f_{17} pour la fusion d'arêtes alignées. Mais ces deux derniers précodes ressortent principalement pour l'image *failles* et pas vraiment pour les autres images. Pour le vérifier, la figure 5.100.*b* présente le pourcentage d'apparition des précodes frontières par rapport au nombre total de ces précodes. Nous voyons sur cette figure que les courbes sont très irrégulières, et même si certains précodes apparaissent plus souvent que d'autres ce n'est pas forcément pour toutes les images.

Ces différences d'apparitions sont assez faibles. Finalement, ce qui ressort de cette étude est que, pour extraire la carte de niveau 3, environ 85% des précodes utilisés sont des précodes lignels, avec 80% uniquement pour le précode l_8. Les précodes faces coplanaires représentent environ 10% des précodes totaux, et les précodes frontières 5%. Cela montre que les précodes les plus impor-



a. En pourcentage du nombre total de précodes.

b. En pourcentage du nombre de précodes frontières.

FIG. 5.100 – Nombre de précodes frontières utilisés lors de l'extraction de la carte de niveau 3.

tants sont finalement les précodes lignels, et principalement le précode l_8. Cela relativise un peu l'impact de l'algorithme optimal et de l'utilisation des précodes pour calculer la carte de niveau 5. Malgré cela, les résultats d'extraction des cartes de niveaux 4 et 5 montrent que la perte en temps d'exécution est non négligeable. En effet, l'utilisation de l'algorithme intermédiaire entraîne des tests afin d'effectuer toutes les fusions nécessaires, mais également des tests de déconnexion. Ces tests sont effectués pour tous les précodes, y compris le précode l_8. C'est pour ce précode que la perte de temps est alors la plus conséquente. L'algorithme optimal se justifie non pas afin de traiter de manière optimale tout les précodes, étant donné le faible pourcentage d'apparition de la majorité d'entre eux, mais car il évite des traitements non optimaux pour le précode l_8 et en ce sens diminue la complexité de l'extraction. En effet, seule l'implantation complète des 5 niveaux de simplification permet, lors du calcul de la carte topologique, de traiter le précode l_8 de manière optimale sans aucun test supplémentaire. Dans ce cas, le gain en temps d'exécution est conséquent étant donné le fort pourcentage d'apparition de ce précode.

5.9 Conclusion

Dans ce chapitre, nous avons défini la carte topologique en dimension 3. Ce modèle combinatoire permet de représenter les images 3d segmentées en régions de manière minimale. Il représente toute la topologie des régions de l'images, (adjacence, incidence, subdivisions...) mais également leur géométrie. Il est stable par rotation, translation et homothétie, c'est-à-dire que deux images isomorphes pour ces transformations auront la même carte topologique, malgré des géométries différentes.

Afin de définir ce modèle, nous avons étendu la notion de niveau de simplification, introduite en dimension 2. Nous avons alors obtenu 5 niveaux de simplification, définis dans un premier temps de manière assez simple. Nous avons ensuite étudié les problèmes que posaient ces pre-

mières définitions. Nous avons rencontré un problème de déconnexion de volume, résolu par l'adjonction d'un arbre d'inclusion des régions, de manière similaire à la dimension 2. Le problème de déconnexion de face a été plus délicat à résoudre. Pour cela, nous avons conservé des arêtes fictives pour que chaque face de la carte soit homéomorphe à un disque topologique.

Ces arêtes fictives permettent de régler le problème de déconnexion de face. Mais elles jouent un rôle différent des arêtes réelles. Nous avons donc étudié précisément leur rôle, les problèmes que leur conservation posaient, et la manière de les régler. Ce point est très important, car c'est la gestion adéquate de ces arêtes qui nous permet d'obtenir la représentation minimale. Nous avons montré, sur plusieurs exemples, comment ces arêtes fictives intervenaient, principalement pour les objets représentés uniquement par une face frontière fermée. En effet, c'est pour ces objets que la gestion des arêtes fictives est la plus importante, étant donné qu'ils sont représentés uniquement par ce type d'arêtes.

Nous avons ensuite étudié la manière de construire ce modèle à partir d'images 3d segmentées. Un premier algorithme naïf peut être défini par application directe des définitions des différents niveaux de simplification. Cet algorithme a l'avantage d'être simple à mettre en œuvre, mais a une complexité quadratique en le nombre de brins de l'image. De plus, il requiert un espace mémoire beaucoup plus important que celui nécessaire à la carte topologique.

Nous avons donc étudié un deuxième algorithme d'extraction, basée sur la notion de précodes. Cet algorithme est optimal, car il crée directement le bon nombre de brins, en utilisant un nombre minimal d'opérations. Sa complexité est linéaire en le nombre de brins de l'image, malgré la constante due au parcours de faces afin de savoir si une fusion va entraîner une déconnexion. Nous avons étudié précisément les précodes à traiter pour chaque niveau de simplification. Cette approche nous a permis de factoriser les précodes similaires, et d'obtenir au final seulement 379 précodes à traiter, au lieu des 4140 existants. De plus, nous avons présenté la notion de précodes isomorphes par rotation, qui permet de factoriser davantage de cas, sans perte en efficacité. Avec cette deuxième factorisation, nous devons définir « seulement » 129 traitements différents. Même si ce nombre est encore important, il est relativement faible en comparaison au nombre total de cas. Enfin, nous avons présenté un troisième algorithme d'extraction, qui est une solution intermédiaire entre les deux premiers algorithmes. Il autorise une implantation assez rapide et simple de l'algorithme d'extraction, tout en restant d'une complexité linéaire. De plus, cet algorithme peut être plus ou moins optimisé en incorporant certains traitements de précodes.

Nous envisageons maintenant la définition de la carte topologique en dimension n. Cela devrait pouvoir se réaliser sans trop de problème, en étendant les solutions proposées en dimension 2 et 3. Nous savons définir les $2n-1$ niveaux de simplification, en mettant dans un premier temps de côté les problèmes de déconnexion. Nous réglons le problème de déconnexion en nd en ajoutant un arbre d'inclusion des régions. Pour les déconnexions de dimensions $n-1$ à 2, il est facile de conserver des éléments fictifs permettant d'avoir uniquement des i-cellules connexes. Enfin, il faut s'intéresser à la gestion particulière de ces éléments fictifs et c'est là que se trouvent les difficultés éventuelles. Intuitivement, cela devrait ressembler à la gestion des arêtes fictives en 3d. Il s'agit de « pousser » les éléments fictifs afin qu'ils n'empêchent pas la fusion d'autres éléments. Il faut s'intéresser à la manière dont ces éléments sont « poussés », et aux conditions pour lesquelles c'est possible. De plus, après avoir donné ces définitions, nous obtenons immédiatement l'algorithme d'extraction naïf. L'algorithme optimal semble difficilement envisageable de par le

nombre important de cas à traiter. Mais l'algorithme intermédiaire pourra être une bonne solution afin de limiter l'espace mémoire nécessaire et conserver une complexité linéaire.

CHAPITRE 6

FUSION ET ÉVOLUTION DES CARACTÉRISTIQUES TOPOLOGIQUES

Chapitre 5, nous avons défini la carte topologique en dimension 3. Cette définition utilise l'opération de fusion, que nous avons présentée et pour laquelle nous avons donné les différents algorithmes. Nous définissons maintenant section 6.1 l'opération de fusion de manière algébrique. Cela permet de prouver que cette opération est valide, et formalise son cadre d'utilisation. Pour cela, nous utilisons le formalisme des cartes généralisées, (présentées section 2.3) afin de profiter de leur homogénéité qui simplifie les différents cas à traiter. Sachant que les cartes combinatoires et les cartes généralisées sont équivalentes lorsque les subdivisions considérées sont orientables et fermées, cela valide cette étude pour les cartes combinatoires fermées. Nous nous intéressons ici uniquement à la dimension 3, la dimension 2 étant plus simple à traiter et qu'elle peut être considérée comme un cas particulier de la dimension 3.

Section 6.2 nous nous intéressons aux fusions dans le cadre de la construction de la carte topologique. Nous avons vu que pour construire cette carte, nous partons de la carte « complète », le niveau 0, où chaque volume représente un voxel de l'image. Nous effectuons ensuite certaines fusions, suivant les contraintes déjà présentées, et arrivons finalement à la carte topologique. Nous allons considérer la suite des fusions effectuées pour arriver à cette carte, et montrer qu'il est possible de la « réorganiser » afin de commencer par les fusions de plus grande dimension, tout en arrivant finalement au même résultat.

Cette possibilité de « réorganisation » est importante, car elle nous permet section 6.3 d'étudier l'évolution des caractéristiques topologiques lors de l'opération de fusion, en étudiant un nombre de cas raisonnable. Cela nous permet de prouver que la carte topologique conserve bien toute les informations topologiques des objets qu'elle représente, ce qui est primordial pour valider ce modèle. Cela nous permet également de prouver que les 5 niveaux de simplification représentent tous les mêmes informations topologiques. Enfin, cette étude exhaustive des cas pouvant se produire pour l'opération de fusion permet de nous assurer d'avoir traité tous les cas de déconnexion possibles.

6.1 Définition algébrique de l'opération de fusion

Nous avons déjà présenté l'opération de fusion lors de la définition de la carte topologique en dimensions 2 et 3. Nous rappelons ici brièvement les points importants. L'opération de fusion consiste à remplacer une ou deux $(i+1)$-cellules adjacentes par une seule $(i+1)$-cellule représentant l'union des deux. Cette modification s'effectue en supprimant la i-cellule incidente aux deux $(i+1)$-cellules, et en mettant à jour les liaisons entre ces cellules. Nous parlons ici de fusion de dimension i lorsque nous supprimons une i-cellule, et disons que nous fusionnons les deux $(i+1)$-cellules le long de cette i-cellule. De manière générale, en dimension n, il existe exactement $n-1$ types de fusions possibles, qui sont les fusions de dimension 0 à $n-1$.

Pour pouvoir effectuer la fusion de deux $(i+1)$-cellules le long d'une i-cellule, nous fixons des contraintes sur cette i-cellule afin de pouvoir garantir la validité de la fusion. Cette i-cellule doit être *localement de degré deux*. Intuitivement, le degré local d'une i-cellule est le nombre de demi-$(i+1)$-cellules qui lui sont incidentes. L'avantage de cette notion est de regrouper les deux cas où la i-cellule est de degré 1 ou 2. Cela évite de les différencier, sachant que la fusion est identique dans les deux cas.

Définition 33 (Cellule localement de degré deux) *Soit une i-cellule $C =< \{\alpha_0, \alpha_1, \alpha_2, \alpha_3\} - \alpha_i > (b)$, $0 \leq i \leq 2$. Nous notons*
- $E_1 =< \{\alpha_0, \alpha_1, \alpha_2, \alpha_3\} - \{\alpha_i, \alpha_{(i+1)}\} > (b)$;
- $E_2 =< \{\alpha_0, \alpha_1, \alpha_2, \alpha_3\} - \{\alpha_i, \alpha_{(i+1)}\} > (\alpha_{(i+1)}(b))$.

La i-cellule C est localement de degré deux *ssi*

1. $E_1 \cup E_2 = C$;
2. $E_1 \cap E_2 = \emptyset$;
3. $\forall b \in C$,
   - *Si* $i = 0$, $\alpha_{12}(b) = \alpha_{21}(b)$;
   - *Si* $i = 1$, $\alpha_{23}(b) = \alpha_{32}(b)$.

De manière intuitive, les deux ensembles E_1 et E_2 correspondent aux deux demi-i-cellules incidentes à C. Si C est localement de degré deux, alors ces deux demi-i-cellules doivent être disjointes, sinon C serait localement de degré 1, et leur union doit être C, sinon C serait localement de degré supérieur à 2. La troisième contrainte est nécessaire afin d'être sûr de pouvoir effectuer cette fusion, et conserver les contraintes des G-cartes après celle-ci. Remarquons que C, E_1 et E_2 ne peuvent jamais être des ensembles vides, étant donné qu'ils contiennent au moins chacun le brin b. Par la suite, nous notons α_j et α_k les deux éléments de $\{\alpha_0, \alpha_1, \alpha_2, \alpha_3\} - \{\alpha_i, \alpha_{(i+1)}\}$, ce qui permet de simplifier l'écriture des deux ensembles E_1 et E_2. Nous présentons maintenant quelques propriétés découlant de la définition des cellules localement de degré deux. Ces propriétés sont nécessaires pour la preuve de la validité de l'opération de fusion.

Remarque *Soit C une i-cellule localement de degré deux. Pour tout brin b de C nous avons $\alpha_{(i+1)}\alpha_j(b) = \alpha_j\alpha_{(i+1)}(b)$ et $\alpha_{(i+1)}\alpha_k(b) = \alpha_k\alpha_{(i+1)}(b)$.*

Cela peut se vérifier exhaustivement pour les trois différentes valeurs de i possibles.
- Pour $i = 0$: $\alpha_j = \alpha_2$ et $\alpha_k = \alpha_3$. Alors, $\alpha_1\alpha_2 = \alpha_2\alpha_1$ est vérifiée par la définition des i-cellules localement de degré 2, et $\alpha_1\alpha_3 = \alpha_3\alpha_1$ est toujours vérifiée par définition des G-cartes.

- Pour $i = 1 : \alpha_j = \alpha_0$ et $\alpha_k = \alpha_3$. Alors $\alpha_2\alpha_0 = \alpha_0\alpha_2$ est vraie par définition des G-cartes, et $\alpha_2\alpha_3 = \alpha_3\alpha_2$ est donnée par la définition des i-cellules localement de degré 2.
- Pour $i = 2 : \alpha_j = \alpha_0$ et $\alpha_k = \alpha_1$. Les deux contraintes $\alpha_3\alpha_0 = \alpha_0\alpha_3$ et $\alpha_3\alpha_1 = \alpha_1\alpha_3$ sont toujours vraies par définition des G-cartes.

Propriété 7 *Soit C une i-cellule localement de degré 2. Alors $\forall b \in C$, b n'est pas $(i+1)$-libre.*

Preuve de la propriété 7 *Soit $C =< \{\alpha_0, \alpha_1, \alpha_2, \alpha_3\} - \alpha_i > (b)$ une i-cellule localement de degré 2. Si b est $(i+1)$-libre, alors nous avons $E_1 = E_2$ ce qui contredit $E_1 \cap E_2 = \emptyset$. Comme $\alpha_{(i+1)}\alpha_j(b) = \alpha_j\alpha_{(i+1)}(b)$, si $\alpha_j(b) \neq b$, alors $\alpha_j(b)$ n'est pas $(i+1)$-libre. Il en est de même pour $\alpha_k(b)$. Nous pouvons effectuer cette démonstration de proche en proche, et finalement prouver que chaque brin de E_1 et de E_2 n'est pas $(i+1)$-libre, et donc que chaque brin de C ne l'est pas non plus.* □

Propriété 8 *Soit C une i-cellule localement de degré 2. Alors $\forall b \in C$, nous avons $b \in E_1 \Rightarrow \alpha_{(i+1)}(b) \in E_2$ et $b \in E_2 \Rightarrow \alpha_{(i+1)}(b) \in E_1$.*

Preuve de la propriété 8 *Soit b_1 un brin de E_1. Nous avons $E_1 =< \alpha_j, \alpha_k > (b)$, et $E_2 =< \alpha_j, \alpha_k > (\alpha_{(i+1)}(b))$. Supposons $b_2 = \alpha_{(i+1)}(b_1) \in E_1$. Alors $\alpha_j(b_2) \in E_1$ par définition de E_1. Étant donné que $\alpha_{(i+1)}\alpha_j(b_1) = \alpha_j\alpha_{i+1}(b_1)$, comme $\alpha_j(\alpha_{(i+1)}(b_1)) \in E_1$, alors $\alpha_{(i+1)}(\alpha_j(b_1)) \in E_1$. De manière similaire, nous pouvons montrer que $\alpha_{(i+1)}(\alpha_k(b_1)) \in E_1$. En effectuant ce raisonnement de proche en proche, nous montrons que $\forall b \in E_1$, nous avons $\alpha_{(i+1)}(b) \in E_1$. Dans ce cas, comme $C =< \alpha_j, \alpha_k, \alpha_{(i+1)} > (b)$, nous avons $E_1 = C$, et comme $E_2 \neq \emptyset$ cela contredit $E_1 \cap E_2 = \emptyset$ et le fait que C est une i-cellule localement de degré 2. Nous pouvons effectuer cette démonstration de manière symétrique pour montrer que $\forall b \in E_2$, $\alpha_{(i+1)}(b) \in E_1$.* □

Remarque *Soit C une i-cellule localement de degré 2. Alors $\forall b \in C$, si nous notons*
- $E'_1 =< \{\alpha_0, \alpha_1, \alpha_2, \alpha_3\} - \{\alpha_i, \alpha_{(i+1)}\} > (b)$
- $E'_2 =< \{\alpha_0, \alpha_1, \alpha_2, \alpha_3\} - \{\alpha_i, \alpha_{(i+1)}\} > (\alpha_{(i+1)}(b))$

alors nous avons ($E'_1 = E_1$ et $E'_2 = E_2$) ou ($E'_1 = E_2$ et $E'_2 = E_1$).

Cette remarque permet, lorsqu'une i-cellule C est localement de degré 2, de montrer que le brin de départ pour calculer les deux demi-cellules n'est pas important, et que la propriété est homogène. Elle peut se prouver facilement, en regardant que pour C une i-cellule localement de degré 2, $\forall b \in C$, soit $b \in E_1$, soit $b \in E_2$.

- Si $b \in E_1$, alors $E'_1 = E_1$ par définition des orbites. Étant donné que nous savons que $\alpha_{(i+1)}(b) \in E_2$, car C est une i-cellule localement de degré 2, nous avons également $E'_2 = E_2$.
- Si $b \in E_2$, nous pouvons faire un raisonnement similaire pour montrer que $E'_1 = E_2$ et $E'_2 = E_1$.

Nous pouvons maintenant définir, de manière formelle et générale, l'opération de fusion.

Définition 34 (Fusion de deux (i+1)-cellules) *Soit $G = (B, \alpha_0, \alpha_1, \alpha_2, \alpha_3)$ une 3-G-carte, $b \in B$ un brin et $i \in \{0, 1, 2\}$, tel que $C =< \{\alpha_0, \alpha_1, \alpha_2, \alpha_3\} - \alpha_i > (b)$, la i-cellule incidente au brin b, soit localement de degré deux. Nous notons $B^2 = \alpha_i(C) - C$ les brins cousus par α_i aux brins de C et n'appartenant pas à C. La* fusion *des deux $(i+1)$-cellules adjacentes le long de C est la 3-G-carte $G' = (B', \alpha'_0, \alpha'_1, \alpha'_2, \alpha'_3)$ définie par :*

- $B' = B - C$;
- $\forall j \in \{0,1,2,3\} - i$, $\alpha'_j = \alpha_j|_{B'}$;
- $\forall b \in B' - B^2$, $\alpha'_i(b) = \alpha_i(b)$;
- $\forall b \in B^2$, ***si*** $\exists b' \in B^2$ *tel que* $b' = (\alpha_{(i+1)}\alpha_i)^k(\alpha_i(b))$, ***alors*** *on pose* $\alpha'_i(b) = (\alpha_{(i+1)}\alpha_i)^{k'}(\alpha_i(b))$ *avec* k' *étant égal au plus petit* k ; ***sinon*** *on pose* $\alpha'_i(b) = b$.

Cette fusion est valable pour toute 3-G-carte, à la seule condition que la i-cellule C soit localement de degré 2. Les notations utilisées pour cette définition sont :

- $G = (B, \alpha_0, \alpha_1, \alpha_2, \alpha_3)$ est la 3-G-carte de départ ;
- C est la i-cellule à supprimer ;
- B^2 est l'ensemble des brins cousus par α_i aux brins de C et n'appartenant pas à C ;
- $G' = (B', \alpha'_0, \alpha'_1, \alpha'_2, \alpha'_3)$ est la 3-G-carte résultant de l'opération de fusion.

Seule l'involution α_i est modifiée pour les brins appartenant à B^2, les autres brins conservent les mêmes coutures. Pour les brins appartenant à B^2, nous différencions deux cas suivant qu'il existe ou non $b' \in B^2$ tel que $b' = (\alpha_{(i+1)}\alpha_i)^k(\alpha_i(b))$. Le deuxième cas permet de traiter les configurations où il n'existe pas un brin image pour α'_i. Ce cas survient uniquement pour les G-cartes ouvertes, et peut être supprimé si nous nous limitons aux G-cartes fermées.

Théorème 1 *Soit G une 3-G-carte et C une i-cellule localement de degré 2, alors G' obtenue par fusion des deux $(i+1)$-cellules adjacentes le long de C est une 3-G-carte.*

Preuve du théorème 1

1. ***pour*** $\boldsymbol{\alpha_j \neq \alpha_i}$ *:* *Soit $b_1 \in B'$. Montrons que $b_2 = \alpha_j(b_1) \in B'$. Supposons que $b_2 \notin B'$. Alors, étant donné que $B' = B - C$ nous avons $b_2 \in C$. Étant donné que $C =< \{\alpha_0, \alpha_1, \alpha_2, \alpha_3\} - \alpha_i > (b)$, nous pouvons dire que si $b_2 \in C$ alors $b_1 = \alpha_j(b_2) \in C$, ce qui est en contradiction avec $b_1 \in B'$.*
2. ***pour*** $\boldsymbol{\alpha_i}$ *:*
   (a) ***pour*** $\boldsymbol{b_1 \in B' - B^2}$ *: Montrons que $b_2 = \alpha_i(b_1) \in B'$, par le même raisonnement que pour le cas précédent. Supposons que $b_2 \notin B'$. Alors nous avons $b_2 \in C$. Dans ce cas, $b_1 = \alpha_i(b_2) \in B^2$ ou $\in C$ par définition de B^2 ce qui, dans les deux cas, est en contradiction avec $b_1 \in B' - B^2$.*
   (b) ***pour*** $\boldsymbol{b_1 \in B^2}$ *: Nous effectuons une démonstration générique, qui sera ensuite instanciée pour les trois différentes fusions possibles. Soit $\alpha_l \in \{\alpha_0, \alpha_1, \alpha_2, \alpha_3\} - \{\alpha_i, \alpha_{(i+1)}\}$. Nous montrons que si nous avons $\alpha_{il} = \alpha_{li}$ pour tout brin, et $\alpha_{(i+1)l} = \alpha_{l(i+1)}$ pour les brins de C, alors nous avons $\alpha'_{il}(b_1) = \alpha'_{li}(b_1)$. Notons $b_2 = \alpha_l(b_1)$. Nous avons $b'_1 = \alpha_i(b_1) \in C$ par définition de B_2 et $\alpha_l(b'_1) \in C$ par définition de C. Étant donné que $\alpha_{il}(b_1) = \alpha_{li}(b_1)$, nous avons donc $\alpha_l(b'_1) = \alpha_i(b_2)$ et donc $b'_2 = \alpha_i(b_2) \in C$. Comme C est localement de degré deux, alors nous avons $\alpha_{(i+1)l} = \alpha_{l(i+1)}$. Comme nous avons également $\alpha_{il} = \alpha_{li}$, alors nous obtenons que pour tout brin de C, $\alpha_{(i+1)i} \circ \alpha_l = \alpha_l \circ \alpha_{(i+1)i}$. Comme $b'_1 \in C$, nous obtenons $\alpha_{(i+1)il}(b'_1) = \alpha_{l(i+1)i}(b'_1) = \alpha_{(i+1)i}(b'_2)$. Si $\alpha_{(i+1)i}(b'_1) \in C$ alors nous pouvons reprendre ce même raisonnement afin de montrer que $\alpha_{(i+1)i(i+1)il}(b'_1) = \alpha_{l(i+1)i(i+1)i}(b'_1)$, et au final pour prouver que $(\alpha_{(i+1)i})^k \circ \alpha_l(b'_1) = \alpha_l \circ (\alpha_{(i+1)i})^k(b'_1)$, où k est le plus petit entier tel que $(\alpha_{(i+1)i})^{(k-1)}(b'_1) \in C$ et $(\alpha_{(i+1)i})^k(b'_1) \notin C$. Si un tel k existe, alors c'est le même que pour la définition de α'_i et nous avons donc bien prouvé $\alpha'_{il}(b_1) = \alpha'_{li}(b_1)$.*

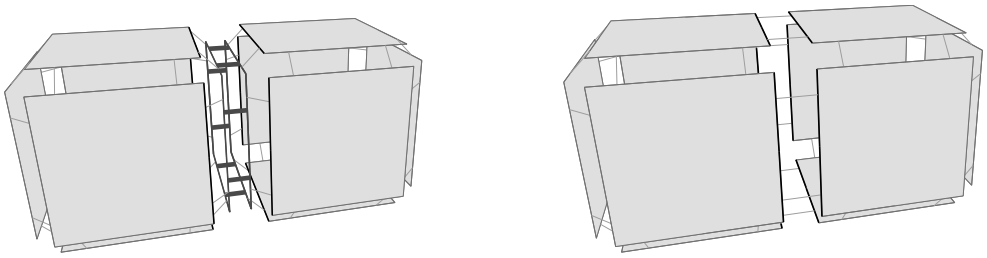


FIG. 6.1 – La fusion de volumes.

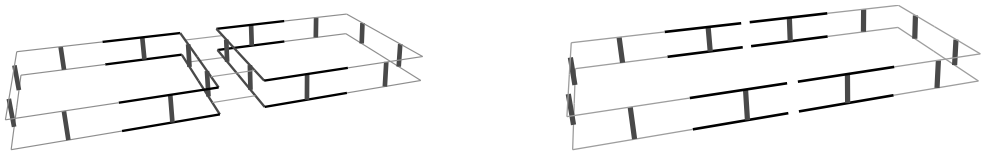


FIG. 6.2 – La fusion de faces.

Si un tel k n'existe pas, alors nous avons un brin bb de C libre pour α_i (car les brins de C sont tous cousu pour $\alpha_{(i+1)}$), tel que $(\alpha_{(i+1)i})^k(b'_1) = bb$. Alors pour ce k, nous avons prouvé que $(\alpha_{(i+1)i})^k \circ \alpha_l(b'_1) = \alpha_l \circ (\alpha_{(i+1)i})^k(b'_1)$. Ce qui peut s'écrire autrement par $(\alpha_{(i+1)i})^k \circ \alpha_l(b'_1) = \alpha_{(i+1)i})^k(b'_2)$. Si $(\alpha_{(i+1)i})^k(b'_1) = bb \in C$, alors $\alpha_{(i+1)i})^k(b'_2) \in C$ et donc nous aurons $\alpha'_l(b_1) = b_1$ et $\alpha'_l(b_2) = b_2$, et nous aurons donc bien $\alpha'_{il}(b_1) = \alpha'_{li}(b_1)$.

- $\boldsymbol{i = 0}$ *:*
  - *Pour $\alpha_l = \alpha_2$, nous avons bien $\alpha_{02} = \alpha_{20}$ pour chaque brin, par définition des G-cartes, et pour les brins de C : $\alpha_{12} = \alpha_{21}$ car C est localement de degré deux. Nous obtenons donc que $\alpha'_{02}(b_1) = \alpha'_{20}(b_1)$.*
  - *Pour $\alpha_l = \alpha_3$, nous avons $\alpha_{03} = \alpha_{30}$ et $\alpha_{13} = \alpha_{31}$ pour tout brin, par définition des G-cartes. Nous obtenons bien $\alpha'_{03}(b_1) = \alpha'_{30}(b_1)$.*
- $\boldsymbol{i = 1}$ *: Pour $\alpha_l = \alpha_3$, nous avons $\alpha_{13} = \alpha_{31}$ pour chaque brin, par définition des G-cartes, et pour les brins de C : $\alpha_{23} = \alpha_{32}$ car C est localement de degré deux. Nous prouvons donc que $\alpha'_{13}(b_1) = \alpha'_{31}(b_1)$.*
- $\boldsymbol{i = 2}$ *: Pour $\alpha_l = \alpha_0$, nous avons $\alpha_{20} = \alpha_{02}$ et $\alpha_{30} = \alpha_{03}$ pour tout brin, par définition des G-cartes. Nous prouvons donc $\alpha'_{20}(b_1) = \alpha'_{02}(b_1)$.*

Lorsque nous regroupons toutes ces informations, nous obtenons, pour les α' égaux à α restreints à B', que les contraintes des G-cartes sont toujours vérifiées, étant donné que nous conservons les mêmes coutures. Pour α'_i, nous avons prouvé, pour chacune des trois fusions possibles, que les différentes contraintes des G-cartes étaient toujours vérifiées. Cela prouve donc que G' est une 3-G-carte. □

Nous pouvons voir trois exemples de fusion, un pour chaque dimension de C, présentés figures 6.1, 6.2 et 6.3. Sur ces figures, nous avons représenté α_2 en gris clair et α_3 en traits épais et foncés. α_0 et α_1 ne sont pas représentés afin de ne pas surcharger les figures, mais peuvent se

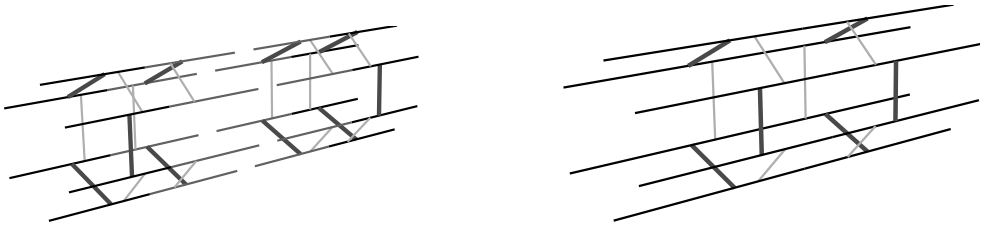


FIG. 6.3 – La fusion d'arêtes.

retrouver sans difficulté de par la manière dont nous dessinons les G-cartes. Enfin, nous avons dessiné en foncé les brins appartenant à C, la cellule que nous supprimons, et en noir les brins appartenant à B_2, brins pour lesquels nous modifions α'_i. Les autres brins sont de couleur plus claire étant donné qu'ils ne sont pas concernés pas la fusion. Ces figures permettent de vérifier que seuls les α'_i-coutures des brins de B_2 sont modifiées. De plus, nous pouvons vérifier que la définition algébrique de la fusion modifie bien la carte initiale pour obtenir le résultat de la fusion.

6.2 Les fusions pour la définition de la carte topologique

La carte topologique en dimension 3 est la carte minimale codant les frontières intervoxel d'une image 3d segmentée en régions. Nous l'avons définie à partir de la carte de niveau 0, où chaque volume représente un voxel de l'image, plus un volume représentant la région infinie entourant ces voxels. À partir de cette carte, nous effectuons les fusions de toutes les cellules adjacentes incidentes à une cellule localement de degré deux. Nous notons $(f_1, \ldots, f_k)$ la suite des fusions effectuées à partir de la carte complète pour obtenir la carte topologique. Chaque fusion s'effectue le long d'une cellule localement de degré deux. Nous notons $d(f_i)$ la dimension de la cellule supprimée par la fusion f_i, qui appartient à $\{0, 1, 2\}$, et $C(f_i)$ cette cellule.

Pour la définition de la carte topologique présentée section 5.2, nous commençons par fusionner les volumes de mêmes régions, puis les faces coplanaires, les arêtes alignées, les faces non coplanaires et enfin les arêtes alignées. Ce choix de l'ordre et du type de cellules fusionnées nous permet d'obtenir la carte des frontières, qui peut être plongée uniquement par ses sommets, comme un niveau intermédiaire de simplification. Mais il est possible de réorganiser ces fusions de manière différente, en obtenant finalement la même carte topologique.

Théorème 2 *Soit G une 3-G-carte et $(f_1, \ldots, f_k)$ une suite de fusions effectuées à partir de G, et G' la G-carte résultant de ces fusions. Alors il existe une suite de fusions $(f_{2_1}, \ldots, f_{2_k}, f_{1_1}, \ldots, f_{1_l}, f_{0_1}, \ldots, f_{0_m})$ qui, partant de G produit également G', et vérifiant $\forall f_{i_j}$, $d(f_{i_j}) = i$.*

En d'autres termes, cela revient à dire que, pour toute suite de fusions, il existe une suite de fusions ordonnées produisant la même G-carte : nous commençons par faire des fusions de volumes, puis des fusions de faces et enfin des fusions d'arêtes. Pour prouver ce théorème, nous donnons des règles de réécriture, qui nous permettent de modifier progressivement une suite de fusions jusqu'à obtenir une suite ordonnée. Soit deux fusions consécutives f_i et f_{i+1} tel que $d(f_i) < d(f_{i+1})$. Il y a trois cas différents.

1. $\boldsymbol{d(f_i) = 0}$ **et** $\boldsymbol{d(f_{i+1}) = 1}$. Si l'arête résultant de la fusion f_i n'est pas $C(f_{i+1})$, alors nous échangeons simplement f_i et f_{i+1} ; sinon nous remplaçons ces deux fusions par deux fusions de faces le long des deux arêtes qui étaient fusionnées par f_i.

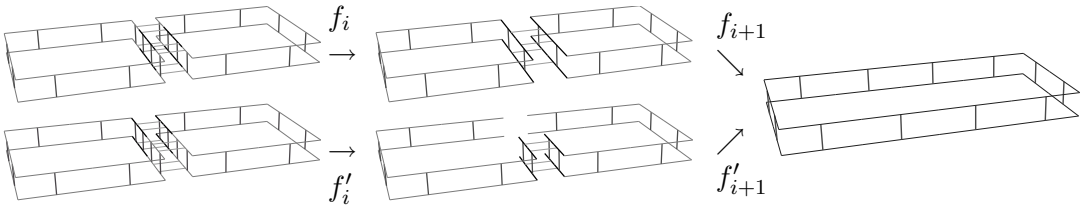


FIG. 6.4 – Avant : f_i est une fusion d'arêtes, et f_{i+1} fusion de faces. Après : f'_i et f'_{i+1} sont deux fusions de faces.

2. $\boldsymbol{d(f_i) = 0}$ **et** $\boldsymbol{d(f_{i+1}) = 2}$. Nous pouvons échanger ces deux fusions sans aucune modification, à condition que la fusion f'_{i+1} ne détruise pas totalement les deux arêtes devant être fusionnées par f_i. En effet, dans ce cas, nous n'avons plus besoin d'effectuer cette deuxième fusion, et nous remplaçons donc ces deux fusions par f'_{i+1}.
3. $\boldsymbol{d(f_i) = 1}$ **et** $\boldsymbol{d(f_{i+1}) = 2}$. Si la face résultant de la fusion f_i n'est pas $C(f_{i+1})$, alors nous échangeons simplement f_i et f_{i+1} ; sinon nous remplaçons ces deux fusions pas deux fusions de volumes le long des deux faces qui étaient fusionnées par f_i.

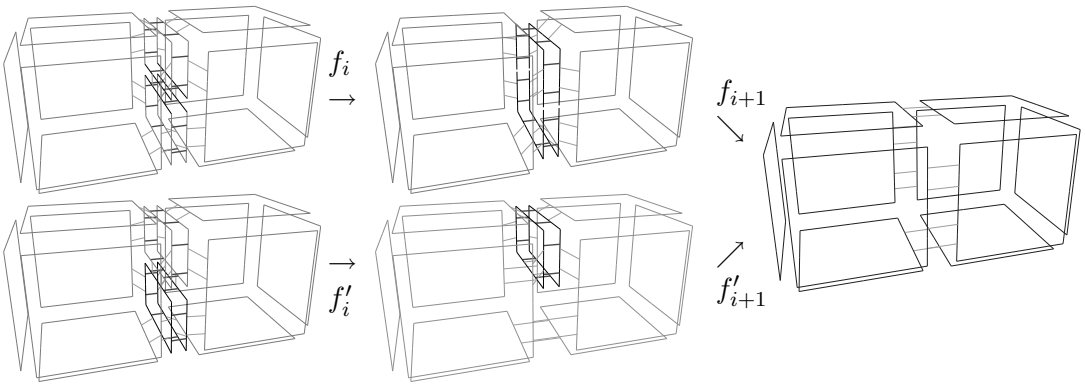


FIG. 6.5 – Avant : f_i est une fusion de faces, et f_{i+1} fusion de volumes. Après : f'_i et f'_{i+1} sont deux fusions de volumes.

Nous montrons maintenant que ces changements ne provoquent aucune modification sur la 3-G-carte obtenue au final. Nous donnons ici seulement les idées de preuves, car la preuve formelle serait longue et technique.

Preuve du théorème 2 *Soit G une 3-G-carte et $(f_1, \ldots, f_k)$ une suite de fusions. Considérons deux fusions consécutives, f_i et f_{i+1} tel que $d(f_i) < d(f_{i+1})$. S'il n'en existe pas, alors la suite de fusions est ordonnée. Sinon, nous modifions ces deux fusions suivant les trois règles de réécriture données ci-dessus. Nous montrons que la G-carte résultant des deux fusions initiale est la même que celle obtenue après avoir modifié ces deux fusions.*

1. $\boldsymbol{d(f_i) = 0}$ *et* $\boldsymbol{d(f_{i+1}) = 1}$. *$C(f_i)$ est un sommet et $C(f_{i+1})$ est une arête. La première fusion consiste à fusionner deux arêtes en une seule, que nous notons A. Si $A \neq C(f_{i+1})$,*

alors nous pouvons échanger sans aucune modification les deux fusions. En effet, au cours de la fusion f_{i+1}, la seule arête supprimée est $C(f_{i+1})$, les autres ne sont pas modifiées. Les deux arêtes fusionnées par f_i existeront et seront identiques si nous commençons à fusionner f_{i+1} puis f_i. Il faut donc étudier le cas où $A = C(f_{i+1})$. Ce cas est présenté figure 6.4, où la deuxième fusion (fusion de faces) s'effectue le long de l'arête résultant de la première fusion. Nous pouvons voir sur cette figure, que ces deux fusions peuvent sans problème se remplacer par deux fusions de faces consécutives le long des deux arêtes qui étaient fusionnées par la première fusion. Nous obtiendrons bien la même 3-G-carte au final que celle obtenue en effectuant les deux fusions f_i et f_{i+1}. De manière intuitive, les coutures modifiées par la fusion f_i concernent des brins qui seront ensuite détruits pas la deuxième fusion.

2. $\boldsymbol{d(f_i) = 0}$ ***et*** $\boldsymbol{d(f_{i+1}) = 2}$. *$C(f_i)$ est un sommet et $C(f_{i+1})$ est une face. Si l'arête résultant de la fusion f_i n'est pas incidente à la face $C(f_{i+1})$, alors nous pouvons intervertir les deux fusions sans aucune modification, car elles sont totalement indépendantes. Sinon, nous pouvons également intervertir les deux fusions sans difficulté. En effet, si nous commençons par la fusion de volumes, la face $C(f_{i+1})$ sera supprimée de la G-carte. Si les deux arêtes fusionnées par f_i existent encore après cette fusion, alors nous pouvons les fusionner par la même fusion que f_i, il y aura juste moins de brins étant donné que ceux appartenant à $C(f_{i+1})$ ont été détruits. Sinon, l'arête entière a été détruite et nous n'avons plus besoin d'effectuer la fusion d'arêtes.*

3. $\boldsymbol{d(f_i) = 1}$ ***et*** $\boldsymbol{d(f_{i+1}) = 2}$. *$C(f_i)$ est une arête et $C(f_{i+1})$ est une face. La première fusion consiste à fusionner deux faces en une seule, que nous notons F. Si $F \neq C(f_{i+1})$, alors nous pouvons échanger sans aucune modification les deux fusions. En effet, au cours de la fusion f_{i+1}, la seule face supprimée est $C(f_{i+1})$, les autres ne sont pas modifiées. Les deux faces fusionnées par f_i existeront et seront identiques si nous commençons à fusionner f_{i+1} puis f_i. Il faut donc étudier le cas où $F = C(f_{i+1})$. Ce cas est présenté figure 6.5, où la deuxième fusion (fusion de volume) s'effectue le long de la face résultant de la première fusion. Nous pouvons voir sur cette figure, que ces deux fusions peuvent sans problème se remplacer par deux fusions de volumes consécutives le long des deux faces qui étaient fusionnées par la première fusion. Nous obtiendrons bien la même 3-G-carte au final que celle obtenue en effectuant les deux fusions f_i et f_{i+1}.*

Ces trois règles de réécriture permettent, à partir de n'importe quelle suite de fusions $(f_1, \ldots, f_k)$, de définir progressivement des nouvelles suites de fusions, chacune produisant au final la même 3-G-carte. Étant donné qu'à chaque étape nous supprimons un cas où deux fusions consécutives sont dans le « mauvais ordre », et que nous n'augmentons pas le nombre d'éléments dans la suite de fusions, nous pouvons garantir que notre processus de réécriture va s'arrêter et produire au final une suite de fusions correctement ordonnée, et produisant la même 3-G-carte que la première suite de fusions. □

6.3 L'évolution des caractéristiques topologiques

Nous étudions maintenant l'évolution des caractéristiques topologiques lors des différentes fusions. Pour cela, nous regardons la G-carte que nous avons avant et après une fusion, et étudions l'évolution de ces caractéristiques. Nous utilisons pour cela la formule d'Euler

$$\#F - \#A + \#S = 2 - 2G$$

où $\#F$ est le nombre de faces, $\#A$ le nombre d'arêtes, $\#S$ le nombre de sommets et G le genre. Elle permet de caractériser un volume, en étudiant le genre de sa surface. Dans la suite de ce travail, nous notons $\#F$, $\#A$ et $\#S$ le nombre de cellules correspondantes et G le genre du volume avant la fusion, et $\#nF$, $\#nA$ et $\#nS$ et nG ces mêmes nombres après la fusion.

Nous nous plaçons dans le cadre des cartes topologiques[1] sachant que n'importe quelle carte topologique peut s'obtenir à partir de la carte complète en effectuant une suite de fusions $(f_1, \ldots, f_k)$ ordonnées, nous traitons d'abord les fusions de volumes, puis les fusions de faces et enfin les fusions d'arêtes. Cet ordre sur la suite de fusions permet de limiter le nombre de configurations possibles. Par exemple, pour étudier la fusion de volumes, nous savons que toutes les faces de la G-carte sont carrées et composées de 8 brins.

La formule d'Euler peut se calculer très simplement sur les 3-G-cartes. Comme elle permet de caractériser uniquement les volumes par leurs surfaces, elle ne prend pas en compte les faces cousues par α_3, et nous comptons donc deux faces dans ce cas. La formule d'Euler peut donc se réécrire par :

$$z(\alpha_0, \alpha_1) - z(\alpha_0, \alpha_2) + z(\alpha_1, \alpha_2) = 2 - 2G$$

où $z(\alpha_i, \alpha_j)$ est le nombre d'orbites $< \alpha_i, \alpha_j >$. Cette formule est applicable uniquement pour un seul et même volume, les différents nombres d'orbites étant calculés sur les brins de ce volume.

Afin de montrer que les 5 niveaux de simplification représentent les mêmes informations topologiques, nous montrons que le genre de chaque objet reste invariant après chaque fusion de faces et d'arêtes. En effet, ce sont les deux seules fusions utilisées pour passer de la carte de niveau 1 aux cinq autres niveaux. L'étude sur la fusion de volumes, utilisée afin de construire la carte de niveau 1 à partir du niveau 0, permet de vérifier que nous avons bien traité correctement toutes les configurations pouvant se produire lors de cette fusion, et que nous n'avons pas oublié de cas de déconnexion.

6.3.1 Fusion de volumes

Nous fusionnons deux volumes v_1 et v_2 le long de la face C incidente aux deux volumes et localement de degré deux. De par l'ordre sur les fusions, nous n'avons encore effectué ni fusion de faces ni fusion d'arêtes. Nous avons donc exclusivement des faces « carrées », c'est-à-dire composées de huit brins de même longueur. Il y a deux cas différents, suivant que $v_1 = v_2$ ou $v_1 \neq v_2$, et dans le premier cas plusieurs cas selon la configuration des arêtes de la face C. Ce sont les différents cas où une ou plusieurs de ces arêtes sont de degré un que nous étudions. Comme la face est carrée, il y a seulement 5 cas différents.

[1] Mais toujours en utilisant le formalisme des G-cartes.

Cas 1 : $v_1 \neq v_2$.

C'est le cas le plus simple où nous fusionnons deux volumes différents. Il est facile de voir

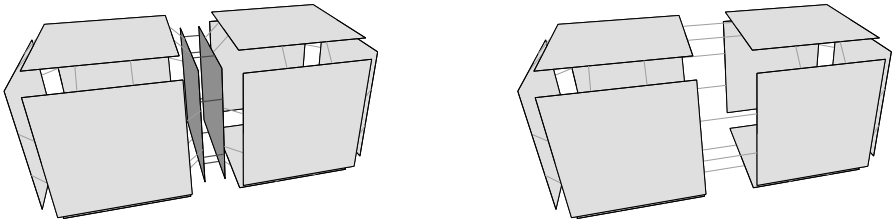


FIG. 6.6 – Fusion de volumes : $v_1 \neq v_2$.

que cette fusion ne peut pas entraîner de déconnexion étant donné que les deux volumes sont différents. Pour calculer les caractéristiques du nouveau volume, nous notons $\#F_1$ $\#A_1$ $\#S_1$ et G_1 les caractéristiques de v_1 et $\#F_2$ $\#A_2$ $\#S_2$ et G_2 les caractéristiques de v_2. Nous obtenons alors

$$\left.\begin{array}{r} \#nF = \#F_1 + \#F_2 - 2 \\ \#nA = \#A_1 + \#A_2 - 4 \\ \#nS = \#S_1 + \#S_2 - 4 \end{array}\right\} \Rightarrow nG = G_1 + G_2$$

Le nouveau volume est l'union des deux anciens. Son genre est donc simplement la somme des genres des deux volumes qui ont été fusionnés.

Cas 2 : $v_1 = v_2$ et aucune arête de C n'est de degré 1.

Ce cas est présenté figure 6.7. Si nous calculons les nouvelles caractéristiques topologiques du

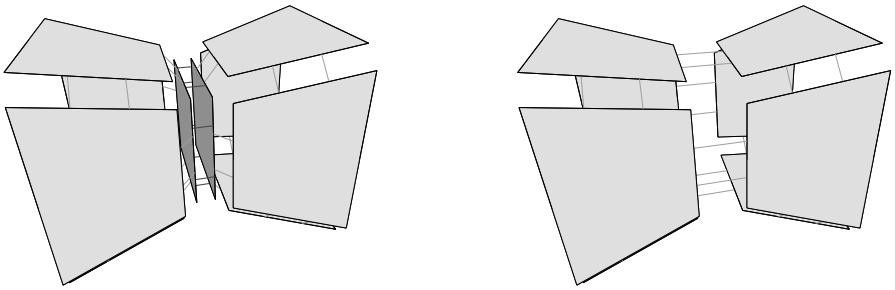
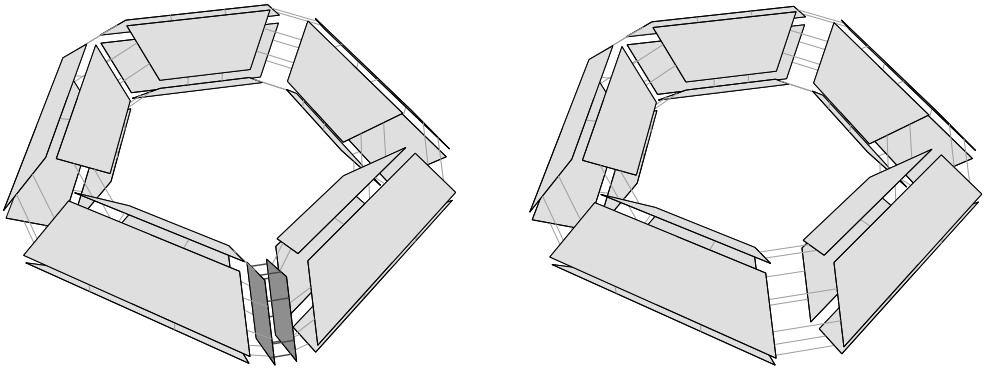


FIG. 6.7 – $v_1 = v_2$ et aucune arête de C n'est de degré 1.

volume après cette fusion, nous obtenons :

$$\left.\begin{array}{r} \#nF = \#F - 2 \\ \#nA = \#A - 4 \\ \#nS = \#S - 4 \end{array}\right\} \Rightarrow nG = G + 1$$

Nous avons donc modifié le genre du volume. Ce cas survient par exemple pour un tore ayant une face intérieure, c'est-à-dire cousue par α_3 à une face du même volume, comme pour l'exemple présenté figure 6.8. Nous pouvons remarquer sur cet exemple qu'avant la fusion, l'objet modélisé



a. $\#F = 22$, $\#A = 44$ et $\#S = 24$
$\Rightarrow G = 0$

b. $\#nF = 20$, $\#nA = 40$ et $\#nS = 20$
$\Rightarrow nG = 1$

FIG. 6.8 – Le tore avec ou sans face intérieure.

est en fait une sphère, ce qui se vérifie par le calcul du genre. En effet, nous ne considérons pas les coutures par α_3 et donc ne tenons pas compte pour le calcul du genre de la présence d'une face intérieure. Par contre, après la fusion nous modélisons bien un tore, comme nous pouvons voir par le calcul du genre, ce qui justifie donc l'incrémentation de celui-ci.

Cas 3 : $v_1 = v_2$ et une arête de C est de degré 1.

Ce cas est présenté figure 6.9. Dans ce cas, nous avons :

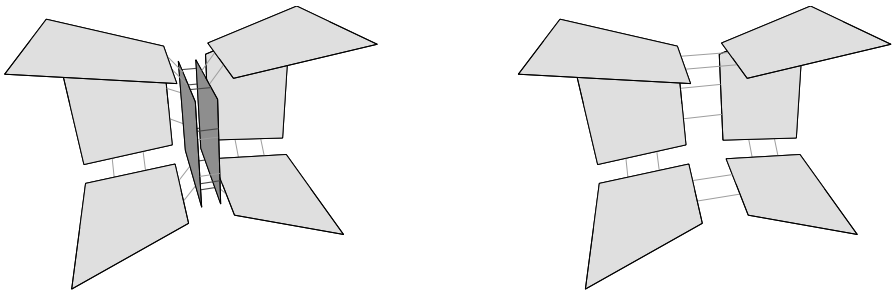


FIG. 6.9 – $v_1 = v_2$ et une arête de C est de degré 1.

$$\left.\begin{array}{c} \#nF = \#F - 2 \\ \#nA = \#A - 4 \\ \#nS = \#S - 2 \end{array}\right\} \Rightarrow nG = G$$

et conservons le genre de l'objet invariant.

Cas 4 : $v_1 = v_2$ et deux arêtes adjacentes de C sont de degré 1.

Ce cas est présenté figure 6.10. Dans ce cas, nous avons :

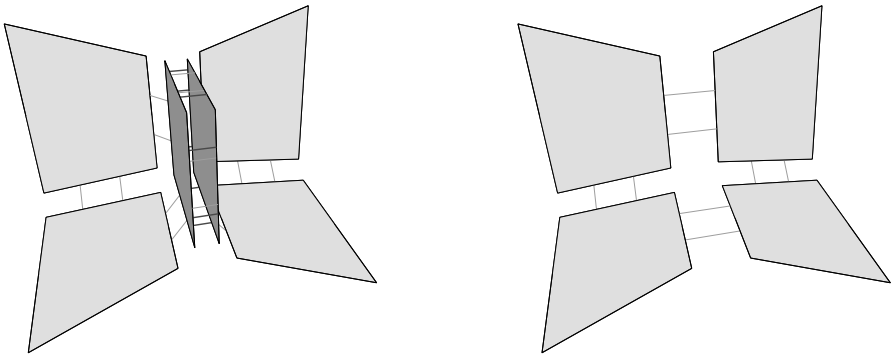


FIG. 6.10 – $v_1 = v_2$ et deux arêtes adjacentes de C sont de degré 1.

$$\left.\begin{array}{r} \#nF = \#F - 2 \\ \#nA = \#A - 4 \\ \#nS = \#S - 2 \end{array}\right\} \Rightarrow nG = G$$

et conservons également le genre de l'objet invariant.

Cas 5 : $v_1 = v_2$ et deux arêtes non-adjacentes de C sont de degré 1.

Ce cas est présenté figure 6.11 Nous avons alors

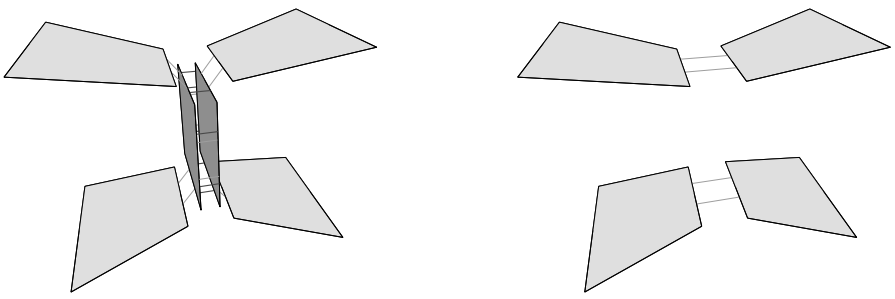
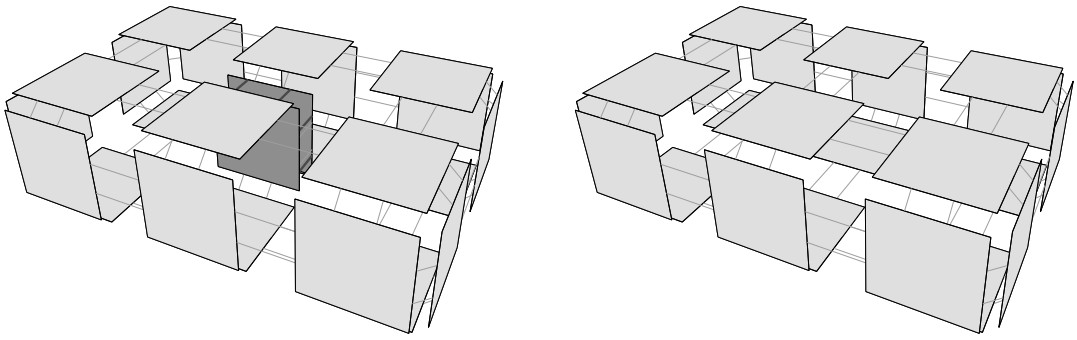


FIG. 6.11 – $v_1 = v_2$ et deux arêtes non-adjacentes de C sont de degré 1.

$$\left.\begin{array}{r} \#nF = \#F - 2 \\ \#nA = \#A - 4 \\ \#nS = \#S \end{array}\right\} \Rightarrow nG = G - 1$$

Cette modification de genre est normale, étant donné que nous modifions le type de l'objet par cette fusion. Nous pouvons voir figure 6.12 un exemple d'objet où nous nous trouvons dans ce cas. L'objet que nous modélisons avant la fusion est un tore, qui est donc de genre 1. Par contre



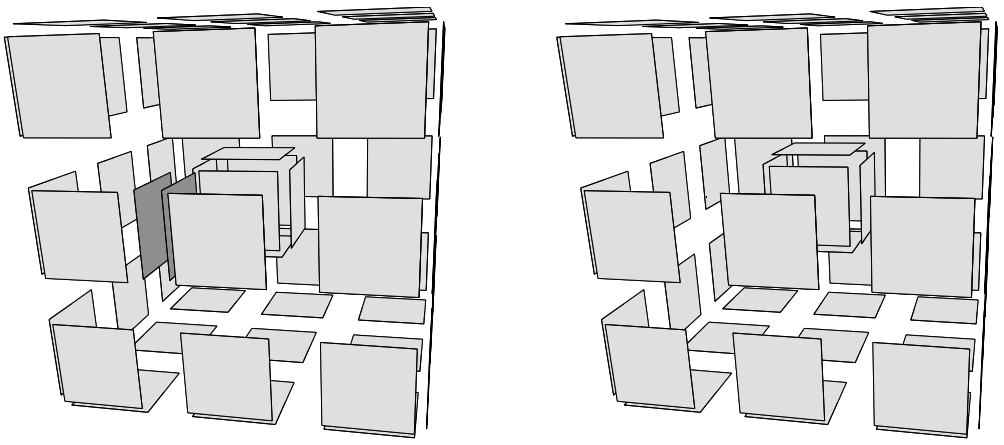
a. $\#F = 24$, $\#A = 48$ et $\#S = 24$ $\Rightarrow G = 1$

b. $\#nF = 22$, $\#nA = 44$ et $\#nS = 24$ $\Rightarrow nG = 0$

FIG. 6.12 – Un exemple où $v_1 = v_2$ et deux arêtes adjacentes de C sont de degré 1.

après la fusion, nous obtenons une sphère et il est donc normal de modifier le genre étant donné que nous avons modifié l'objet représenté.

Mais cette fusion peut entraîner une déconnexion de la carte modélisant le bord du volume, comme nous pouvons le voir sur l'exemple figure 6.13. Après la fusion, nous obtenons deux com-



a. $\#F = 62$, $\#A = 124$ et $\#S = 64$ $\Rightarrow G = 0$

b. $\#F_1 = 54$, $\#A_1 = 108$ et $\#S_1 = 56 \Rightarrow G_1 = 0$
$\#F_2 = 6$, $\#A_2 = 12$ et $\#S_2 = 8 \Rightarrow G_2 = 0$

FIG. 6.13 – Le problème de la déconnexion lors de la fusion de volumes.

posantes connexes distinctes, une pour l'extérieur du cube et une pour l'intérieur. Nous avons vu section 5.3, lors de la présentation des problèmes de déconnexion, que nous conservons un arbre d'inclusion des régions. Cet arbre permet de retrouver l'ensemble des régions incluses dans une région donnée, et permet donc de positionner les composantes connexes les unes par rapport aux autres. La formule d'Euler permettant uniquement de caractériser un objet par sa surface, elle ne peut pas caractériser ce type d'objet. Mais nous pouvons caractériser séparément la surface extérieure du cube de la surface intérieure. Nous obtenons ici deux fois un genre 0, et en déduisons grâce à l'arbre d'inclusion que cet objet est une sphère contenant une autre sphère. En utilisant la relation d'inclusion, nous avons plus d'information que seulement le genre de l'objet. Mais il n'est pas possible de caractériser les deux surfaces obtenues, étant donné que cela va dépendre de la manière dont s'effectue la coupure. Nous savons seulement que, si nous notons G_1 et G_2 le genre de ces deux surfaces, nous avons $G_1 + G_2 = G$, le genre initial de la surface.

Cas 6 : $v_1 = v_2$ et une seule arête de C n'est pas de degré 1.

Ce cas est présenté figure 6.14. Nous avons alors

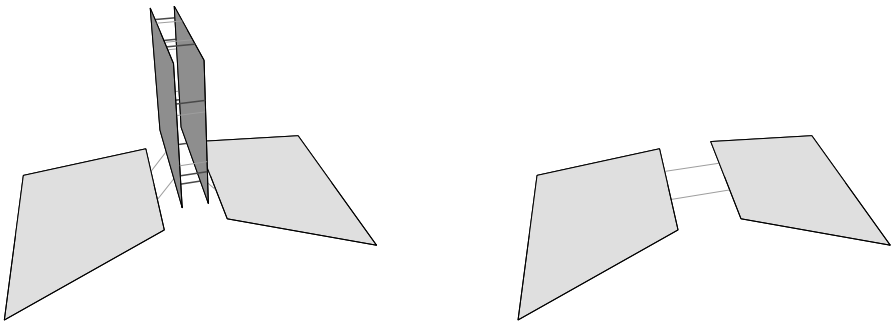


FIG. 6.14 – $v_1 = v_2$ et une seule arête de C n'est pas de degré 1.

$$\left.\begin{array}{c} \#nF = \#F - 2 \\ \#nA = \#A - 4 \\ \#nS = \#S - 2 \end{array}\right\} \Rightarrow nG = G$$

Cas 7 : $v_1 = v_2$ et les quatre arêtes de C sont de degré 1.

Dans ce cas, la fusion supprime entièrement la face. Nous *devrions* donc avoir

$$\left.\begin{array}{c} \#nF = \#F - 2 \\ \#nA = \#A - 4 \\ \#nS = \#S - 4 \end{array}\right\} \Rightarrow nG = G + 1$$

Mais ce cas survient uniquement lorsque la face à supprimer est à l'intérieur d'un volume. Cette face est donc une surface intérieure, et sa disparition entraîne la disparition totale de cette surface. Il n'existe donc plus de surface à caractériser, et le calcul de nG n'a donc pas de sens.

En résumé

Nous avons étudié toutes les configurations possibles lors de la fusion de volumes. En effet, étant donné l'ordre des fusions effectuées, nous savons que nous avons uniquement des faces carrées, ce qui limite beaucoup le nombre de configurations à étudier. Cette étude de cas aurait été plus laborieuse sans la réorganisation de la suite des fusions. Le tableau 6.1 récapitule l'évolution des caractéristiques topologiques pour la fusion de volumes, pour chacun des cas que nous venons d'étudier.

TAB. 6.1 – Récapitulation de l'évolution des caractéristiques topologiques pour la fusion de volumes.

	#nF	#nA	#nS	nG
Cas 1	$\#F_1 + \#F_2 - 2$	$\#A_1 + \#A_2 - 4$	$\#S_1 + \#S_2 - 4$	$G_1 + G_2$
Cas 2	$\#F - 2$	$\#A - 4$	$\#S - 4$	$G + 1$
Cas 3	$\#F - 2$	$\#A - 4$	$\#S - 2$	G
Cas 4	$\#F - 2$	$\#A - 4$	$\#S - 2$	G
Cas 5.*a*	Déconnexion : obtention de deux surfaces différentes			
Cas 5.*b*	$\#F - 2$	$\#A - 4$	$\#S$	$G - 1$
Cas 6	$\#F - 2$	$\#A - 4$	$\#S - 2$	G
Cas 7	Suppression totale de la surface			

Remarquons que le cas 5 est découpé en deux sous-cas : le cas 5.*a* lorsque la fusion entraîne une déconnexion de la carte et le cas 5.*b* sinon. Lorsqu'il y a déconnexion, nous obtenons deux surfaces différentes ayant chacune des caractéristiques topologiques que nous ne pouvons pas calculer à partir des caractéristiques du volume initial. Le cas symétrique est le cas 7, où la suppression de la face entraîne également la suppression totale de la surface qui n'a donc plus de caractéristique. Le seul cas pouvant entraîner une déconnexion est donc le cas 5.*a*, tout les autres conservent la carte connexe.

Nous pouvons remarquer que les cas 1, 2 et 5.*b* ne conservent pas invariant le genre après la fusion. Pour le cas 1, c'est évident car nous fusionnons deux volumes différents. Pour les deux autres cas, cela est nécessaire car ils modifient la topologie de l'objet représenté. En effet, la carte de niveau 0 ne représente pas l'image segmentée mais uniquement un ensemble de voxels, donc seulement des volumes de genre 0. Avec les fusions de volumes, nous calculons la carte de niveau 1 qui représente les régions de l'image, et il est donc normal que les caractéristiques topologiques changent afin de représenter la topologie de ces objets. De ce fait, cette étude ne montre pas que les caractéristiques restent invariantes, mais permet d'étudier toutes les configurations possibles lors de la fusion de volumes, et de s'assurer que nous n'avons pas oublié des cas de déconnexion. De plus, nous avons vu comment caractériser un volume contenant d'autres volumes, sachant que ce type de volume n'est pas caractérisable à l'aide de la formule d'Euler.

6.3.2 Fusion de faces

Nous effectuons maintenant la même démarche, mais pour la fusion de faces. Nous fusionnons donc deux faces f_1 et f_2 le long de l'arête C qui est incidente aux deux faces et localement de

degré deux. Toujours grâce à l'ordre sur les fusions, nous n'avons pas encore effectué de fusion d'arêtes, et chaque arête représente donc un lignel. Cela a pour conséquence qu'il est impossible d'avoir des boucles, qui sont autrement des cas particuliers. Nous montrons que la fusion de faces conserve invariant le genre des objets quelle que soit la configuration des faces.

Il existe deux cas différents, suivant si $f_1 = f_2$ ou $f_1 \neq f_2$, et dans le premier cas plusieurs sous-cas suivant la configuration de l'arête C. Nous savons que la carte est fermée, et que nous n'avons pas de face à l'intérieur d'un même volume, car nous avons effectué toutes les fusions de volumes possibles. Pour chaque demi-face d'un volume (donc une orbite $< \alpha_0, \alpha_1 >$), nous savons qu'il existe un autre volume ayant une face identique à la première, avec leurs brins cousus par α_3 deux à deux. Comme nous voulons étudier l'évolution des caractéristiques topologiques seulement pour un volume, nous oublions donc la deuxième face cousue par α_3. Nous nous ramenons donc en dimension 2.

Cas 1 : $f_1 \neq f_2$.

Ce cas est présenté figure 6.15. Nous avons alors

FIG. 6.15 – Fusion de faces : $f_1 \neq f_2$.

$$\left.\begin{array}{r} \#nF = \#F - 1 \\ \#nA = \#A - 1 \\ \#nS = \#S \end{array}\right\} \Rightarrow nG = G$$

Cas 2 : $f_1 = f_2$ et aucun sommet de C n'est de degré 1.

Ce cas est présenté figure 6.16. Nous pouvons voir sur cet exemple, que cette fusion entraîne

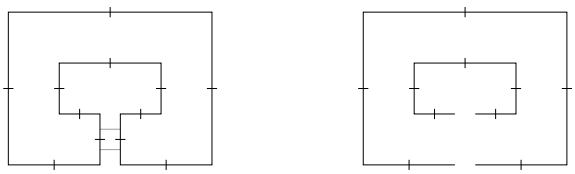


FIG. 6.16 – Fusion de faces : $f_1 = f_2$ et aucun sommet de C n'est de degré 1.

une déconnexion de la face. Ce type de fusion n'est pas possible dans le cadre de la carte topologique. Nous avons en effet rajouté dans les deux définitions utilisant la fusion de faces la contrainte explicite de ne pas fusionner deux faces lorsque cela entraîne une déconnexion.

Cas 3 : $f_1 = f_2$ et un sommet de C est de degré 1.

Ce cas est présenté figure 6.17.

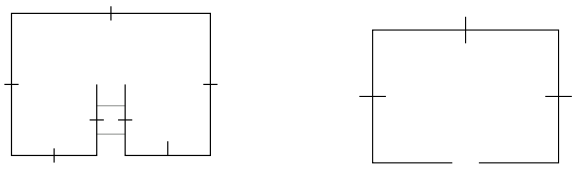


FIG. 6.17 – Fusion de faces : $f_1 = f_2$ et un sommet de C est de degré 1.

Nous avons ici

$$\left.\begin{array}{r} \#nF = \#F \\ \#nA = \#A - 1 \\ \#nS = \#S - 1 \end{array}\right\} \Rightarrow nG = G$$

Cas 4 : $f_1 = f_2$ et les deux sommets de C sont de degré 1.

Ce cas ne peut pas se produire dans le cadre des cartes topologiques. En effet, cette arête représente une sphère topologique (cas étudié section 5.3), et nous n'effectuons alors pas cette fusion. Nous avons en effet ajouté dans la définition de la carte de niveau 4 la contrainte de ne pas effectuer la fusion de deux faces si cela entraîne la disparition complète de la face. Remarquons que cette arête ne peut pas représenter autre chose qu'une sphère, étant donné que nous conservons chaque face connexe. Il n'est donc pas possible qu'elle soit au « milieu » d'une autre face.

En résumé

Le nombre de cas possibles est beaucoup plus faible que pour la fusion de volumes. En effet, nous fusionnons deux faces le long d'une arête. Les seules possibilités sont lorsque les deux faces sont différentes ou non, et lorsque le degré des sommets est de degré un ou non. Nous pouvons remarquer, comme pour l'étude de la fusion de volumes, que ce faible nombre de cas résulte de la réorganisation des fusions. Le tableau 6.2 récapitule l'évolution des caractéristiques topologiques pour la fusion de faces.

TAB. 6.2 – Récapitulation de l'évolution des caractéristiques topologiques pour la fusion de faces.

	#nF	#nA	#nS	nG
Cas 1	$\#F - 1$	$\#A - 1$	$\#S$	G
Cas 2	Impossible			
Cas 3	$\#F$	$\#A - 1$	$\#S - 1$	G
Cas 4	Impossible			

Nous voyons donc que deux cas conservent les caractéristiques topologiques invariantes. Les deux autres cas poseraient éventuellement problèmes, mais ont été interdits lors des définitions de nos différents niveaux de simplification. Cela prouve que nous ne perdons pas d'information topologique lors de fusion de faces, mais également que nous avons bien pris en compte tous les cas possibles, et traité les deux posant problème en modifiant nos définitions.

6.3.3 Fusion d'arêtes

La fusion d'arêtes ne peut pas entraîner de déconnexion. De plus, il n'existe pas plusieurs cas différents. Nous avons tout le temps

$$\left.\begin{array}{r} \#nF = \#F \\ \#nA = \#A - 1 \\ \#nS = \#S - 1 \end{array}\right\} \Rightarrow nG = G$$

La seule configuration qui pourrait poser problème, est la fusion d'une boucle avec une autre arête. Mais ce cas ne peut pas arriver pour la carte de niveau 3, étant donné que nous fusionnons seulement des arêtes alignées. Pour le niveau 5, nous avons rajouté la condition explicite interdisant ce type de fusion.

Nous avons vu que les fusions de faces et d'arêtes conservent invariant le genre des objets, et ce pour n'importe quelle configuration. Cela prouve donc que les cartes de niveaux 1 à 5 représentent bien les mêmes informations topologiques, étant donné que ce sont les deux seules fusions utilisées afin de calculer ces différents niveaux à partir du niveau 1.

6.4 Conclusion

Dans ce chapitre, nous avons donné une définition algébrique de l'opération de fusion. Pour cela, nous avons introduit la notion de degré deux local, qui permet de regrouper les cas de cellules de dimension 1 et dimension 2. Nous avons pu ainsi prouver que l'opération de fusion est valide. De plus, la définition est ici un peu plus générale que nos besoins pour la définition de la carte topologique. Pour cette dernière, nous savons qu'elle est fermée et orientable, fait que nous n'avons pas utilisé pour la définition de la fusion. De ce fait, cette définition est la plus générique possible, et fonctionne dans n'importe quelle configuration, ce qui laisse plus de possibilités pour son utilisation.

Nous nous sommes ensuite intéressé à l'ordre des différentes fusions effectuées afin de calculer la carte topologique à partir de la carte de niveau 0. Nous avons ainsi montré que la suite des fusions effectuées peut être réorganisée afin d'être ordonnée selon la dimension décroissante des fusions. Cette partie est intéressante pour deux raisons. Tout d'abord car cela nous a permis de diminuer grandement le nombre de cas à considérer pour l'étude de l'évolution des caractéristiques topologiques. Mais cela valide également l'algorithme optimal d'extraction, qui effectue les fusions de manière désordonnée.

Enfin, l'étude sur l'évolution des caractéristiques topologique est primordiale car elle prouve que les cartes de niveaux 1 à 5 représentent toutes les mêmes informations topologiques. De plus,

elle permet de s'assurer que nous n'avons pas oublié de cas particulier. Elle confirme également que les cas pouvant poser problèmes ont tous été correctement traités.

Pour être complet, cette étude aurait mérité d'intégrer la gestion des arêtes fictives. En effet, nous nous sommes ici uniquement intéressé à l'opération de fusion elle-même. Mais nous avons vu lors de la définition de la carte topologique que la gestion des arêtes fictives est primordiale afin d'obtenir la représentation minimale. Il serait donc intéressant de montrer que le principe de décalage d'arêtes fictives ne modifie pas les différentes caractéristiques topologiques. Ce point est l'une de nos perspectives de recherche. Il faut pour cela montrer que ce décalage ne change aucune caractéristique topologique, ce qui est évident intuitivement car nous ne modifions pas le nombre de cellules, mais il faut également montrer que ce décalage conserve la connexité de chaque face, point seulement abordé intuitivement dans ce travail.

CHAPITRE 7

TRAVAUX CONNEXES ET DÉVELOPPEMENTS

Durant ce travail de thèse, nous avons eu l'occasion d'aborder plusieurs projets liés à notre travail de recherche. Nous présentons ici quelques-uns de ces travaux. Le premier projet, présenté section 7.1, concerne le développement d'un noyau géométrique à base topologique. Ce noyau s'intègre dans le cadre du projet RNTL[1] *Nogémo*. Le but de ce projet est de concevoir et de développer un noyau géométrique modulaire pour modeleurs et simulateurs. Ce noyau est au cœur de différents travaux. Nous avons travaillé sur deux d'entre eux, un modeleur géométrique à base topologique, *Moka*, présenté section 7.2, et un projet d'analyse d'images pour une aide à la détection de tumeurs cérébrales présenté section 7.3.

Section 7.4 nous présentons une étude sur l'intégration de la carte topologique dans un processus de segmentation markovienne. Ce travail montre que la carte topologique peut être intégrée dans des algorithmes de segmentation. Nous avons également travaillé sur quelques opérations de modification de la carte topologique 2d et 3d, principalement les opérations de fusion et de coupure. En effet elles constituent les opérations de base pour plusieurs algorithmes de segmentation. De plus, nous avons étudié une opération intéressante, le *coraffinement*, qui permet d'envisager l'implantation d'algorithmes de segmentation en parallèle sur différents « morceaux » d'images, l'image totale étant ensuite recomposée au moyen de cette opération. Ces opérations sont présentées brièvement section 7.5. Enfin, nous avons étudié différentes possibilités pour implanter efficacement la carte topologique, tout en conservant un programme modulaire et extensible. Cette réflexion est présentée section 7.6.

7.1 Un noyau générique de 3-G-cartes

Ce noyau est la base du projet RNTL *Nogémo*. Il a été initialement développé par Allan Fousse et Daniel Menevaux, maîtres de conférences au laboratoire IRCOM-SIC. Nous l'avons ensuite entièrement repris, afin de le stabiliser, de l'homogénéiser, de l'optimiser mais également de le valider. Ce noyau étant au centre d'un projet de plate-forme logicielle, il se doit d'être le plus

[1] Réseau National des Technologies Logicielles.

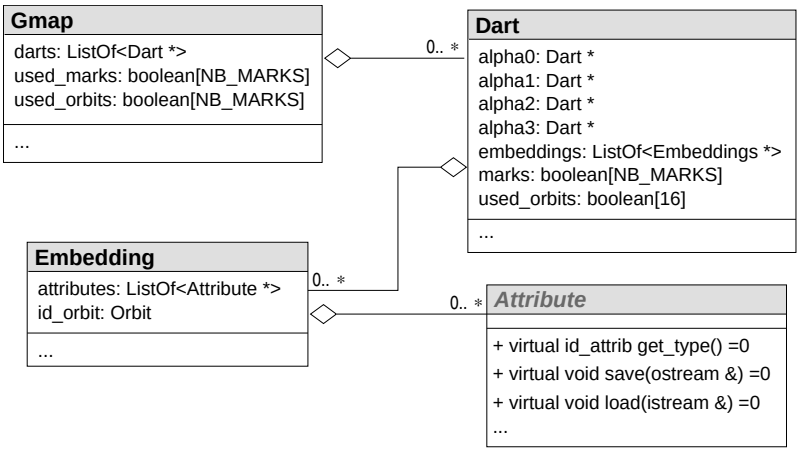


FIG. 7.1 – Le schéma UML du noyau de 3-G-carte, représentation partielle.

générique possible. En effet, les différentes applications pouvant l'utiliser ont toutes des besoins spécifiques différents. Dans sa version actuelle, ce noyau est basé sur les 3-G-cartes. Il permet donc la modélisation de quasi-variétés orientables ou non de dimension inférieure ou égale à trois. Il a été développé en C++, afin d'obtenir un code facilement modifiable, et extensible.

Nous expliquons brièvement la structure globale de ce noyau. Il comprend trois classes principales : la classe *Gmap* est la classe de base permettant de déclarer une 3-G-carte, la classe *Dart* représente un brin, et enfin la classe *Attribute* représente un attribut associé à une orbite particulière. Cet attribut peut être géométrique, comme les coordonnées d'un point 3d que nous associons à une orbite sommet, ou une 2-G-carte que nous associons à un plongement face, mais également d'autres attributs de couleur, texture... Ce noyau permet d'associer à n'importe quelle orbite de la 3-G-carte n'importe quel type d'attribut. De plus, quel que soit le plongement et le type d'attribut, nous avons implanté les coutures et décousures afin qu'elles mettent à jour ces attributs lorsque cela est nécessaire.

Un attribut spécifique est associé à une orbite particulière. Plusieurs attributs différents peuvent être associés à une même orbite. Par exemple, nous pouvons associer à une face une surface 2d, une couleur et une texture. Ces attributs sont regroupés dans la classe *Embedding*. Nous pouvons voir figure 7.1 le schéma représentant ces classes, et la manière dont elles sont reliées. Sur ce schéma UML, nous avons représenté seulement les principaux champs afin de ne pas entrer dans les détails techniques.

La classe *Gmap* est principalement composée d'un ensemble de brins, mais possède également d'autres attributs. Le champ *used_orbits* est un tableau de 16 booléens, qui permet pour chaque orbite de savoir si elle est plongée ou non. Cela permet de ne pas effectuer de parcours afin de chercher un plongement, lorsque nous savons que cette orbite n'est pas plongée. Enfin, le champ *used_marks* donne les marques booléennes qui sont en cours d'utilisation. La classe *Gmap* possède des méthodes permettant de demander et de réserver une marque libre, et de libérer une marque. Ces deux méthodes vont utiliser et mettre à jour ce tableau. Cette classe possède un grand nombre de méthodes, permettant de créer ou supprimer des brins, de coudre ou découdre ces brins, avec ou sans mise à jour des plongements, des méthodes pour affecter, supprimer ou récupérer un *Attribut*

pour une orbite particulière. D'autres méthodes permettent de tester si un brin est marqué, de le marquer ou le démarquer. Ce sont les principales méthodes que l'utilisateur peut appeler. Il existe bien entendu plusieurs autres méthodes, certaines étant privées et utilisées en interne pour par exemple mettre à jour les plongements, ou tester si deux brins appartiennent à une même orbite...

La classe *Dart* possède quatre pointeurs afin de représenter les quatre involutions α et une liste d'*Embedding* portées par ce brin. Chaque *Embedding* correspond à une orbite particulière. Le tableau de booléens *used_orbit* permet de savoir en *O(1)* si ce brin porte un *Embedding* pour une orbite. Cela évite de parcourir la liste *Embedding* inutilement. Enfin le tableau *marks* contient les marques booléennes de ce brin.

La classe *Embedding* contient un champ *id_orbit* correspondant à un identificateur de l'orbite correspondant à cet *Embedding*. Elle contient ensuite la liste des *Attribute* contenu dans cet *Embedding*. La classe *Attribute* est une classe virtuelle pure. L'utilisateur désirant un attribut particulier va créer sa classe dérivant de *Attribute*, et fixer son « comportement ». Il faut, entre autres, donner un identifiant différent à chaque *Attribut*, définir les méthodes *Save* et *Load*... Cela permet par exemple d'écrire une méthode de sauvegarde générique dans la classe *Gmap*, qui va utiliser la méthode *Save* des classes *Attribut* redéfinies par l'utilisateur. Nous avons également implanté toutes les classes *Coverage* permettant de parcourir les brins de chaque orbite possible. Ces classes sont implantées « à la STL », comme des itérateurs. Cela permet par exemple d'utiliser l'opérateur ++ pour passer au brin suivant du parcours.

Nous ne détaillons pas plus les fonctionnalités de ce noyau. Son principal atout est d'être très générique, nous n'avons fixé aucune contrainte sur son utilisation, à l'exception de la dimension. De plus, il gère de manière transparente l'utilisation de n'importe quel plongement, ce qui permet à un utilisateur non spécialiste de l'utiliser sans se préoccuper de cet aspect. Ses deux défauts principaux sont sa lenteur et l'espace mémoire occupé. En effet, de par sa généricité, il effectue beaucoup de tests afin de mettre à jour tous les éventuels plongements, et la structure en liste de listes pour les attributs est coûteuse en espace mémoire. Mais ce noyau peut être considéré comme un prototype. Lorsqu'une application spécifique nécessite un plongement particulier, il est possible de spécialiser le noyau pour tenir compte de ce plongement, et ainsi supprimer des tests, et les listes d'attributs. Nous obtenons alors un noyau spécialisé, pour lequel nous ne pouvons plus plonger n'importe quelle orbite, mais moins coûteux en temps d'exécution et en espace mémoire. Ce changement de noyau pourra être effectué sans aucune modification des sources de l'application, simplement en conservant la même interface entre la version générale et la version spécialisée.

7.2 Moka : un modeleur géométrique à base topologique

Moka[2] est un modeleur géométrique à base topologique. Il a été développé par Frédéric Vidil lors de plusieurs stages que nous avons encadrés et co-encadrés. Durant ces mois de travail, nous avons participé au développement de certaines fonctions spécifiques, comme par exemple les opérations de fusion, bouchage, triangulation ou l'export xfig[3].

[2] Pour modeleur de cartes.

[3] La majorité des figures de cette thèse sont d'ailleurs réalisées avec ce modeleur et cette fonction.

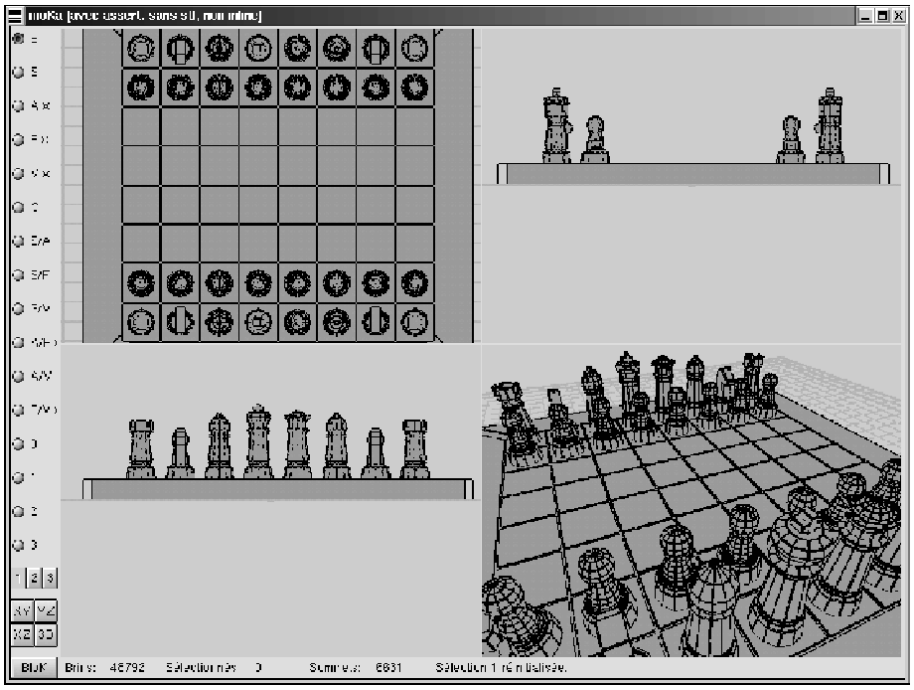
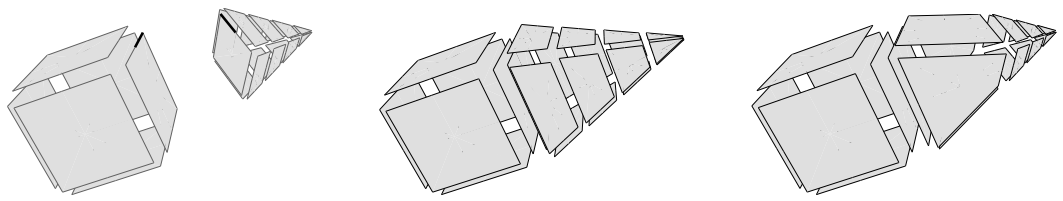


FIG. 7.2 – Une photo d'écran de Moka.

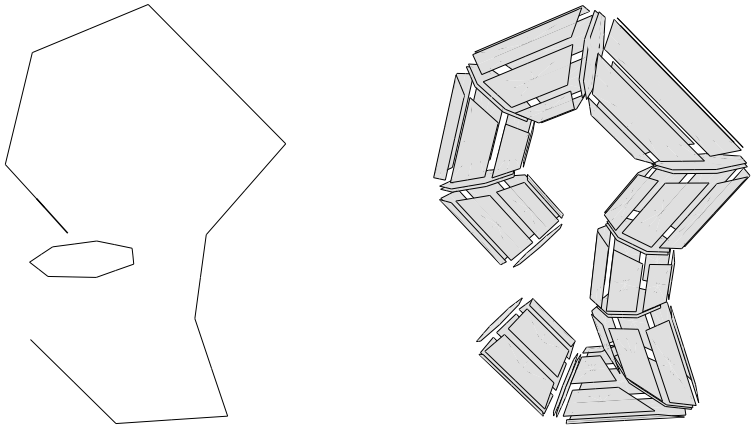
Ce modeleur est basé sur le noyau présenté section précédente. De ce fait, nous avons pu profiter de l'ensemble des fonctions déjà existantes, et nous consacrer ici uniquement à la couche application et aux fonctions de haut niveau. La figure 7.2 présente une photo d'écran de Moka. Nous pouvons voir que ce modeleur possède 4 fenêtres de visualisation différentes : 3 vues 2d sur les différents plans possibles, et une vue 3d. Il est possible de basculer l'affichage sur une seule de ces vues. L'utilisateur a à sa disposition un certain nombre d'objets qu'il peut créer, combiner et déformer à sa guise. Ces objets de base sont de dimension 2, par exemple des lignes brisées, des polygones réguliers ou non, ou des objets 3d comme des cubes, sphères, tores, pyramides, cônes. Tout les différents paramètres de ces objets sont paramétrables, comme par exemple le nombre de subdivisions de la sphère sur les parallèles et les méridiens, la position du centre, la direction…

Il existe de nombreuses opérations, les deux principales étant bien entendu la couture et la décousure. Il est possible d'effectuer ces deux opérations pour toutes les dimensions, mais également de manière « intuitive » où la dimension est calculée automatiquement, suivant les dimensions déjà cousues. La couture peut être réalisée de manière topologique ou géométrique, comme nous pouvons voir figure 7.3. La couture géométrique déforme le deuxième objet, par translation, rotation et homothétie, afin que la face cousue soit de la même géométrie que la face du premier objet. Nous voyons figure 7.3.*b* que cette couture permet de « plaquer » les deux objets. La couture topologique (cf. figure 7.3.*c*) n'effectue aucune modification géométrique. La face cousue du deuxième objet récupère la géométrie de la face du premier objet de manière automatique, car les orbites sommets sont fusionnés, mais le reste de l'objet ne subit aucune déformation géométrique.



a. Deux objets. b. Après la couture géométrique. c. Après la couture topologique.

FIG. 7.3 – La couture topologique et géométrique.



a. Une face et un chemin. b. Après l'extrusion.

FIG. 7.4 – L'opération d'extrusion d'une face le long d'un chemin.

Quelques-unes des autres opérations existantes sont le bouchage, la triangulation, la quadrangulation, la fusion, l'insertion de cellules, différents maillages, différentes extrusions... La plupart de ces opérations fonctionnent pour n'importe quelle dimension. Par exemple la triangulation est développée en dimension 1, ce qui revient à couper un segment en deux, en dimension 2 pour trianguler une face, et en dimension 3 pour trianguler un volume en plusieurs tétraèdres. Nous pouvons voir figure 7.4 un exemple d'extrusion d'une face le long d'un chemin. Cette opération permet de construire un volume en faisant « courir » la face le long du chemin.

Ce modeleur continue a être développé, afin d'intégrer d'autres opérations plus complexes, dont certaines sont primordiales dans le monde des CAO : principalement le chanfreinage et les opérations booléennes, mais également les plongements splines, nurbs... De plus, c'est ce modeleur qui va être la base de travail pour la réalisation du projet Nogémo évoqué section 7.1. Mais la version actuelle est déjà opérationnelle, validée d'ailleurs par sa grande utilisation dans cette thèse, et possède suffisamment d'opérations pour modeler des scènes élaborées.

7.3 Projet de détermination du volume tumoral cérébral

Ce projet a été réalisé par Cyrius Cayrous [Cay01] lors de son stage de DESS que nous avons co-encadré. L'objectif de ce stage était la réalisation d'une interface intégrant des outils de segmentation et de visualisation volumique sur des coupes IRM de cerveau. La réalisation de cette application s'inscrit dans le cadre d'un projet de détermination du volume tumoral et de ses variations lors du suivi évolutif. L'outil informatique à développer a pour but d'aider le médecin dans son choix de stratégie thérapeutique.

Nous avons tout d'abord intégré des algorithmes de segmentation [CAFMF01] au cœur d'une interface graphique permettant de charger un ensemble de coupes scanner, et de les segmenter en autorisant bien entendu le réglage de tous les paramètres de cette segmentation. Nous avons également développé certaines opérations en 2d, comme la possibilité de visualiser l'ensemble des coupes sous forme de mosaïque, de modifier les paramètres de couleur et contraste, afin de faire ressortir certaines régions de l'image...

Puis, nous avons intégré l'algorithme optimal d'extraction de la carte de niveau n à partir d'une image segmentée, présenté section 5.6. Étant donné le faible temps imparti, nous avons implanté uniquement le niveau 3 de simplification. En effet, le nombre de traitements différents à écrire pour traiter les précodes de ce niveau est beaucoup moins important que pour la carte topologique. Cette application montre l'avantage de nos différents niveaux de simplification qui offrent le choix aux utilisateurs de travailler avec un niveau particulier. Ce développement est également basé sur le noyau de 3-G-cartes présenté section 7.2. De ce fait, il nous a fallu étudier les traitements des précodes sur les G-cartes, étant donné que le travail présenté section 5.6 utilise les cartes combinatoires. Mais cette conversion s'effectue sans difficulté particulière. Après avoir construit cette carte, nous avons ensuite implanté des opérations en dimension 3, principalement différentes possibilités de visualisation.

La figure 7.5 montre une capture d'écran du logiciel développé. Cette figure présente le mode 3d, sur lequel nous voyons la carte des frontières reconstruite à partir des coupes IRM préalablement segmentée. Nous avons sélectionné sur une de ces coupes une région, qui est présentée en faces pleines en 3d. Nous pouvons indifféremment visualiser seulement cette région, en faces pleines ou non, ou toutes les régions et cette région sélectionnée en faces pleines. Ce dernier mode de visualisation permet de visualiser la position de cette région par rapport à l'image totale. Cet affichage représente pour le moment les faces en intervoxel, ce qui explique le rendu « cubique » des objets. Nous envisageons l'implantation d'algorithmes de lissage afin d'obtenir un meilleur rendu pour l'affichage.

Ce travail peut être considéré comme un prototype, permettant de montrer quelques fonctionnalités en 3d utilisant la carte combinatoire reconstruite. Mais nous envisageons de nombreuses autres opérations afin de faciliter le travail du praticien, par exemple la modification interactive de la segmentation lorsqu'il n'est pas satisfait de cette dernière. Pour cela, il faut pouvoir fusionner certaines régions, ou au contraire relancer la segmentation seulement sur certaines régions en modifiant les paramètres. De plus, nous devons définir les opérations permettant le calcul volumétrique d'une région sélectionnée de manière précise. En effet, la version actuelle se contente de compter les voxels de cette région, et d'approximer le volume en connaissant les caractéristiques de l'examen. Même si cela permet une première estimation, ce calcul n'est pas très précis et mérite d'être amélioré.

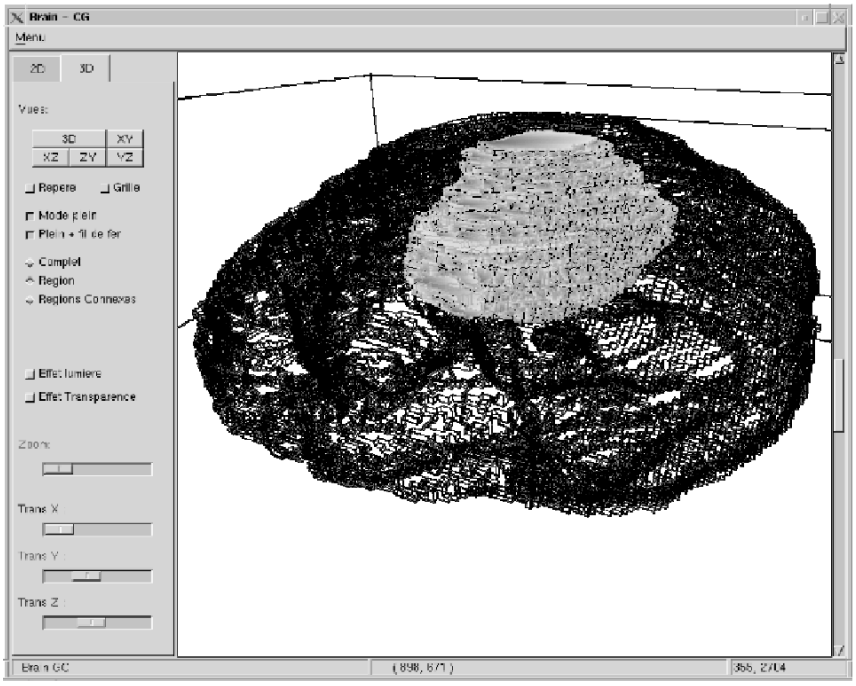


FIG. 7.5 – Capture d'écran du logiciel de détermination du volume tumoral cérébral.

7.4 Intégration de la carte topologique 2d dans un processus de segmentation markovienne

Ce travail est le résultat du stage de recherche de DEA effectué par Pascal Bourdon [Bou01] que nous avons co-encadré. La segmentation par champs de Markov est une des approches souvent utilisées dans le monde du traitement des signaux et des images. C'est une méthode dite *stochastique*, qui consiste à considérer l'image comme un processus aléatoire discret paramétrable dont nous tentons d'estimer les paramètres associés à chaque région. Cette approche permet d'intégrer facilement les critères d'appartenance à une région sous forme de *potentiels*.

L'idée de ce travail de recherche consiste à intégrer la carte topologique dans le processus de segmentation markovienne, afin de pouvoir ainsi définir des potentiels calculés sur cette carte. Pour cela, nous avons tout d'abord développé l'algorithme classique de segmentation markovienne, sans utiliser la carte topologique. Afin d'intégrer ensuite ces dernières, nous avons implanté l'algorithme optimal d'extraction de la carte topologique à partir d'une image. La partie importante de recherche a été la définition de nouveaux potentiels pouvant se calculer sur cette carte. De plus, il nous a fallu étudier la manière dont ces nouveaux potentiels peuvent s'intégrer aux potentiels classiques.

Nous avons défini un potentiel *taille* permettant de défavoriser la création de région de petite taille. Ce potentiel peut se calculer très simplement et efficacement sur la carte topologique. Les résultats obtenus sont encourageants et montrent que l'intégration de ce potentiel améliore la segmentation, en limitant effectivement la création de petites régions, qui sont souvent le résultat

d'une sur-segmentation. Les différents problèmes techniques rencontrés lors de ce travail nous ont empêchés de tester d'autres potentiels. Mais ces premiers résultats montrent la faisabilité de cette nouvelle approche, tant pour le domaine du traitement des signaux et des images que pour celui de l'infographie ou de l'imagerie combinatoire. Nous envisageons de poursuivre ce travail lors d'un second stage de DEA. Nous pouvons maintenant étudier directement les potentiels calculés sur la carte topologique, étant donné que les problèmes techniques ont déjà été résolus. Les possibilités sont nombreuses : nous envisageons des potentiels de rugosité, de forme, de relation de voisinages... Le calcul de ces critères sera plus facile et efficace grâce à la carte topologique.

7.5 Quelques opérations sur la carte topologique 2d et 3d

Ce travail est le résultat du stage de recherche de DEA effectué par Patrick Resch [Res01a, Res01b] que nous avons encadré. Le but de ce stage était de définir les opérations de fusion et de coupure sur la carte topologique 3d. Ces opérations sont en effet la base de nombreux algorithmes de segmentations en dimension 2, et donc de futurs algorithmes de segmentations 3d. De plus, ces deux opérations sont également importantes dans une optique interactive de contrôle d'une segmentation. En effet, la plupart des outils de segmentation doivent autoriser le contrôle, et éventuellement la modification interactive d'une segmentation automatique. Un expert doit en effet obligatoirement valider la segmentation dans les domaines sensibles, comme par exemple l'imagerie médicale, et pouvoir fusionner deux régions s'il juge qu'elles ont été sur-segmentées, ou au contraire couper une région lorsqu'il le désire.

Au cours de ce travail, nous nous sommes également intéressé à une autre opération, dans une optique de parallélisation de l'algorithme de segmentation : le *coraffinement*. L'idée consiste à couper une grosse image en plusieurs petits blocs, segmenter ensuite chaque bloc séparément en parallèle, puis reconstruire la segmentation de l'image globale en « recollant » chaque morceau d'image. L'opération effectuant ce recollement est le coraffinement. Ce principe permet d'envisager la segmentation de grosses images. En effet, les algorithmes de segmentation sont souvent gourmands en temps de calcul, et la taille des données à traiter est très importante en 3d.

Afin de définir ces trois opérations en dimension 3, nous avons tout d'abord commencé à les étudier en dimension 2, de manière similaire à la démarche utilisée pour ce travail de thèse. En effet, les problèmes de visualisation de la carte topologique 3d freinent considérablement sa compréhension. Le fait de repartir en 2d permet de bien comprendre les problèmes des différentes opérations, et de mieux envisager ces mêmes problèmes en passant à la dimension supérieure. De plus, nous avons réutilisé certaines techniques 2d pour certains traitements spécifiques 3d.

Nous avons défini ces opérations de manière locale, chaque fois que cela était possible. Cette technique permet de limiter le nombre de cas à traiter, et simplifie donc les algorithmes. De plus, nous avons également préféré une légère perte en complexité au profit d'une plus grande simplicité des algorithmes, afin d'en faciliter la compréhension. En effet, de nombreux points sont assez techniques et l'écriture optimale de l'opération devient vite incompréhensible. Nous ne détaillons pas ici ces algorithmes car cela demanderait un chapitre complet, mais le lecteur intéressé pourra se référer à [Res01a, Res01b, DR02]. Nous allons maintenant étudier comment intégrer ces algorithmes dans un processus de segmentation 3d.

7.6 Implantation optimisée de la carte topologique 2d et 3d

Afin d'implanter la carte topologique ainsi que les différents algorithmes d'extraction, nous avons développé un programme permettant de la représenter et de la manipuler. Étant donné les plongements différents qu'il est possible d'utiliser, ce programme devait être modulaire afin de pouvoir changer facilement de plongement sans trop de problème. Nous avons pour cela choisi le langage C++ afin de réaliser un modèle objet facilement extensible. Mais nous avons un problème d'optimisation, tant au niveau de l'espace mémoire que du temps d'exécution. En effet, de par la taille importante des données à traiter, principalement en dimension 3, il faut minimiser l'espace mémoire utilisé par l'application afin de pouvoir travailler sur des images de taille raisonnable. Mais nous avons également un problème de complexité en temps d'exécution. Si l'algorithme d'extraction demande trop de temps, il ne sera pas utilisable dans des domaines applicatifs traitant les images dans des délais assez courts. Ces trois problèmes sont souvent en opposition. Un code C++ modulaire est souvent plus lent en temps d'exécution qu'un code C totalement optimisé, et l'amélioration du temps d'exécution se fait souvent au détriment de l'espace mémoire. Nous avons donc essayé de trouver un compromis entre ces trois contraintes, mais en privilégiant légèrement l'espace mémoire car nous avons jugé que c'était la contrainte la plus importante.

Nous ne détaillons pas précisément cette étude, mais en donnons seulement les idées principales. Le lecteur intéressé pourra se reporter au rapport de recherche [DF99]. La première constatation que nous avons faite est que l'utilisation des méthodes virtuelles est coûteuse en temps d'exécution et en espace mémoire. En effet, chaque classe possédant une méthode virtuelle réserve un pointeur vers la table des méthodes virtuelles afin de résoudre les appels à ces méthodes de manière dynamique. Une classe ayant une méthode virtuelle a 4 octets de plus que la même classe sans méthode virtuelle. De plus, les appels aux méthodes virtuelles sont résolus dynamiquement, ce qui entraîne une indirection supplémentaire, mais surtout interdit le mécanisme d'*inline*. Une méthode *inline* est recopiée à l'endroit où elle est appelée, ce qui évite un appel de fonction ainsi que l'empilement des paramètres. De ce fait, un appel à une méthode *inline* est moins coûteux en temps qu'un appel à une méthode non *inline*. Ces deux raisons font qu'un appel à une méthode virtuelle prend plus de temps qu'un appel à une méthode non virtuelle. Mais les méthodes virtuelles sont les seules possibilités, en C++, d'obtenir un code modulaire. Il faut donc jongler entre les deux aspects, et définir certaines méthodes virtuelles lorsqu'elles sont susceptibles d'être redéfinies, et les autres non virtuelles afin qu'elles soient moins coûteuses en temps d'exécution. Les différents tests que nous avons effectués entre une implantation où toutes les méthodes sont virtuelles, et une deuxième où seules un petit nombre le sont montrent que le temps d'exécution de la première solution est en moyenne deux fois plus important que pour la deuxième implantation.

Nous avons ensuite étudié la manière de représenter les brins par rapport aux différents plongements que nous utilisons. En dimension 2, nous avons implanté le plongement « arête ouverte et sommet » présenté section 4.3, afin de ne pas représenter un même sommet géométrique plusieurs fois. L'implantation directe de ce plongement consiste à ajouter un pointeur par brin pour le plongement arête, et un pointeur par brin pour le plongement sommet. Mais un seul brin par orbite sommet désigne effectivement le plongement correspondant, et un seul brin par orbite arête désigne le plongement arête. Les autres brins n'utilisent pas ces pointeurs, et l'espace mémoire correspondant est inutile. Afin de résoudre ce problème de perte d'espace mémoire, nous avons utilisé pleinement les mécanismes objets, et implanté le modèle objet présenté figure 7.6.

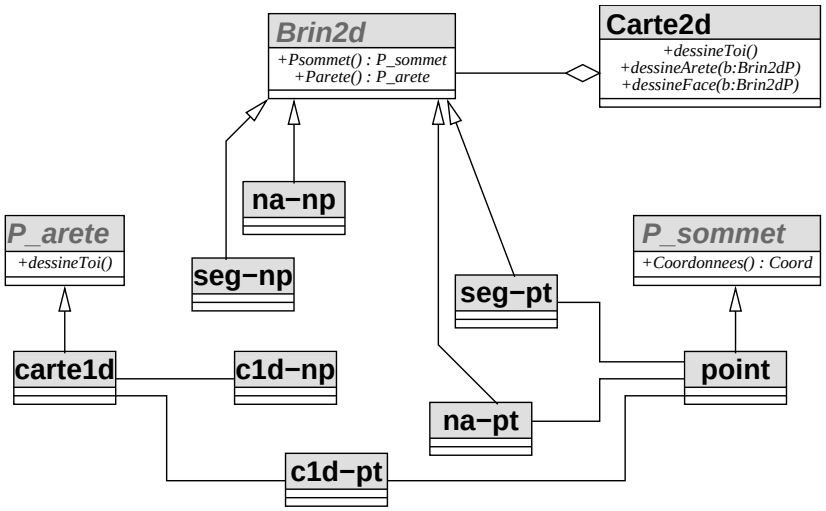


FIG. 7.6 – Le modèle objet de notre implantation de la carte topologique 2d.

La classe *Carte2d* est un ensemble de *Brin2d*. Cette classe *Brin2d* est virtuelle pure et n'a aucun attribut. Elle possède des méthodes permettant de récupérer le plongement arête et le plongement sommet qui sont virtuelles pures. Les deux classes virtuelles pures *P_arete* et *P_sommet* représentent les plongements arêtes et sommets. Le fait d'avoir ces classes génériques autorise le développeur à choisir la manière dont il désire réaliser ces plongements. Nous plongeons les sommets par un point géométrique, et les arêtes par des cartes 1d. Nous dérivons ensuite 6 classes de brins différentes de la classe *Brin2d*. Ces six classes s'expliquent par l'ensemble des possibilités de plongement d'un brin. Il peut porter un plongement sommet ou non, porter un plongement arête ou non. Dans ce dernier cas, soit le plongement arête est un segment, soit c'est une courbe ouverte. Nous différencions ces deux cas car une arête ayant pour plongement un segment ne pointe pas vers une 1-carte de plongement, étant donné que ces cartes sont ouvertes. Ces six classes ont pour nom sur le modèle objet de la figure 7.6 *seg-np*, *seg-pt*, *c1d-np*, *c1d-pt*, *na-pt* et *na-np*. La première partie du nom concerne le plongement arête : *seg* pour plongé segment, *c1d* pour plongé carte 1d, *na* pour non plongé arête ; et la deuxième partie concerne le plongement sommet : *pt* pour plongé sommet et *np* pour non plongé sommet. Six est donc le nombre de possibilités de combiner un des trois plongements arêtes avec un des deux plongements sommets. Grâce à ces six classes, chaque brin de la carte possède exactement les attributs qui lui sont nécessaires, il n'y a plus aucun attribut inutile et l'espace mémoire utilisé est donc optimal.

Ces six classes de brins définissent chacune les méthodes permettant de récupérer le plongement sommet et le plongement arête, qui suivant les cas (que nous ne détaillons pas ici) retourneront l'attribut porté par le brin, ou appelleront la méthode d'un autre brin appartenant à la même orbite. Cette implantation privilégie l'espace mémoire au détriment d'une légère perte en temps d'exécution. En effet, les méthodes permettant d'affecter un plongement à un brin devront, suivant le type de ce brin, créer un nouveau brin d'un autre type afin de pouvoir effectuer cette affectation. Cela entraîne donc une allocation mémoire suivie d'une désallocation. Ces modifications de type de brins se font de manière transparente sans que l'utilisateur ait besoin de les gérer. De plus, suivant les besoins en espace ou en temps, il est possible d'effectuer des compromis afin

de faire moins de changements de type de brins en regroupant certaines classes de brins, ce qui entraîne donc une légère perte en espace mémoire. Nous avons comparé cette implantation avec l'implantation classique de ce même modèle, ayant une seule classe brin avec deux pointeurs vers les éventuels plongements. Les résultats obtenus montrent un gain de l'ordre de 30% en espace mémoire, ce qui est très important pour de grosses images. De plus, la perte en temps d'exécution est minime, de l'ordre de 1%.

Pour l'implantation de la carte topologique 3d, nous n'avons pas développé le même modèle objet car nous n'avons pas implanté le plongement surface ouverte, arête ouverte et sommet. Pour réaliser cette implantation, nous aurions alors 18 classes de brins représentant toutes les combinaisons possibles entre les trois plongements. Mais comme pour la dimension 2, il est envisageable de regrouper certaines de ces classes, afin de diminuer leur nombre ainsi que les modifications de type, mais au prix d'une perte en espace mémoire. Pour implanter le plongement uniquement face, nous avons seulement deux types de brin suivant s'il pointe vers un brin d'une 2-G-carte ou non. Dans ce deuxième cas, c'est que le brin lui étant β_3-cousu porte cette information, que nous pouvons alors retrouver sans problème. Afin de représenter les 2-G-cartes de plongements, nous avons utilisé le même principe que pour la carte topologique 2d, et implanté les 6 classes de brins afin que ces cartes soient minimales en espace mémoire.

Nous ne détaillons pas plus les différents tests que nous avons effectués afin de comparer différentes implantations, des solutions intermédiaires, avec ou sans méthodes virtuelles, avec des allocateurs par blocs... Cette étude permet de comprendre un peu mieux les mécanismes objets du C++ et permet d'implanter la carte topologique de manière efficace dans ce langage. Bien entendu cette implantation ne peut pas être aussi efficace qu'une implantation « brutale » en langage machine, mais nous obtenons un bon compromis entre efficacité en espace, en temps et modularité. C'est bien notre objectif initial, afin d'obtenir un programme facilement extensible et modifiable.

7.7 Conclusion

Nous avons présenté dans ce chapitre quelques-uns de nos différents travaux, réalisés dans des domaines proches de notre cadre de recherche. Ces travaux ouvrent tous des perspectives de recherche ou des possibilités de développement logiciel. De plus, certains de ces travaux ont été réalisés en collaboration avec des chercheurs en traitement du signal, domaine que nous ne connaissions que très peu, et qui s'est avéré très enrichissant.

Moka, bien que n'étant pas totalement achevé, est opérationnel et laisse entrevoir de nombreuses possibilités d'évolutions. De plus, la plate forme logicielle qui va être développée autour du noyau doit intégrer de nombreuses applications différentes. Des logiciels de CAO, inspirés plus ou moins directement de Moka, des logiciels d'analyses d'images 3d, des logiciels de segmentation utilisant le modèle des cartes topologiques, un modeleur discret...

Pour la partie opérations sur la carte topologique, nous avons jusqu'ici étudié trois opérations de modification, mais nous envisageons maintenant de nombreuses autres possibilités. Ce travail est nécessaire afin d'offrir de nombreux outils aux utilisateurs éventuels de ce modèle. Une perspective nous tenant particulièrement à cœur est le développement d'une branche de Moka travaillant avec des cartes topologiques. Cela pose quelques problèmes, étant donné que nous avons

défini ce modèle dans le cadre de la représentation d'images segmentées, ce qui n'est plus le cas si nous travaillons maintenant en interactif. Mais les modifications à apporter doivent être pas trop importantes et ce projet doit donc être réalisable.

Notre étude sur l'implantation de la carte topologique est primordiale afin de pouvoir obtenir un programme efficace en temps d'exécution et n'occupant pas un espace mémoire « trop » important, tout en étant modulaire et extensible. Ces trois problématiques sont en opposition, mais notre solution consiste en des compromis entre ces trois aspects. Suivant les besoins spécifiques d'une application, il est possible de privilégier la complexité en temps ou en espace mémoire, en utilisant des solutions intermédiaires.

CHAPITRE 8

CONCLUSION ET PERSPECTIVES

Au cours de cette thèse, nous avons défini un modèle topologique minimal de représentation d'images segmentées en deux et trois dimensions : la carte topologique. Ce modèle est similaire en dimension deux à deux modèles existants, le graphe topologique des frontières et les cartes discrètes. L'apport de ce travail pour cette dimension est d'avoir introduit une nouvelle notion de niveau de simplification. Ces niveaux permettent de définir simplement et progressivement la carte topologique 2d, et ainsi d'étendre cette définition en dimension supérieure. De plus, ces niveaux facilitent l'étude de ce modèle, en permettant de se concentrer sur chaque niveau de simplification, et sur le passage entre deux niveaux successifs.

Cela nous a permis de mettre en évidence les problèmes de déconnexion pouvant survenir, principalement en dimension trois étant donné que c'est pour cette dimension qu'ils sont les plus délicats à résoudre. Leur résolution est facilitée car un problème particulier survient pour la construction d'un niveau précis, ce qui limite les configurations à étudier. Ces problèmes de déconnexion nous ont amené à étudier précisément les arêtes fictives et le rôle qu'elles jouent dans la définition de la carte topologique. Ce rôle est en effet primordial afin d'obtenir la représentation minimale. L'étude détaillée de l'opération de fusion nous a ensuite permis de valider la définition de la carte topologique, en montrant que la construction des différents niveaux de simplification n'entraînait pas de perte d'informations topologiques.

Nous avons ensuite défini des algorithmes d'extraction de ce modèle. Un premier algorithme simple découle immédiatement de notre définition progressive de la carte topologique. Puis, nous avons défini un algorithme optimal, en une seule passe de l'image et un nombre minimal d'opérations. Cet algorithme reprend le principe d'extraction à base de précodes, déjà utilisé par Christophe Fiorio. Mais les niveaux de simplification nous ont permis de factoriser de nombreux cas différents et de réduire ainsi ce nombre. Pour la dimension trois, nous sommes ainsi passé de 4140 cas différents à 129. Cette factorisation est grandement facilitée par les différents niveaux. En effet, nous étudions seulement pour chaque niveau les cas supplémentaires à traiter par rapport au niveau précédent.

Les travaux menés dans cette thèse peuvent être poursuivis dans différentes directions. Tout d'abord, nous devons étudier plus avant les deux points partiellement traités dans ce travail. L'étude des arêtes fictives et de leur gestion pour l'algorithme optimal d'extraction en dimension trois doit être approfondie afin de prouver que nous obtenons bien la représentation minimale.

En effet, la preuve a été donnée pour la définition de la carte topologique, mais il est nécessaire d'effectuer cette preuve pour l'algorithme optimal et les différents précodes. Le deuxième point concerne les arêtes fictives et l'étude de l'évolution des caractéristiques topologiques lors du décalage de ces arêtes. Nous devons prouver que ce décalage préserve le genre des objets représentés et conserve également la connexité de chaque face, fait que nous avons seulement abordé de manière intuitive.

Nous désirons ensuite nous intéresser à la définition de la carte topologique en dimension n. La définition progressive peut s'étendre sans problème particulier. Nous obtenons alors $2n-1$ niveaux de simplification, le niveau n étant la carte des frontières pouvant se plonger uniquement par ses sommets. Il faut ensuite s'intéresser aux problèmes de déconnexion. La technique utilisée en dimension deux et trois peut s'étendre sans problème en dimension quelconque. L'adjonction d'un arbre d'inclusion des régions permet de résoudre la déconnexion en dimension n. Pour les déconnexions des dimensions $n-1$ à deux, il suffit de conserver des éléments fictifs. Le point intéressant à étudier est l'extension du principe de décalage des arêtes fictives. C'est en effet ce principe qui permet l'obtention de la représentation minimale. Il s'agit d'étudier la manière de décaler les éléments fictifs de n'importe quelle dimension, ce qui doit pouvoir s'effectuer de manière similaire au décalage d'arête, mais également de voir les configurations possibles de ces éléments fictifs. Par exemple, en 4d, est-il possible d'avoir une arête fictive à l'intérieur d'une face fictive ? Comme pour la dimension trois, la gestion de ces éléments fictifs est primordiale et nécessite une étude approfondie.

Nous proposons également quelques perspectives autour des algorithmes d'extraction. Un premier point porte sur l'étude de l'ordre des fusions pour l'algorithme d'extraction naïf. En effet, nous avons vu que nous devions tester à nouveau l'ensemble des arêtes de la carte après une fusion de faces, afin de ne pas oublier des fusions. Il serait intéressant de déterminer un ordre pour ces fusions de faces permettant de s'affranchir de ces tests redondants, mais également des tests de déconnexion. Il faut pour cela trouver un ordre sur les fusions de faces conservant chaque face connexe, et commençant par les fusions aux extrémités de la carte. En effet, une fusion de faces le long d'une arête incidente à un sommet de degré un ne peut pas entraîner de déconnexion. De plus, cette fusion peut entraîner la création d'un nouveau sommet de degré un, et la prochaine fusion à effectuer pourrait alors être le long de l'arête lui étant incidente. La définition d'un tel ordre entraînerait la diminution de la complexité de l'algorithme naïf. Il deviendrait linéaire au lieu d'être quadratique.

Pour ce qui concerne l'algorithme optimal, il serait intéressant d'obtenir les formules combinatoires permettant de calculer le nombre de précodes variétés en dimension n, mais également les formules permettant de calculer le nombre de précodes supplémentaires à traiter pour chaque niveau de simplification. Nous avons défini la formule correspondant au niveau 1, qui est simplement 2^n, ainsi que celles des niveaux 2 et n. Mais il reste à déterminer les nombreuses autres formules. De plus, nous devons nous intéresser aux possibilités de factorisation des précodes similaires. Nous avons en effet introduit la notion de précodes isomorphes par rotation qui nous a permis de réduire le nombre de cas à étudier, mais il existe peut-être d'autres possibilités.

Nous avons également plusieurs projets qui consistent à poursuivre les travaux présentés au chapitre 7. Nous devons définir différentes opérations de modification de la carte topologique 3d, afin de mettre à la disposition d'utilisateurs potentiels un panel d'outils leur permettant de travailler avec ce modèle. Nous allons également nous intéresser à la manière de développer une

branche de Moka basée sur les cartes topologiques. Ces deux points sont liés, étant donné que pour définir un modeleur basé sur la carte topologique, nous avons besoin de différentes opérations de modification. Les résultats de ces travaux pourront ensuite s'intégrer à un logiciel de segmentation 3d, comme par exemple le projet de détermination du volume tumoral cérébral, afin de permettre des opérations interactives autorisant un expert à corriger éventuellement le résultat de la segmentation.

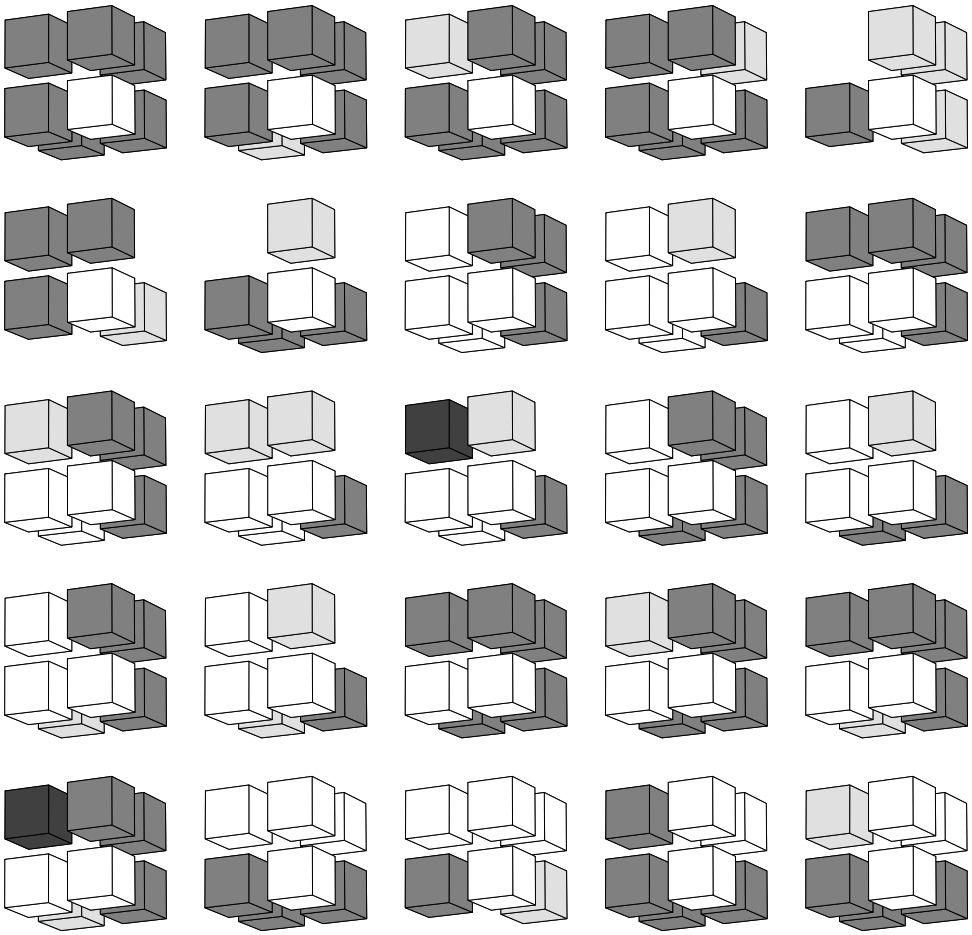
Nous allons également poursuivre nos travaux afin d'utiliser la carte topologique, en deux et trois dimensions, dans des algorithmes de segmentation où pour faire du raffinement de segmentation. Nous voulons étudier comment utiliser les informations topologiques fournies par notre modèle afin de définir des critères de segmentation. Nous envisageons également d'autres utilisations de la carte topologique, par exemple lors d'une phase de pré-segmentation pour distinguer les zones texturées ou lors d'une phase de post-segmentation pour supprimer les petites régions que nous pouvons considérer comme du bruit.

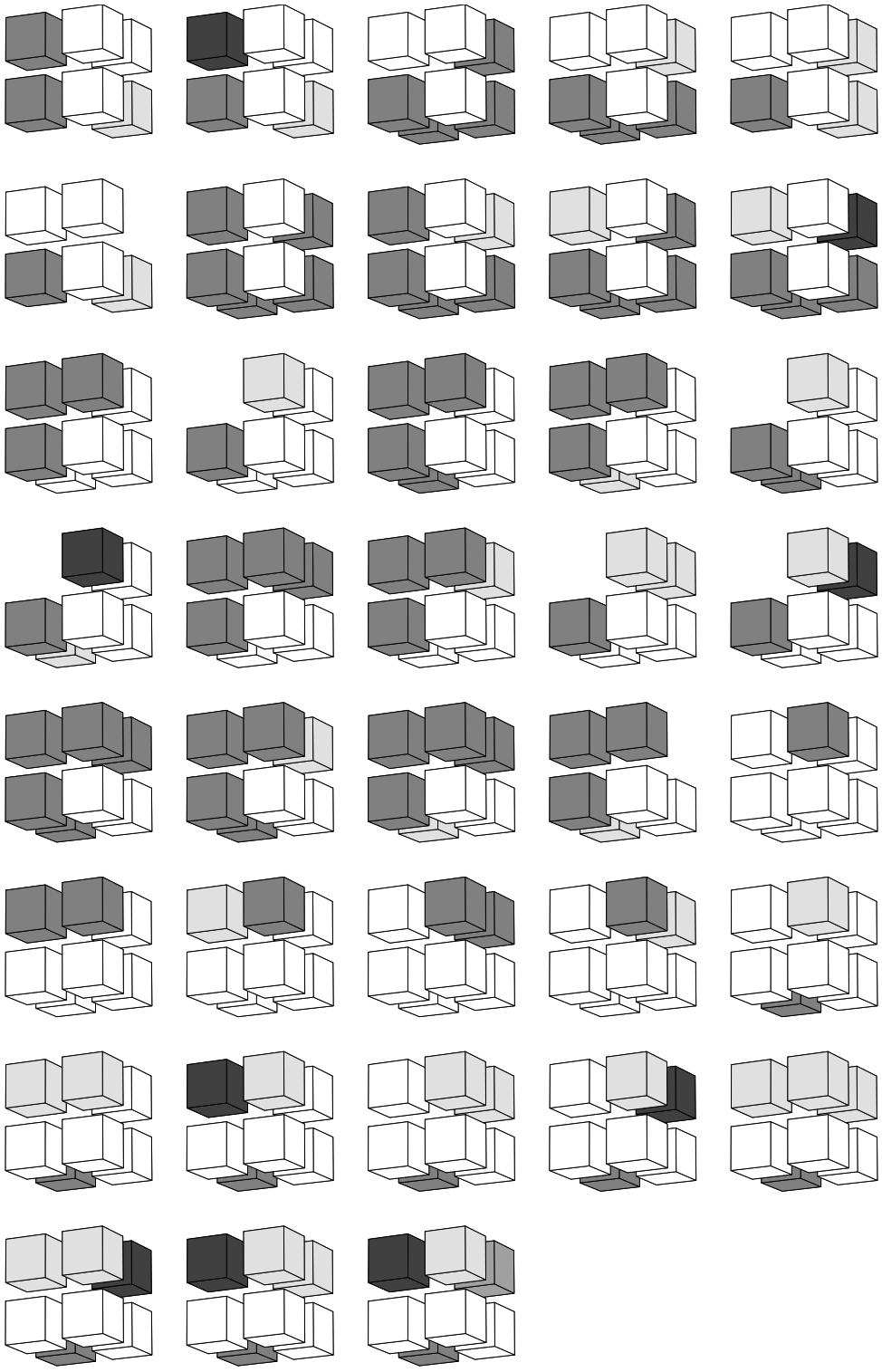
Un autre point de recherche qui nous intéresse particulièrement, mais que nous n'avons pas du tout abordé pour le moment, est la reconnaissance des plans discrets afin de représenter les surfaces de plongement. En effet, le plongement utilisé actuellement est composé de 2-G-cartes représentant uniquement des faces parallèles aux plans orthotropes. La polyédrisation de ces surfaces entraînerait la diminution de l'espace mémoire nécessaire à leurs représentations, mais permettrait également d'utiliser des algorithmes de géométrie discrète de visualisation, de lissage. Dans la même voie de recherche, nous allons collaborer au développement d'un modeleur discret. Il doit avoir des fonctionnalités similaires à celles proposées par Moka, mais doit pouvoir utiliser des plongements discrets. Il est basé sur une évolution du noyau générique de 3-G-carte permettant de représenter les non variétés. Ce projet ressemblant à Moka sous certains aspects est très proche de notre travail de recherche et des préoccupations de cette thèse, étant donné que nous représentons des images, donc des assemblages de cubes en dimension trois.

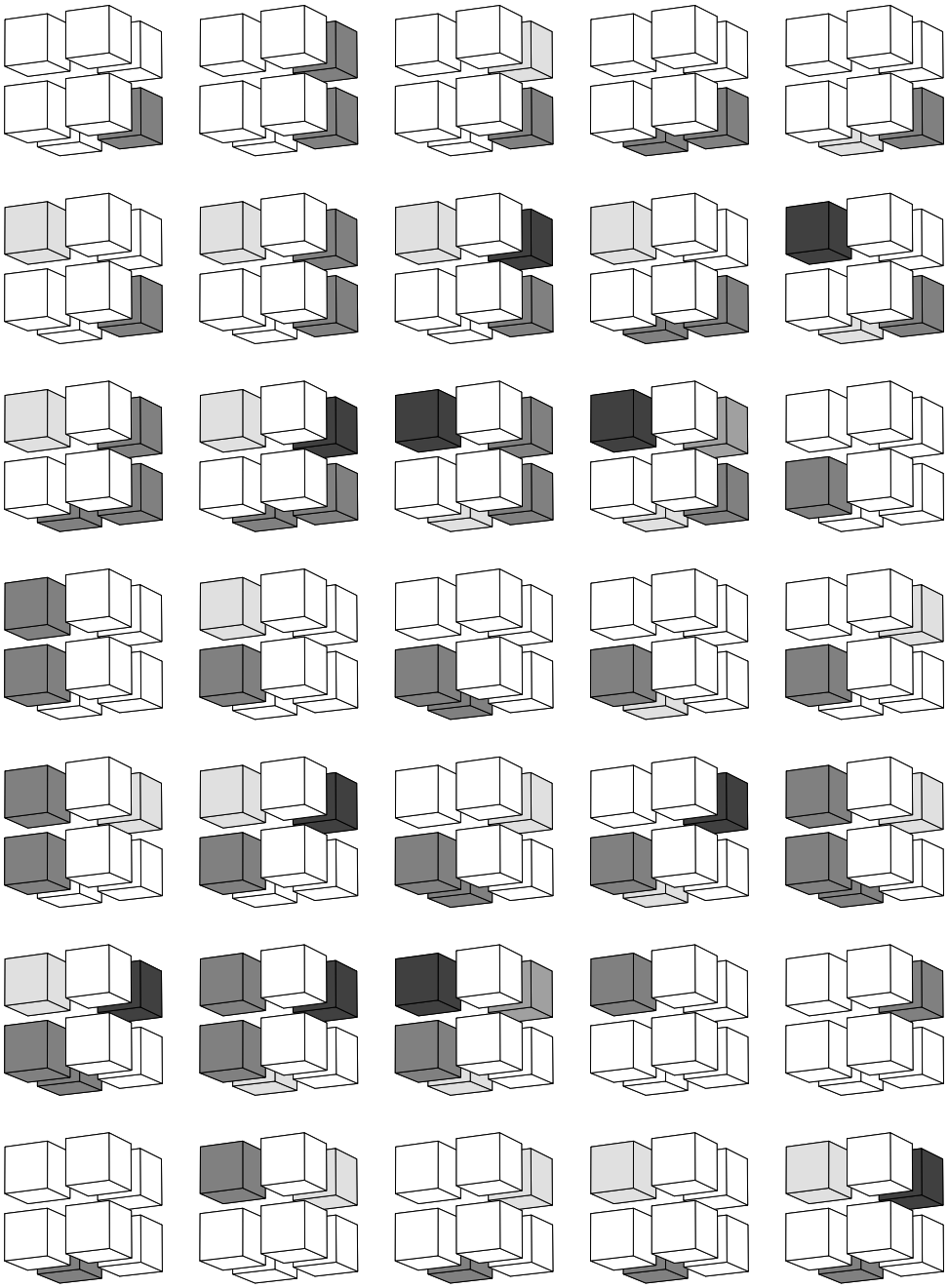
Un autre thème de recherche que nous allons très prochainement aborder concerne les pyramides combinatoires. En effet, les travaux autour de ces pyramides sont très proches de notre travail, et il est intéressant d'en étudier les similarités et les différences. Nous devons pour cela étudier les opérations de contraction et suppression d'arêtes en regard de l'opération de fusion. Sous certaines conditions, ces opérations sont en effet équivalentes. De plus, les travaux actuels portent pour le moment sur la dimension deux, et nous pourrions apporter la possibilité de passer en dimension supérieure.

ANNEXE A

LES 98 PRÉCODES DE NIVEAU 4

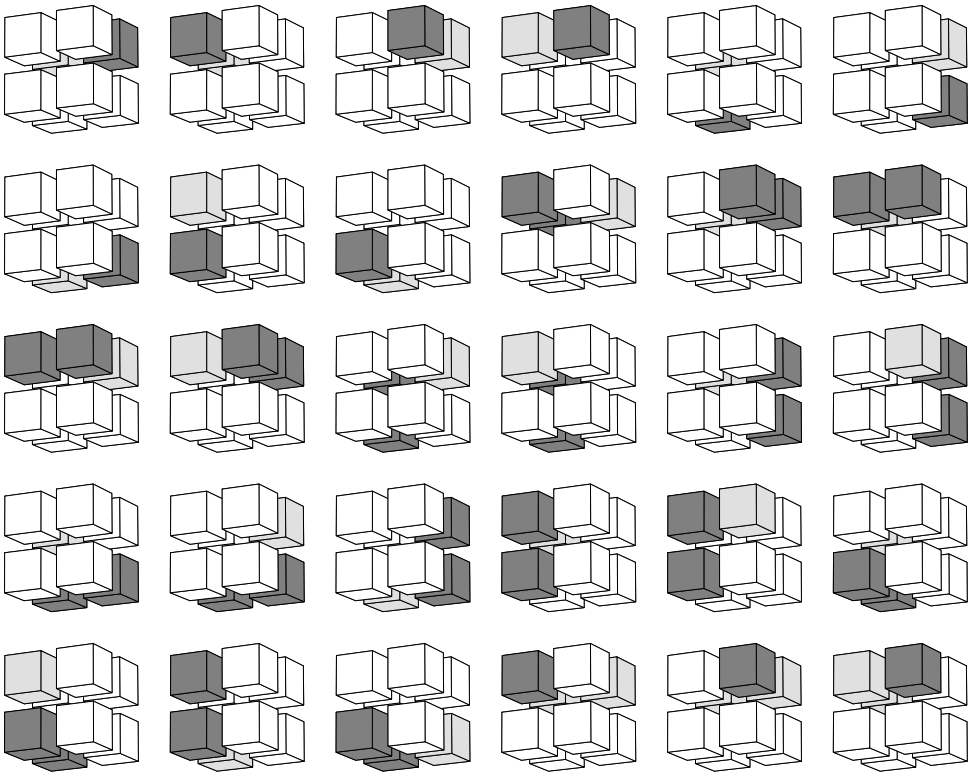


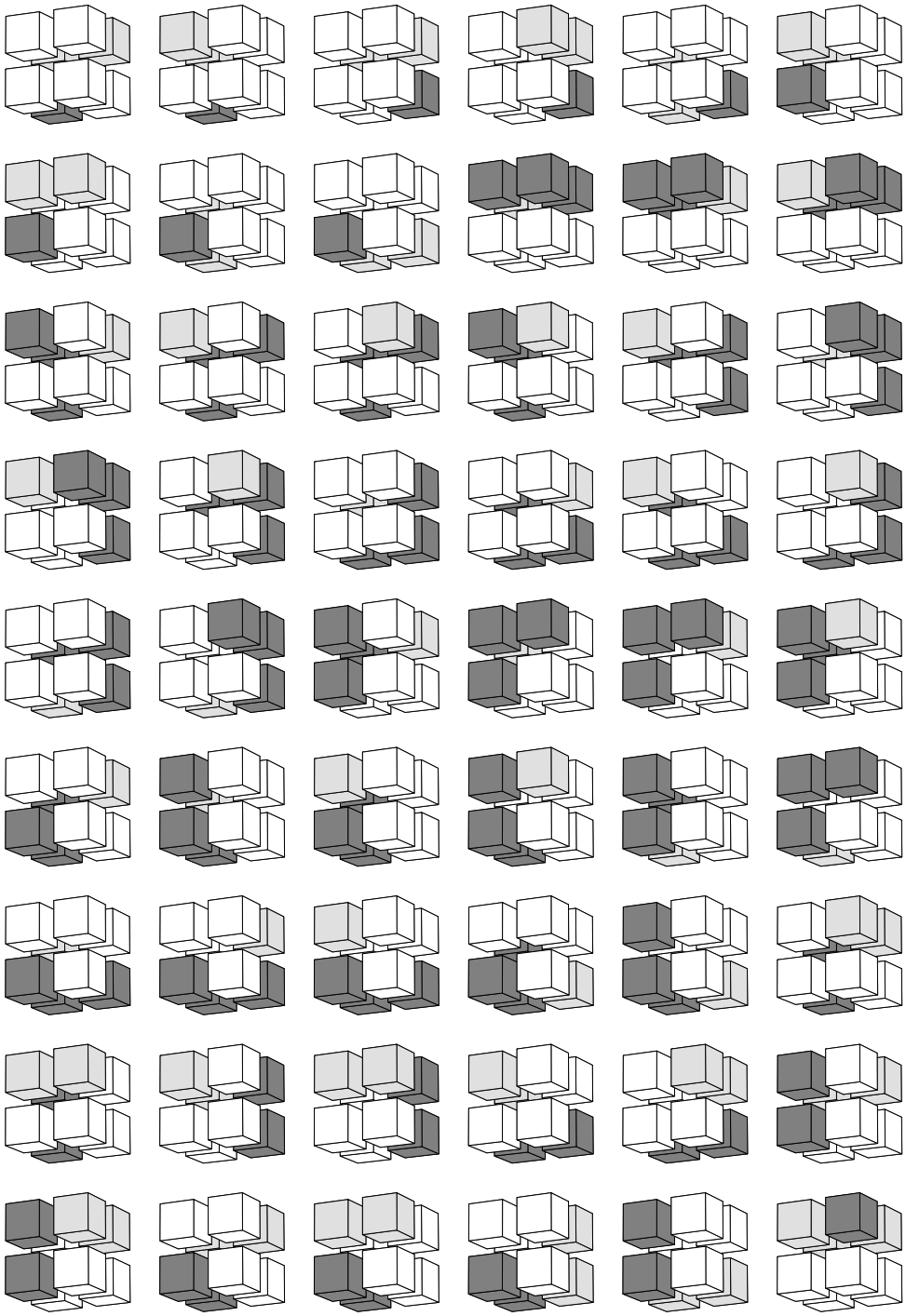


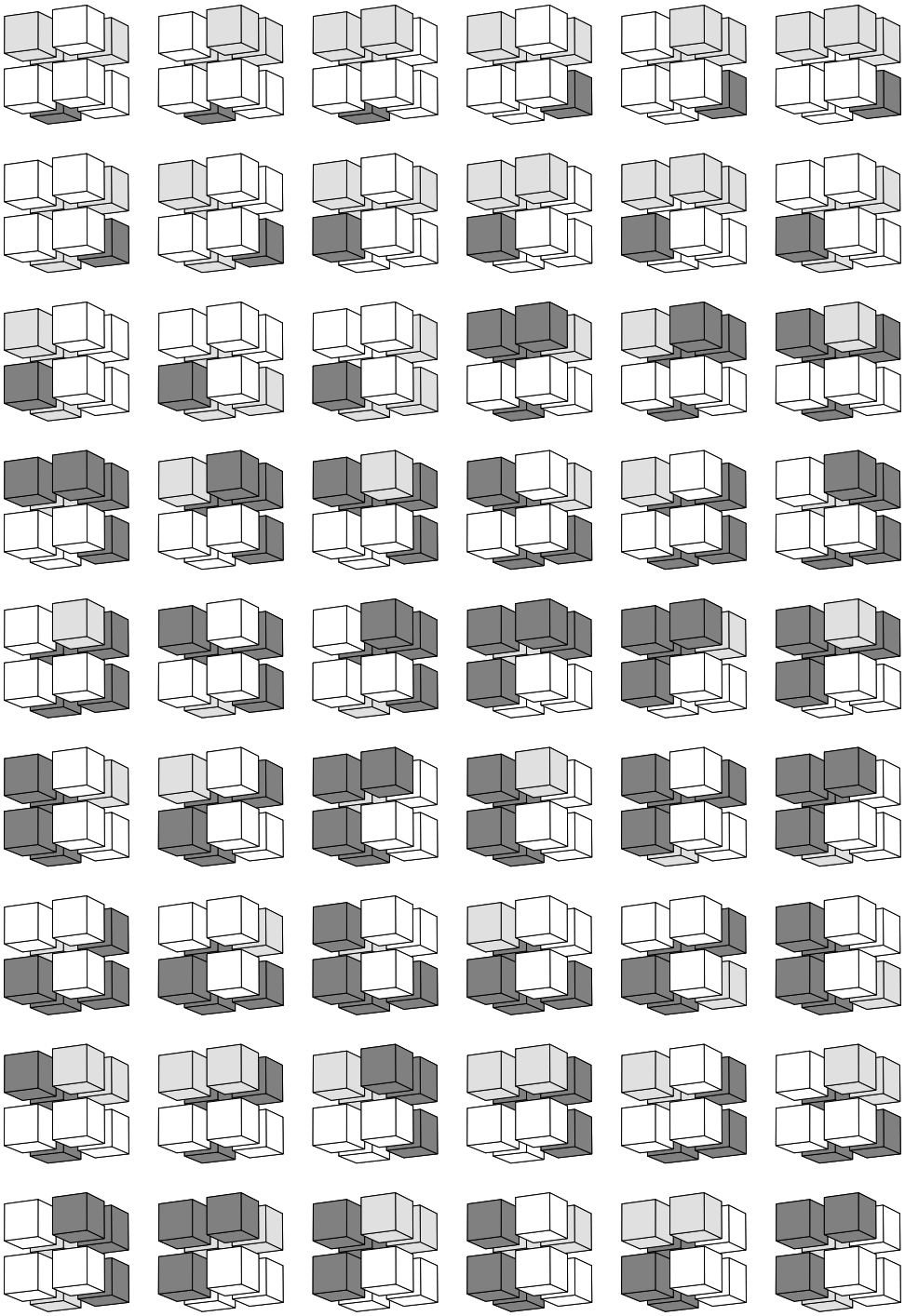


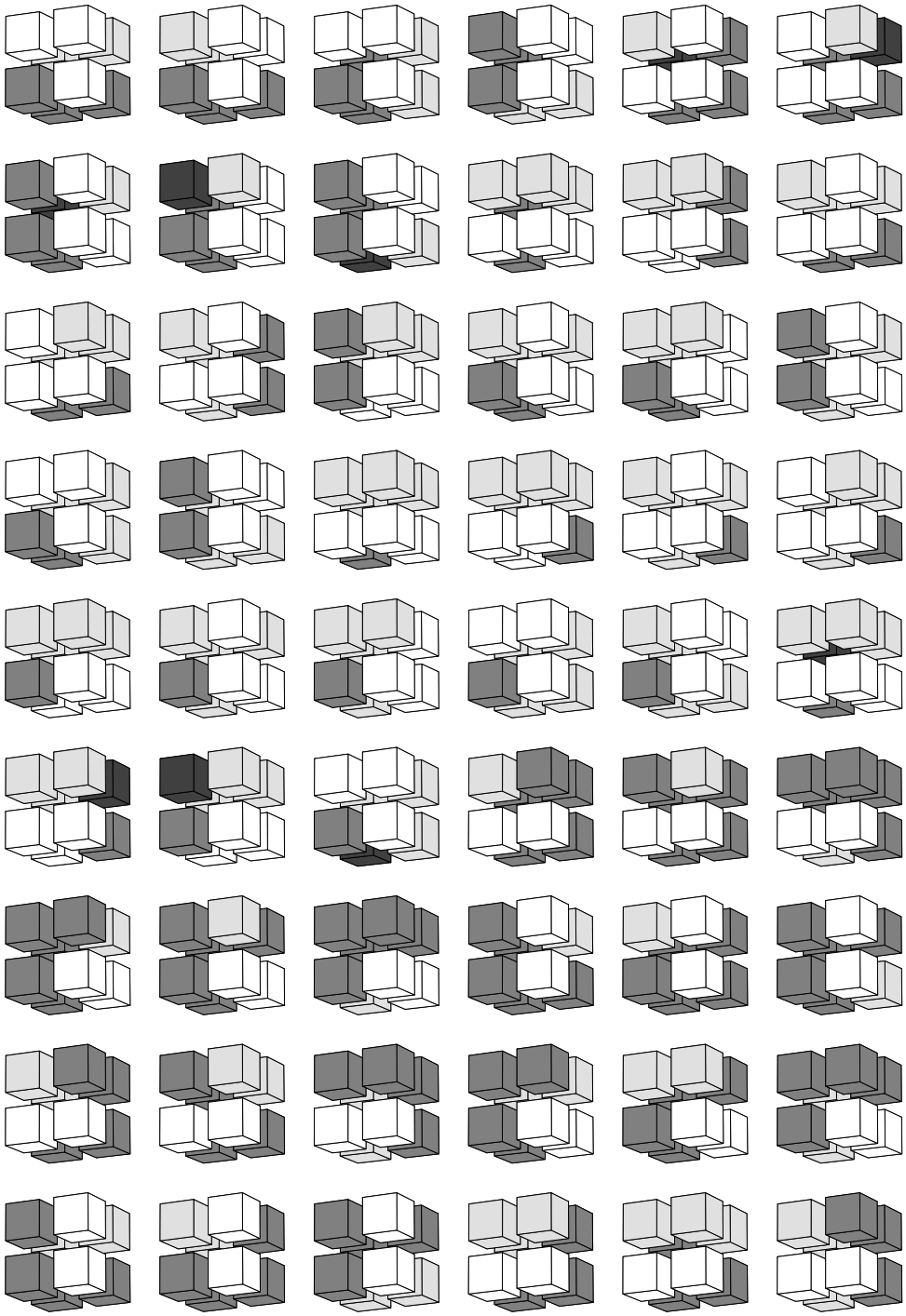
ANNEXE B

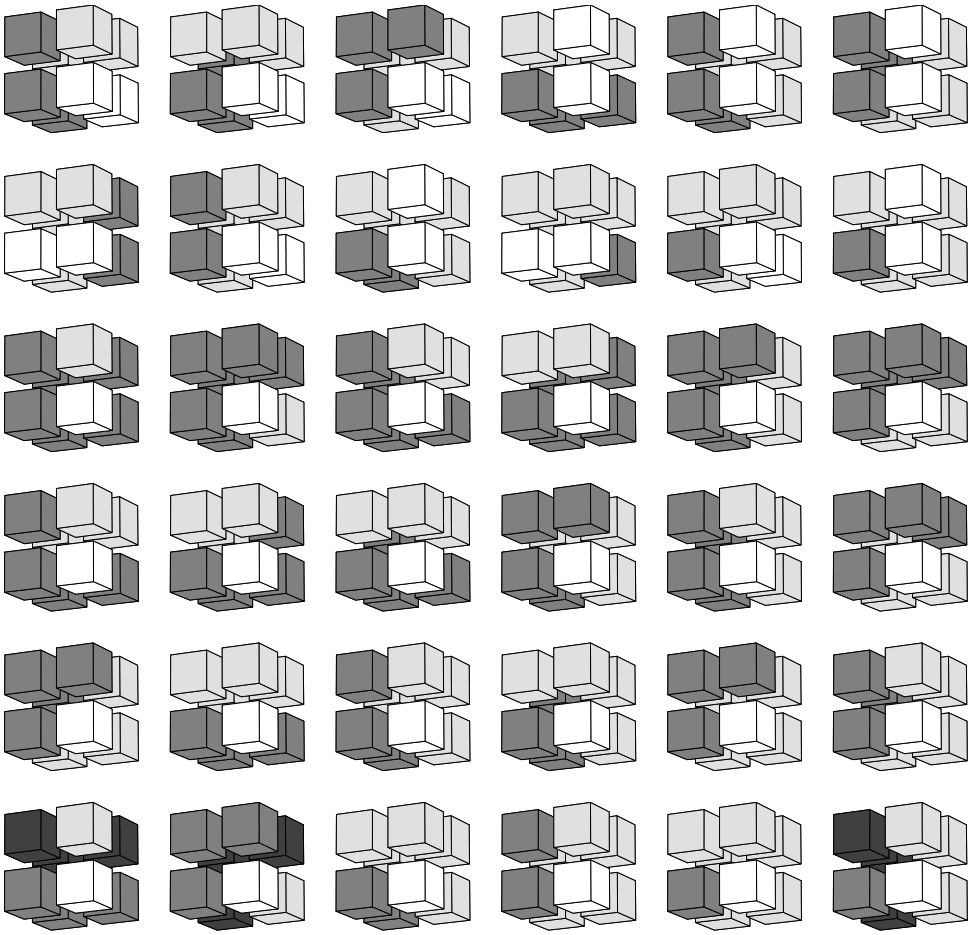
LES 228 PRÉCODES DE NIVEAU 5











BIBLIOGRAPHIE

[AAF95] E. Ahronovitz, J.P. Aubert, and C. Fiorio. The star-topology : a topology for image analysis. In *Discrete Geometry for Computer Imagery*, pages 107–116, september 1995.

[ADFF85] S. Ansaldi, L. De Floriani, and B. Falcidieno. Geometric modeling of solid objects by using a face adjacency graph representation. *Comput. Graph.*, 19(3) :131–139, 1985. Proc. SIGGRAPH '85.

[AFG99] E. Ahronovitz, C. Fiorio, and S. Glaize. Topological operators on the topological graph of frontiers. In *Discrete Geometry for Computer Imagery*, number 1568 in Lecture Notes in Computer Science, pages 207–217, Marne-la-Vallée, France, 1999.

[AFH81] E. Artzy, G. Frieder, and G.T. Herman. The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm. *Computer Graphics and Image Processing*, 15 :1–24, 1981.

[Ago76] M. Agoston. *Algebraic topology : a first course*. Lecture Notes in Pure and Applied Mathematics. Marcel Dekker, New York, 1976.

[AK88] D. Arques and P. Koch. Définition et implémentation de pavages dans l'espace. Technical Report 46, Laboratoire d'Informatique, UFR Sciences et Techniques, Besançon, France, août 1988.

[AK89] D. Arques and P. Koch. Modélisation de solides par les pavages. In *Pixim 89*, pages 47–61, Paris, 1989.

[Ale61] P.S. Aleksandrov. *Elementary concepts of topology*. Dover Publications Inc., New York, 1961.

[Bau75] B. Baumgart. A polyhedron representation for computer vision. In *Proc. AFIPS Nat. Conf.*, volume 44, pages 589–596, 1975.

[BB98] J.P. Braquelaire and L. Brun. Image segmentation with topological maps and inter-pixel representation. *Journal of Visual Communication and Image Representation*, 9(1) :62–79, march 1998.

[BD97] L. Brun and J.P. Domenger. A new split and merge algorithm with topological maps and inter-pixel boundaries. In *The fifth International Conference in Central Europe on Computer Graphics and Visualization*, february 1997.

[BD99] J.P. Braquelaire and J.P. Domenger. Representation of segmented images with discrete geometric maps. *Image and Vision Computing*, 17(10) :715–735, 1999.

[BDB97] L. Brun, J.P. Domenger, and J.P. Braquelaire. Discrete maps : a framework for region segmentation algorithms. In *Workshop on Graph based representations*, Lyon, april 1997. IAPR-TC15. published in Advances in Computing (Springer).

[BDF00] Y. Bertrand, G. Damiand, and C. Fiorio. Topological encoding of 3d segmented images. In *Discrete Geometry for Computer Imagery*, number 1953 in Lecture Notes in Computer Science, pages 311–324, Uppsala, Sweden, december 2000.

[BDFL92] Y. Bertrand, J.F. Dufourd, J. Françon, and P. Lienhardt. Modélisation volumique à base topologique. Rapport de recherche 92-16, Université Louis Pasteur, Centre de Recherche en Informatique, Strasbourg, France, 1992.

[Ber92] Y. Bertrand. *Spécification algébrique et réalisation d'un modeleur interactif d'objets géométriques volumiques*. Thèse de doctorat, Université Louis-Pasteur de Strasbourg, 1992.

[BF70] C. Brice and C. Fennema. Scene analysis using regions. *Artificial intelligence*, 1(3) :205–226, 1970.

[BF97] Y. Bertrand and J. Françon. Variétés combinatoires 2d pour la représentation d'objets par leur bord : étude statistique des cellules. Rapport de recherche 97-08, Université Louis-Pasteur, Centre de Recherche en Informatique, Strasbourg, France, mars 1997.

[BFP99] Y. Bertrand, C. Fiorio, and Y. Pennaneach. Border map : a topological representation for nd image analysis. In *Discrete Geometry for Computer Imagery*, number 1568 in Lecture Notes in Computer Science, pages 242–257, Marne-la-Vallée, France, 1999.

[BK99] L. Brun and W.G. Kropatsch. Dual contraction of combinatorial maps. In *Workshop on Graph based representations*, Austria, may 1999. IAPR-TC15.

[BK00] L. Brun and W.G. Kropatsch. The construction of pyramids with combinatorial maps. Technical report 63, Institute of Computer Aided Design, Vienna University of Technology, Austria, june 2000. URL : http ://www.prip.tuwien.ac.at/.

[BK01] L. Brun and W.G. Kropatsch. Contraction kernels and combinatorial maps. In *Workshop on Graph based representations*, pages 12–21, Ischia, Italy, may 2001. IAPR-TC15.

[Bou01] P. Bourdon. Segmentation de régions par approches stochastiques et calcul de cartes des frontières. Mémoire de Dea, Université de Poitiers, 2001.

[Bri89] E. Brisson. Representing geometric structures in d dimensions : topology and order. In 5^{th} *Annual ACM Symposium on Computational Geometry*, pages 218–227, Saarbrücken, Germany, 1989.

[Bro82] J.D. Browning. Segmentation of pictures into regions with tile-by-tile method. *Pattern Recognition*, 15(1) :1–10, 1982.

[Bru96] L. Brun. *Segmentation d'images couleur à base topologique*. Thèse de doctorat, Université Bordeaux I, décembre 1996.

[CAFMF01] A.S. Capelle, O. Alata, C. Fernandez-Maloigne, and J.C. Ferrié. Unsupervised algorithm for the segmentation of three-dimensional magnetic resonance brain images. In *ICIP'01, IEEE International Conference on Image Processing*, october 2001. Thessaloniki, Grece.

[Cay01] C. Cayrous. Développements autour d'un noyau de cartes généralisées : Modélisation 3d d'environnement urbain et logiciel d'aide au diagnostic par IRM de tumeurs cérébrales. Diplôme d'etudes supérieur spécialisées « DESSTAUP », Université de Poitiers, 2001.

[Cha95] P. Charnier. *Outils algorithmiques pour le codage interpixel et ses applications*. Thèse de doctorat, Université Montpellier II, 13 janvier 1995.

[Cor73] R. Cori. *Un code pour les graphes planaires et ses applications*. PhD thesis, Université Paris VII, 1973.

[Cor75] R. Cori. Un code pour les graphes planaires et ses applications. In *Astérisque*, volume 27. Soc. Math. de France, Paris, France, 1975.

[DF99] G. Damiand and C. Fiorio. Une expérience de rétro-conception objet d'une application graphique. Rapport de recherche 99116, Université Montpellier II, Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, France, 1999.

[DL87] D.P. Dobkin and M.J. Laszlo. Primitives for the manipulation of three-dimensional subdivisions. In *Proc. 3rd Symp. on Computational Geometry*, pages 86–99, Waterloo, Ontario, Canada, june 1987.

[Dom92] J.P. Domenger. *Conception et implémentation du noyeau graphique d'un environnement* $2D1/2$ *d'édition d'images discrètes*. Thèse de doctorat, Université Bordeaux I, avril 1992.

[DR02] G. Damiand and P. Resch. Topological map based algorithms for 3d image segmentation. In *Discrete Geometry for Computer Imagery*, LLNCS, Bordeaux, France, april 2002.

[DRS80] R.C. Dyer, A. Rosenfeld, and H. Samet. Region representation : boundary codes from quadtrees. *Communications of the ACM*, 23(3) :171–179, march 1980.

[Ede87] H. Edelsbrunner. Algorithms in combinatorial geometry. In W. Brauer, G. Rozenberg, and A. Salomaa, editors, *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1987.

[Edm60] J. Edmonds. A combinatorial representation for polyhedral surfaces. *Notices of the American Mathematical Society*, 7, 1960.

[Elt94] H. Elter. *Etude de structures combinatoires pour la représentation de complexes cellulaires*. Thèse de doctorat, Université Louis-Pasteur de Strasbourg, septembre 1994.

[Fio95] C. Fiorio. *Approche interpixel en analyse d'images : une topologie et des algorithmes de segmentation*. Thèse de doctorat, Université Montpellier II, 24 novembre 1995.

[Fio96] C. Fiorio. A topologically consistent representation for image analysis : the frontiers topological graph. In *Discrete Geometry for Computer Imagery*, number 1176 in Lecture Notes in Computer Science, pages 151–162, Lyon, France, november 1996.

[GS85] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Trans. Graph.*, 4(2) :74–123, 1985.

[GU89] D. Gordon and J.K. Udupa. Fast surface tracking algorithm in three-dimensional binary images. *Computer Vision, Graphics, and Image Processing*, 45 :196–214, 1989.

[HW83] G.T. Herman and D. Webster. A topological proof of a surface tracking algorithm. *Computer Vision, Graphics, and Image Processing*, 23 :162–177, 1983.

[Jac70] A. Jacques. Constellations et graphes topologiques. In *Combinatorial Theory and Applications*, volume 2, pages 657–673, 1970.

[KF81] N. Keskes and O. Faugeras. Surface simple dans z^3. In 3*ème Congrès Reconnaissance des Formes et d'Intelligence Artificielle*, pages 718–728, Septembre 1981.

[KKM90] E. Khalimsky, R. Kopperman, and P.R. Meyer. Boundaries in digital planes. *Journal of Applied Mathematics and Stochastic Analysis*, 3(1) :27–55, 1990.

[KKM91] T.Y. Kong, R. Kopperman, and P.R. Meyer. A topological approach to digital topology. *American Mathematical Monthly*, 98(10) :901–917, 1991.

[KM95] W.G. Kropatsch and H. Macho. Finding the structure of connected components using dual irregular pyramids. In *Discrete Geometry for Computer Imagery*, pages 147–158, *invited lecture*, september 1995.

[Kov89] V.A. Kovalevsky. Finite topology as applied to image analysis. *Computer Vision, Graphics, and Image Processing*, 46 :141–161, 1989.

[KR89] T.Y. Kong and A. Rosenfeld. Digital topology : introduction and survey. *Computer Vision, Graphics, and Image Processing*, 48 :357–393, 1989.

[Kro95] W.G. Kropatsch. Building irregular pyramids by dual-graph contraction. *Vision, Image and Signal Processing*, 142(6) :366–374, december 1995.

[KU92] T.Y. Kong and J.K. Udupa. A justification of a fast surface tracking algorithm. *Computer Vision, Graphics, and Image Processing : Graphical Models and Image Processing*, 54 :162–170, 1992.

[Lie88] P. Lienhardt. Extension of the notion of map and subdivisions of a three-dimensional space. In 5^{th} *Symposium on the Theoretical Aspects of Computer Science*, volume 294 of *Lecture Notes in Computer Science*, pages 301–311, Bordeaux, France, february 1988.

[Lie89] P. Lienhardt. Subdivision of n-dimensional spaces and n-dimensional generalized maps. In *Proc.* 5^{th} *Annual ACM Symposium on Computational Geometry*, pages 228–236, Saarbrücken, Germany, 1989.

[Lie91] P. Lienhardt. Topological models for boundary representation : a comparison with n-dimensional generalized maps. *Commputer Aided Design*, 23(1) :59–82, 1991.

[Lie93] P. Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manitolds. Research Report R 93-04, Université Louis Pasteur, département d'informatique, Strasbourg, France, mars 1993.

[Lie94] P. Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal of Computational Geometry and Applications*, 4(3) :275–324, 1994.

[Lie97] P. Lienhardt. Aspects in topology-based geometric modeling. In *Discrete Geometry for Computer Imagery*, volume 1347 of *Lecture Notes in Computer Science*, pages 33–48. Springer, december 1997. (invited speaker).

[MK95] H. Macho and W.G. Kropatsch. Finding connected components with dual irregular pyramids. In *Proc. of 19th OAGM and 1st SDVR Workshop*, pages 313–321, 1995.

[PJ97] J.G. Pailloncy and J.M. Jolion. The frontier-region graph. In *Workshop on Graph based representations*, volume 12 of *Computing Supplementum*, pages 123–134. Springer, april 1997.

[PRW82] M. Pietikainen, A. Rosenfeld, and I. Walter. Split and link algorithms for image segmentation. *Pattern Recognition*, 15(4) :287–298, 1982.

[Res01a] P. Resch. Algorithmes pour la manipulation des cartes topologiques en 2 et 3 dimensions. Mémoire de Dea, Université Montpellier II, juin 2001.

[Res01b] P. Resch. Algorithmes pour la manipulation des cartes topologiques en 2 et 3 dimensions. Annexe technique, Université Montpellier II, juin 2001.

[RK82] A. Rosenfeld and A.C. Kak. *Digital picture processing*, volume 2. Academic Press, New York, 1982.

[RKW91] A. Rosenfeld, T.Y. Kong, and A.Y. Wu. Digital surfaces. *Computer Vision, Graphics, and Image Processing : Graphical Models and Image Processing*, 53(4) :305–312, july 1991.

[RO89] J. Rossignac and M. O'Connor. SGC : A dimension-independent model for pointsets with internal structures and incomplete boundaries. In M. Wozny, J. Turner, and K. Preiss, editors, *Geometric Modeling for Product Engineering*. Elsevier Science, 1989.

[Ros74] A. Rosenfeld. Adjacency in digital pictures. *Information and Control*, 26(1) :24–33, 1974.

[RP68] A. Rutovitz and J.L. Pfaltz. Data structures for operations on digital images. *Pictorial Pattern Recognition*, pages 105–133, 1968.

[Sam80] H. Samet. Region representation : quadtrees from boundary codes. *Communications of the ACM*, 23 :163–170, 1980.

[Sam84] H. Samet. The quadtree and related hierarchical data structures. *Computing surveys*, 16(2) :187–260, 1984.

[Spe91] J.C. Spehner. Merging in maps and in pavings. *Theoretical Computer Science*, 86(2) :205–232, September 1991.

[Tak91] T. Takala. A taxonomy on geometric and topological models. In C.Pienovi B.Falcidieno, I.Herman, editor, *Computer Graphics and Mathematics*, pages 147–171. Springer-Verlag, 1991.

[Tut63] W.T. Tutte. A census of planar maps. *Canad. J. Math.*, 15 :249–271, 1963.

[Wei85] K. Weiler. Edge-based data structures for solid modeling in curved-surface environments. *IEEE Computer Graphics and Applications*, 5(1) :21–40, 1985.

[Wei88] K. Weiler. The radial edge structure : a topological representation for non-manifold geometric boundary modeling. In M.J. Wozny, H.W. McLaughlin, and J.L. Encarnacao, editors, *Geometric Modeling for CAD Applications*, pages 217–254. Elsevier Science, 1988.

[Wil95] D. Willersinn. *Irreguläre Kurvenpyramiden : ein Schema für perzeptuelle Organisation*. PhD thesis, Technische Universität Wien, march 1995.

LISTE DES ALGORITHMES

INDEX

Symbols

A

B

C

O

P

Q

R

S

T

U

V