Modèles topologiques

Sommaire

3.1 Introduction

Le chapitre précédent a montré que pour simuler des évolutions topologiques d'objets physiques maillés, il était préférable de se baser sur un modèle dédié spécifiquement à la représentation de la topologie de l'objet. La modélisation géométrique, notamment à base topologique, nous offre une panoplie de modèles topologiques que ce chapitre se propose de brièvement présenter. Dans l'ensemble de ces approches, notre choix s'est porté sur le modèle des cartes généralisées, que nous présentons de façon plus approfondie dans ce chapitre. Ce choix est en grande partie motivé par l'utilisation de l'outil Jerboa, qui propose de manipuler les cartes généralisées à l'aide d'un langage à base de règles de transformation de graphes. Durant nos travaux, nous nous sommes entièrement reposé sur cet outil, à la fois pour la modélisation et la simulation de nos modèles. Aussi, ce chapitre se termine par la présentation de l'outil Jerboa proprement dit et principalement de la syntaxe de son langage de règles.

3.2 Modélisation géométrique à base topologique

La modélisation à base topologique vise à proposer des structures pour représenter graphiquement des objets sur ordinateur. Ces structures permettent de représenter, manipuler et modifier des objets géométriques structurés en différentes

dimensions. La diversité de ces structures est due à l'existence de plusieurs classes d'objets à représenter dans différents domaines d'applications comme l'architecture, la géologie, la mécanique, le domaine médical, les jeux vidéo...

En modélisation géométrique à base topologique, on différencie la topologie de l'objet de sa géométrie. La structure topologique est la décomposition de l'objet en un ensemble de cellules topologiques (sommets, arêtes, faces...). Une cellule de dimension i est appelée **i-cellule**, où i est inférieur ou égal à la dimension de l'objet considéré. Par exemple pour un objet de dimension 3, on peut définir les volumes en tant que *3-cellules*, les faces comme *2-cellules*, les arêtes comme *1-cellules* et finalement les sommets comme *0-cellules*. Par ailleurs, dans le cas des objets fermés, pour tout $i \geq 1$, les bords d'une i-cellule sont des cellules de dimension $i - 1$. Ces derniers constituent sa frontière. Citons par exemple un volume de dimension 3 qui est délimité par des faces, en d'autres termes des 2-cellules. Ces cellules sont liées par des relations de voisinage. Deux types de relations topologiques peuvent se présenter :

- **les relations d'incidence** : ces relations relient deux cellules voisines avec des dimensions différentes (*i.e* relient une cellule aux cellules de son bord). Par exemple, les arêtes sont incidentes à la face à laquelle elles appartiennent.

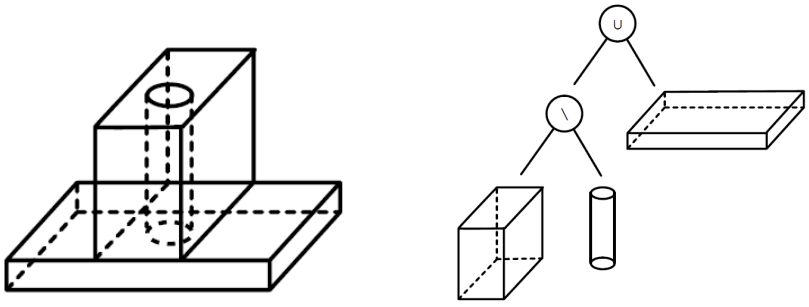
- **les relations d'adjacence** : ces relations relient deux cellules voisines de même dimension n qui partagent un bord (par exemple deux faces voisines qui sont liées par une arête ou deux volumes reliées par un sommet)

Le modèle géométrique de l'objet vise alors à compléter le modèle topologique en lui ajoutant une forme, une position, etc. Dans la littérature, il existe deux grandes familles de modèles généralement utilisés pour représenter les objets : les modèles de construction géométrique de solides (CSG) et les modèles de représentation par les bords (B-Rep).

Les modèles de construction géométrique de solides (CSG)

Le modèle de construction géométrique de solides a été introduit par Laidlaw *et al.* [LTH86]. Il représente un objet par un arbre dont les feuilles sont les composantes élémentaires et les nœuds sont les opérations booléennes (la différence, l'union, l'intersection...) comme l'illustre la figure 3.1. Il aide à construire rapidement la forme d'un objet, mais son inconvénient est qu'il ne représente pas explicitement l'objet construit mais uniquement son processus de construction. De plus, la représentation d'un objet avec cette méthode n'est pas unique, plusieurs constructions pouvant aboutir au même objet.

Les CSG sont généralement utilisés pour modéliser un objet via son processus constructif. Cela peut être utile en modélisation, voire en conception et usinage par ordinateur, mais ce modèle se révèle inadapté pour la simulation de solides déformables.



(a) Objet (b) Arbre CSG

FIGURE 3.1 – Exemple de représentation d'un objet par arbre CSG.

Les modèles de représentation par les bords

Au contraire des modèles CSG, la représentation par les bords (*B-rep*) sert à représenter un objet par le bord qui le délimite, c'est à dire sa surface en 3D, ou sa courbe en 2D. Souvent, ces modèles sont décomposés en deux parties, une partie topologique définie par les faces, les arêtes, les sommets et leurs relations d'adjacence et d'incidence et une partie géométrique définie par les points, les courbes et les surfaces. Un exemple de représentation d'un objet par les bords est donné à la figure 3.2.

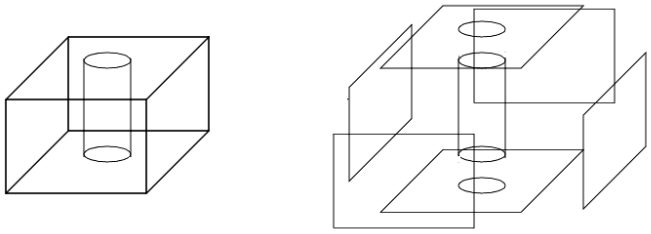
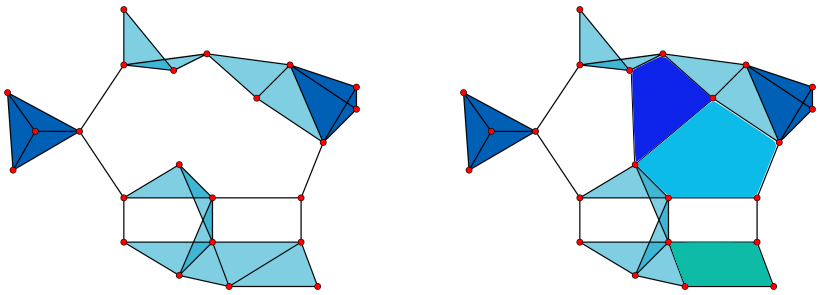


FIGURE 3.2 – Une représentation par les bords d'un objet.

Par rapport au modèle CSG qui utilise uniquement les composantes de l'objet et des opérations booléennes, les modèles B-rep sont plus flexibles et offrent plusieurs types d'opérations (chanfreinage, extrusion...). Les modèles de représentation par les bords les plus utilisés sont **les modèles cellulaires** (voir la figure 3.3). Ils autorisent la représentation d'un objet en utilisant des cellules régulières ou quelconques. Les modèles qui consistent à décomposer un objet en un ensemble de cellules régulières (plus particulièrement des triangles) dans des dimensions différentes (*i.e* sommets, arêtes, triangles, tétraèdres...) sont appelés **les modèles simpliciaux** (voir la figure 3.3(a)). Au contraire, il existe **les modèles cellulaires généraux** (voir la figure 3.3(b)) qui permettent la décomposition d'un objet en un ensemble de cellules quelconques de dimensions différentes (*i.e* sommet, arête, face, volume...).

Dans le reste de ce manuscrit nous nous intéressons uniquement aux modèles cellulaires, seuls modèles utilisés par notre approche. En effet, nous avons choisi



(a) Modèle simplicial (b) Modèle cellulaire général

FIGURE 3.3 – Modèle simplicial et modèle cellulaire.

de travailler avec un modèle cellulaire parce que l'un des objectifs de cette thèse est de créer une plate-forme générale qui permet de simuler des objets déformables représentés par différents types de maillages, potentiellement hétérogènes, c'est à dire avec différents types de n-cellules en dimension n. En utilisant des modèles simpliciaux, il est impossible de représenter directement des maillages rectangulaires en 2D, hexaédriques en 3D et, en conséquence tout maillage mélangeant les éléments, hormis en re-subdivisant chaque élément en simplexes (triangles, tétraèdres, etc.). Nous introduisons dans la suite quelques modèles cellulaires : les graphes d'incidence, puis les cartes combinatoires orientées, puis finalement, les cartes généralisées que nous avons exploitées dans notre travail.

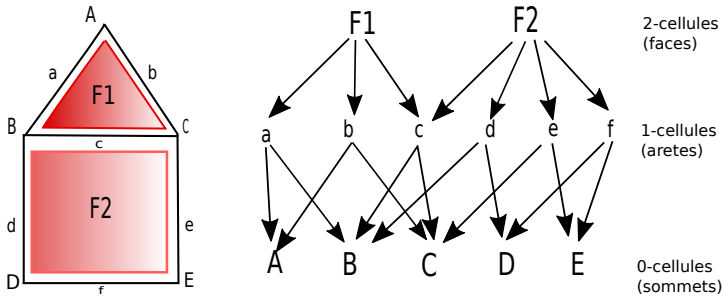
3.3 Modèles topologiques cellulaires

3.3.1 Modèles de représentation explicite de cellules

Graphes d'incidence

Le graphe d'incidence est un modèle cellulaire général qui permet de représenter des objets en tant que graphes orientés sans cycle. C'est un graphe dans lequel les nœuds correspondent aux cellules topologiques et les arcs du graphe correspondent aux relations d'incidences qui lient une cellule de dimension i à ses bords de dimension $(i-1)$.

Un exemple de graphe d'incidence est donné par la figure 3.4. Les faces $F1$ et $F2$, qui sont des 2-cellules, ont comme filles les arêtes (1-cellules) $(a, b, c, d, e$ et $f)$ qui leur sont incidentes. De même les arêtes ont comme fils les sommets (0-cellules) qui leur sont incidents. Par exemple, la face $F1$ est incidente à l'arête b (b est l'une des arêtes de la face $F1$) qui est elle-même incidente au sommet A (A est une extrémité de b). Les relations d'adjacence se déduisent par les relations d'incidence à une même cellule : par exemple, les arêtes a et b sont incidentes à un même sommet A, donc elles sont adjacentes l'une de l'autre. Les relations d'incidence entre cellules de dimensions non consécutives se retrouvent par transitivité des relations du graphe d'incidence : la face $F1$ est incidente au sommet A, par l'intermédiaire des arêtes a

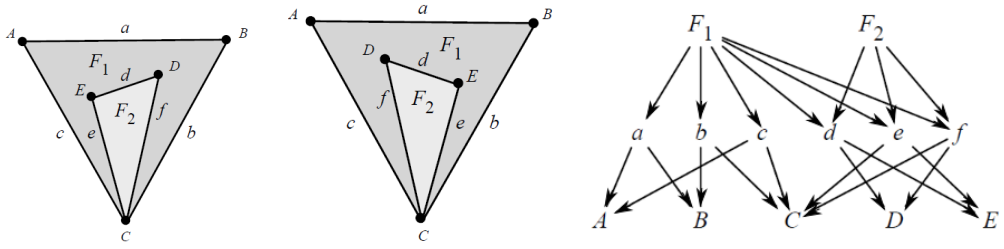


(a) Objet 2D (b) Graphe d'incidence

FIGURE 3.4 – Graphe d'incidence d'un objet 2D.

et b.

Cependant, l'exemple présenté par la figure 3.5 montre qu'un même graphe d'incidence peut représenter plusieurs objets ayant des topologies différentes parce qu'il ne présente pas d'une manière explicite l'ordre dans lequel les cellules sont placées les unes par rapport aux autres. Ainsi, sur la figure, les parcours de la face $F1$ dans le sens trigonométrique de l'objet 1 (figure 3.5(a)) puis dans l'objet 2 (figure 3.5(b)), fournissent respectivement les deux suites d'arêtes suivantes : (a, c, e, d, f, b) et (a, c, f, d, e, b). Comme on peut le remarquer ces deux suites sont différentes. On obtient également deux suites de sommets différents. Cette différence n'apparaît malheureusement pas dans le graphe de la figure 3.5(c). Le modèle n'est pas ordonné, il en résulte des ambiguïtés. Or, pour l'implantation de certains modèles mécaniques, nous avons besoin de connaître l'ordre des sommets, en particulier pour le modèle des éléments finis. Pour cette raison, dans ce document, nous avons choisi de nous placer dans le cadre des modèles ordonnés.



(a) Objet 1 (b) Objet2 (c) Graphe d'incidence de l'objet 1 et 2

FIGURE 3.5 – Graphe d'incidence ambigu.

Graphes d'incidence indicés

Les graphes d'incidences indicés sont une implantation particulière des graphes d'incidence sous une forme ordonnée. Ce modèle est très répandu : on le trouve dans des formats de fichiers graphiques comme OBJ par exemple. Il est à la base de la plupart des fichiers de description de maillages d'éléments finis comme, par

exemple, le format msh utilisé par GMsh [GR09]. Il est également utilisé par les *Vertex Arrays* proposés dans la bibliothèque graphique OpenGL. Il est, enfin, le modèle topologique de base proposé dans la bibliothèque SOFA [Sof], car c'est un modèle potentiellement très rapide. Ceci est dû à la structure en tableau qui évite l'éparpillement mémoire et une utilisation efficace des caches des processeurs.

Dans ce modèle, chaque cellule est désignée par un indice dans un tableau dédié aux cellules de sa dimension. Il y a donc autant de tableaux que de dimensions de cellules utiles à l'application. Pour chaque cellule, le tableau contient les relations d'incidence, voire d'adjacence, dont l'application a besoin, en désignant les cellules voisines par leur indice. Des informations géométriques peuvent être adjointes à chaque type de cellule. La plupart du temps, on associe au moins une position à chaque sommet.

Á titre d'exemple, la figure 3.6 illustre une structure de base où sont d'abord énumérés les sommets (lignes démarrant par « v »pour *vertex*), essentiellement caractérisés par leur coordonnées de position. L'indice des sommets correspond à leur ordre d'apparition dans le fichier (en démarrant à 1). Ces sommets sont suivis par les faces triangulaires (lignes démarrant par « f »), définies seulement par les liens d'incidence avec leurs sommets (désignés par leur indice). Notons que l'ordre des sommets est important puisqu'il définit l'orientation de la face (le modèle est ainsi effectivement ordonné).

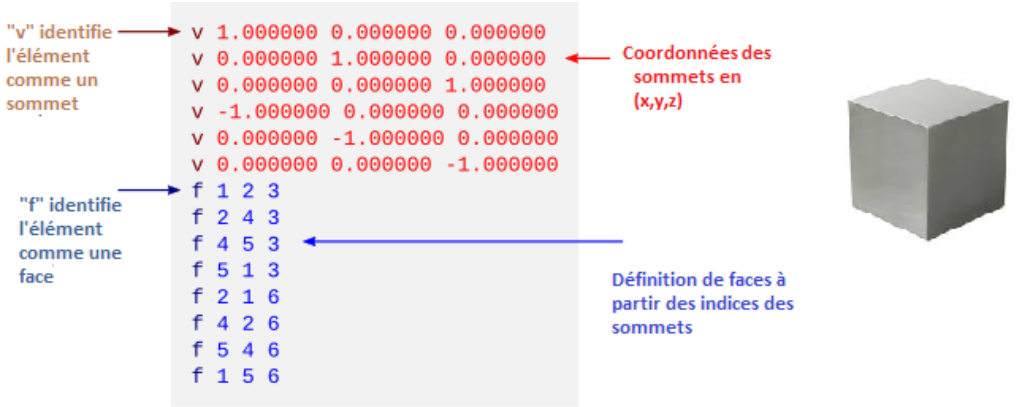


FIGURE 3.6 – Exemple d'un fichier OBJ.

Nous voyons que ce type de modèle est assez souple et peut s'adapter aux divers besoins de l'application, tant sur le plan des relations d'incidence/adjacence (on ajoute des indices supplémentaires dans les propriétés des cellules) que concernant les informations non topologiques à associer aux cellules (tout type d'information peut être ajouté aux cellules, à l'image des coordonnées des sommets). On peut le trouver sous une forme très épurée, comme dans le fichier OBJ (voir figure 3.6), ou sous une forme beaucoup plus riche.

Cette souplesse se paye néanmoins par un compromis : toute information de

voisinage non explicitement stockée dans la structure mais nécessaire à l'application doit être reconstituée. Pour cela, il faut, la plupart du temps, recourir à des algorithmes de recherche (sans tri) de complexité linaire en nombre de cellules. Par exemple, dans le cas de la structure OBJ, pour trouver tous les triangles incidents à un sommet, il est nécessaire de parcourir l'intégralité de la liste des triangles pour trouver tous ceux qui référencent l'indice du sommet. Pour éviter ces recherches, il est possible d'enrichir la structure afin de pré-stocker l'information de voisinage.

Cependant, ce stockage pose deux problèmes. D'abord, il n'est pas de taille définie *a priori*, ce qui impose le recours à des structures à allocation dynamique (type tableaux de taille non fixe ou listes). Par exemple, le nombre de triangles autour de chaque sommet dépend du sommet considéré. Le second problème concerne le maintien de la cohérence des informations après une modification de la structure. Les graphes d'incidence indicés ne proposent, en effet, aucune contrainte de cohérence, *a priori*. Par exemple, sans contrôle de cohérence spécifique, un sommet pourrait potentiellement référencer une face comme incidente, alors qu'il n'est pas listé comme sommet de cette face. Plus la structure est complexe (i.e. avec de nombreux liens d'adjacence et d'incidence additionnels), plus il faut ajouter des contrôles de cohérence dédiés. Ces contrôles visent à assurer la réciprocité des relations d'adjacence et d'incidence, mais aussi les relations obtenues par composition. Par exemple, si un sommet est incident à une arête qui est elle-même incidente à une face, alors le sommet est incident à la face. Ou, comme autre exemple, si deux faces sont incidentes à une même arête, elles sont adjacentes. Ainsi, l'exhaustivité des contraintes de cohérence est difficile à garantir.

Par ailleurs, les graphes d'incidence indicés n'offrent aucune contrainte sur le type d'objet constructible (hormis le fait d'être un assemblage de cellules). Il est alors facile d'obtenir, par exemple après une modification topologique, des assemblages non obligatoirement désirés, comme, par exemple, deux faces ou deux volumes reliés par un seul sommet. Dans [FDA05], Forest *et al.* recensent un nombre important de cas topologiquement dégénérés mais codables dans le modèle, qui nécessitent des traitements supplémentaires pour être évités.

Finalement, on peut également critiquer le fait que les graphes d'adjacence indicés ne séparent pas nettement les informations d'ordre topologique des autres informations (tout est stocké dans un même tableau). Or, en cas de modification topologique, ce sont, d'abord et avant tout, les informations topologiques qu'il faut modifier et gérer de façon cohérente. Les autres informations peuvent être traitées dans un stade ultérieur où les nouvelles relations d'incidence et d'adjacence pourront être exploitées pour les mises à jour.

Il semble ainsi assez clair qu'un modèle plus rigoureux autant sur le plan des objets modélisables que de la cohérence des relations de voisinages est nécessaire sitôt que les modifications topologiques du modèle sont fréquentes. La suite de ce chapitre présente, à ce titre, des modèles beaucoup plus solides sur le plan topologique, évitant les écueils que nous venons d'exposer : les diverses familles de cartes combinatoires.

3.3.2 Modèles de représentation implicite de cellules

Les principaux inconvénients des graphes d'incidence exposés dans la section précédente proviennent essentiellement de la représentation qui est basée sur les cellules. En utilisant des éléments plus abstraits, il est possible de mieux définir et manipuler les relations topologiques. Les cellules sont alors représentées de façon implicite, à partir d'éléments abstraits. Des modèles de ce type ont d'abord été proposés en 2D avant d'être généralisés.

3.3.2.1 Modèles 2D

Les modèles topologiques 2D sont apparus dans les années 70. Parmi ces structures nous pouvons citer les modèles qui sont basés sur les arêtes. Ces derniers permettent et facilitent la recherche des informations d'adjacence entre les faces, les arêtes et les sommets. Parmi ces modèles, se trouve le modèle des arêtes ailées (*winged-edges*) [Bau75] et des demi-arêtes (*half-edges*) [Man88].

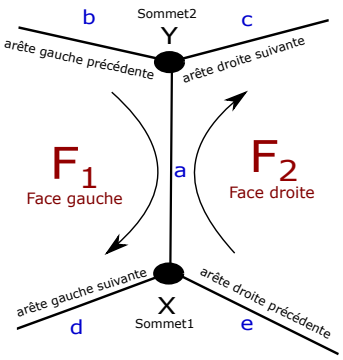


FIGURE 3.7 – Représentation d'un objet 2D avec la structure des arêtes ailées.

Le modèle des arêtes ailées est un modèle cellulaire orienté. Il se base sur un seul type de cellule, les arêtes, qui sont à ce titre orientées. Chaque arête regroupe huit informations détaillées sur la figure 3.7 :

- d'une part, les informations non topologiques (souvent géométriques) concernant les sommets de l'arête, X et Y (par exemple leur position), et concernant les deux faces incidentes à l'arête considérée, F_1 et F_2 (par exemple leur normale et/ou leur couleur).
- d'autre part, les informations topologiques, qui se résument en réalité à référencer l'arête précédente et l'arête suivante sur chacune des faces incidentes, F_1 et F_2.

Comme on peut le remarquer sur la figure, les arêtes b, c, d et e sont les « ailes » de l'arête a. L'inconvénient de cette structure est que, pour une seule arête, il existe deux parcours (un pour chaque face) selon que l'on utilise ou pas le sens de l'orientation choisi pour l'arête.

La structure des demi-arêtes est apparue pour améliorer la structure des arêtes ailées. Chaque arête est décomposée en deux demi-arêtes orientées comme montré sur la figure 3.8. Alors qu'une arête ailée est liée à deux faces, une demi-arête n'est liée qu'à une seule face. Les deux demi-arêtes font référence l'une à l'autre, ce qui reconstitue l'arête et permet de représenter directement l'adjacence de deux faces. Chaque demi-arête recense également la demi-arête suivante (en noir) et la précédente (en bleu) sur la face. De la même manière que pour les arêtes ailées, il est possible d'attacher à la demi-arête les informations non topologiques concernant son sommet de départ (dans notre exemple le sommet X) ainsi que celles concernant sa face (ici F_1).

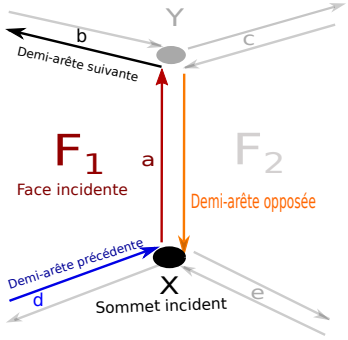


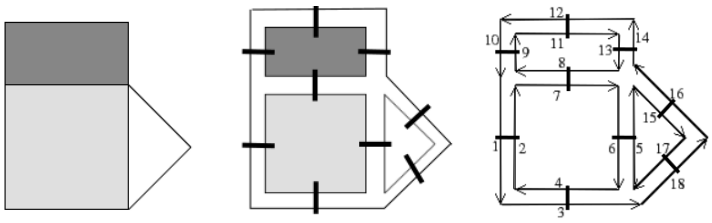
FIGURE 3.8 – Représentation d'un objet 2D avec la structure des demi-arêtes.

Les cellules topologiques de cette structure (faces, arêtes et sommets) sont représentées implicitement. En effet, une arête apparaît comme l'union de deux demi-arêtes. Une face apparaît comme un ensemble de demi-arêtes successives. Ces demi-arêtes référencent les mêmes informations de face. Un sommet regroupe l'ensemble des demi-arêtes qui partent de ce sommet. Pour le sommet X, cet ensemble contient la demi-arête a. Reconstruire les cellules nécessitent ainsi un parcours de la structure.

Les demi-arêtes permettent de représenter uniquement des objets 2D de type variété, avec ou sans bord selon que l'on oblige ou pas chaque demi-arête à s'associer à une autre demi-arête pour former une arête. Cette structure a pu être formalisée et généralisée aux dimensions supérieures à 2 par les cartes combinatoire orientées que nous présentons dans la section suivante.

3.3.2.2 Cartes combinatoires orientées

Il est possible d'étendre la formulation des demi-arêtes 2D aux dimensions supérieures grâce au formalisme des *cartes combinatoires orientées* (ou plus simplement « cartes orientées ») [BS85, DL14]. Les cartes orientées permettent de représenter les différentes subdivisions de l'espace et les relations d'incidence entre les cellules. Elles sont définies mathématiquement en dimension quelconque.



(a) Objet géométrique (b) Décomposition en faces (c) Carte orientée

FIGURE 3.9 – Décompositions successives d'un objet géométrique 2D pour trouver la carte orientée correspondante.

Une carte orientée peut être définie par décompositions successives des différentes cellules d'un objet, comme on peut le voir sur la figure 3.9(a), pour un objet en dimension 2. Tout d'abord, cet objet est décomposé en faces liées le long d'une arête. Sur la figure 3.9(b), la relation d'adjacence entre deux faces est représentée par un trait noir que l'on peut appeler β_2. De même, les faces sont subdivisées en arêtes. Ainsi, chaque arête est constituée de deux demi-arêtes qui sont appelées *brins* de la carte orientée présentée sur la figure 3.9(c). Chaque brin indique quel est son successeur pour la face dans laquelle il se trouve, par une liaisons que l'on peut appeler β_1. Chaque brin est ainsi lié à deux autres brins et la liaison se représente sous la forme d'une flèche.

Cet exemple représente un objet 2D, mais avec les cartes orientées, des objets de toute dimension peuvent être représentés. Les brins sont liés à autant de brins que l'exige la dimension, ce qui implique d'ajouter des liaisons β_n. Par exemple en 3D, il faut définir la liaison β_3 pour lier deux volumes.

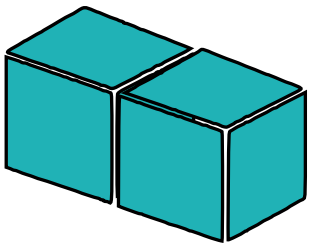
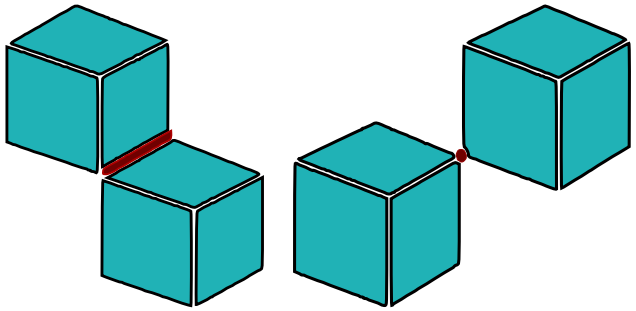


FIGURE 3.10 – Exemple de quasi-variétés.

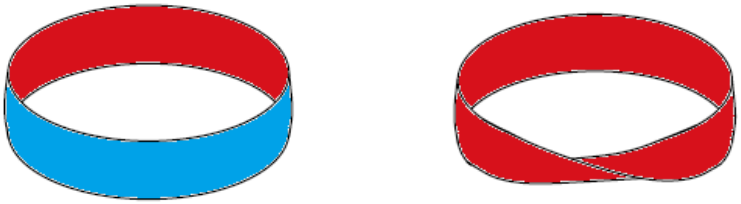
Les cartes orientées permettent de représenter des objets de type quasi-variétés et orientables. Une quasi-variété est un ensemble de cellules de dimension n qui sont liées le long d'une cellule de dimension $(n-1)$. Par exemple, sur la figure 3.10, les deux cubes (cellules de dimension 3) sont liés par une face (cellule de dimension 2).

Quelques exemples d'objets non quasi-variétés sont présentés sur la figure 3.11. Deux cubes (3-cellules) liés le long d'une arête (1-cellule) (voir la figure 3.11(a)) ou par un sommet (0-cellule) (voir la figure 3.11(b)) ne sont pas des quasi-variétés et ne sont donc pas représentables avec des cartes orientées.



(a) (b)

FIGURE 3.11 – Exemples de non-variétés.



(a) Anneau : objet orientable

(b) Ruban de Moebius : objet non orientable

FIGURE 3.12 – Exemple d'objets orientable et non orientable.

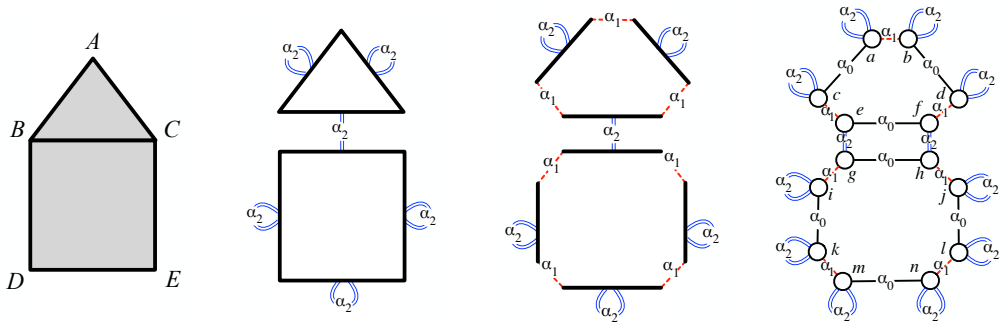
Une surface est dite orientable, si elle admet deux faces : intérieure et extérieure (si la surface est fermée) ou recto et verso (sinon). Par exemple, l'anneau de la figure 3.12(a) est une surface orientable, car il a deux faces (respectivement rouge et bleue sur la figure). Au contraire, le ruban de Moebius de la figure 3.12(b) est non orientable, car il n'admet qu'une seule face. Cette notion d'orientation est extensible en toute dimension. Notons que certaines implantations du modèle des cartes orientées peuvent être restreintes aux objets fermés, car ce choix est plus efficace sur le plan algorithmique.

Comme pour les modèles 2D précédemment présentés, les cartes orientées ne représentent pas explicitement les cellules. Ces dernières apparaissent comme un ensemble de brins et leurs liaisons. Pour définir la géométrie et fournir aux cellules des données applicatives, il est donc, là aussi, possible d'ajouter des informations non topologiques, en les référençant par leurs brins. Ces informations sont appelées *plongements*. Cependant, contrairement aux premiers modèles topologiques proposés qui cantonnent les informations aux seules cellules, il est possible d'ajouter des plongements à des ensembles de brins ne correspondant pas obligatoirement à des cellules. Nous présentons et discutons des contraintes de ce mécanisme dans la section suivante.

3.3.2.3 Cartes combinatoires généralisées

Les *cartes combinatoires généralisées* ou plus simplement cartes généralisées (ou *G-cartes*) [Lie89, Lie91] permettent de représenter des quasi-variétés, avec ou sans bord, orientables ou non. Ce modèle se distingue ainsi des cartes orientées par sa capacité à représenter des objets non orientables, mais également par le fait que sa définition est homogène en toute dimension, car tous les liens qu'il utilise sont non orientés.

Initialement, les G-cartes sont définies algébriquement [Lie89]. Dans les travaux de Mathieu Poudret [Pou09] et Thomas Bellet [Bel12], les G-cartes ont été redéfinies en tant que graphes afin qu'elles puissent être manipulées par des transformations de graphes. Dans ce manuscrit, nous privilégions cette définition car l'outil que nous utilisons se base sur ce formalisme.



(a) Objet géométrique (b) Décomposition en faces (c) Décomposition en arêtes (d) G-carte

FIGURE 3.13 – Décompositions successives d'un objet géométrique 2D.

Comme pour les cartes orientées, la représentation d'un objet par une G-carte consiste à le décomposer successivement en cellules topologiques (volumes, faces, arêtes, sommets...) de dimension décroissante. Un exemple de décomposition d'un objet 2D est donné sur la figure 3.13. Dans cet exemple, l'objet initial (figure 3.13(a)) est décomposé en deux faces (figure 3.13(b)) qui sont liées le long d'une arête commune par des liens dits « de dimension 2 » nommés α_2, car ils lient des cellules de dimension 2. Ensuite, ces deux faces sont décomposées en arêtes (figure 3.13(c)) qui sont liées par des liens dits « de dimension 1 » nommés α_1. Finalement, chaque arête est décomposée en deux sommets (figure 3.13(d)), qui sont liés par des liens de dimension 0 nommées α_0. Ces « sommets », plus précisément ces sommets issus d'une arête elle-même issue d'une face (qui est éventuellement issue d'un volume, etc.) sont appelée *brins*. Ils constituent les nœuds du graphe de la carte généralisée et les liaisons constituent les arcs du graphe.

Avant de continuer cette partie, tout au long de ce manuscrit, pour des raisons de simplification de lecture des schémas, nous utilisons une convention graphique introduite sur la figure 3.14. Les liaisons α_i sont représentées par des lignes noires pour α_0, un trait rouge pointillé pour α_1, et un double trait bleu pour α_2.

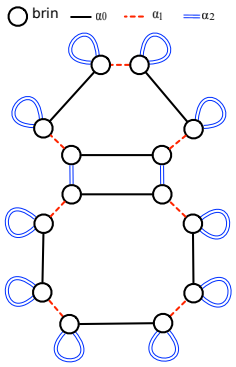
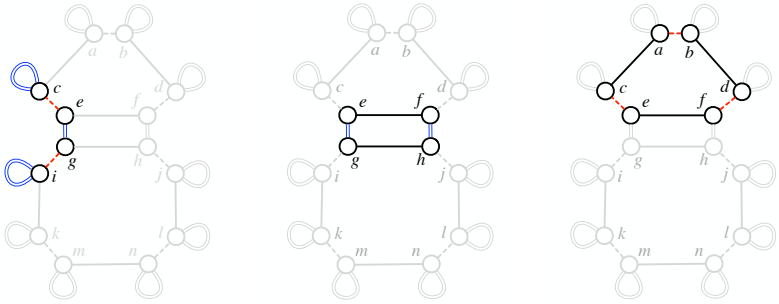
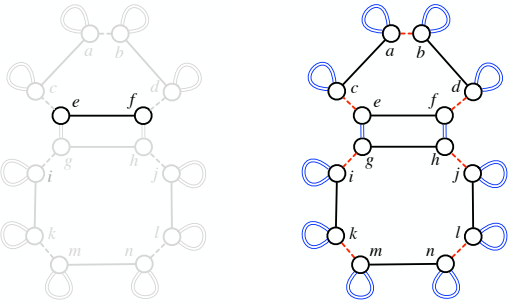


FIGURE 3.14 – Conventions graphiques.



(a) Sommet (b) Arête (c) Face



(d) Demi-arête (e) Composante connexe

FIGURE 3.15 – Exemples des orbites adjacentes au brin e.

A l'instar des cartes orientées, les cellules topologiques des objets (volumes, faces, arêtes, sommets...) ne sont pas représentées explicitement dans les G-cartes. Elles sont définies à partir de sous-graphes appelés *orbites*. Par exemple, le sommet B (figure 3.13(a)) est défini sur la figure 3.15(a), par le sous-graphe comprenant le brin e et l'ensemble des brins atteignables par les liens de dimensions 1 et 2 (*i.e.* les brins c, e, g, et i) et l'ensemble des liaisons correspondantes. Cette orbite est notée $\langle \alpha_1, \alpha_2 \rangle(e)$. L'arête BC du même objet (figure 3.13(a)) est définie par le sous-graphe

comprenant le brin e et l'ensemble des brins atteignables par les liens de dimensions 0 et 2 (*i.e.* les brins e, f, g, et h) et les liaisons correspondantes. Cette orbite se note $\langle\alpha_0, \alpha_2\rangle(e)$ (voir figure 3.15(b)). La face ABC est définie par l'orbite $\langle\alpha_0, \alpha_1\rangle(e)$ qui est le sous-graphe qui contient le brin e et l'ensemble des brins atteignables par les liens de dimensions 0 et 1 (*i.e.* les brins a, b, c, d, e et f sur la figure 3.15(c)) et les liaisons elles-mêmes.

Il faut préciser que les cellules ne sont que des cas particuliers d'orbites. Il existe, en effet, d'autres orbites. Par exemple, la demi-arête, que l'on appellera désormais en 2D *arête de face* par souci de précision et pour lever toute ambiguïté, contient les brins e et f sur la figure 3.15(d) ce qui correspond à l'orbite $\langle\alpha_0\rangle(e)$. La composante connexe, c'est à dire, dans ce cas, tout l'objet (figure 3.15(e)), est désignée par l'orbite $\langle\alpha_0, \alpha_1, \alpha_2\rangle(e)$. Lorsque l'orbite ne fait mention d'aucun α_i, alors il s'agit de l'orbite brin comme $\langle\rangle(e)$, qui ne comprend que le brin e lui-même. Nous définissons ainsi la notion d'orbite : pour un sous-ensemble orb de $\{\alpha_0, ..., \alpha_n\}$ et b un brin d'une G-carte G de dimension n, l'**orbite** $\langle orb\rangle(b)$ est définie par le sous-graphe de G qui contient b, tous les brins de G atteignables par une suite de liaisons étiquetés par les éléments de orb, ainsi que ces liaisons.

Une carte généralisée de dimension n peut ainsi être définie comme un graphe [1] G dont les arcs sont étiquetés par α_i où i appartient à $[0..n]$ et qui satisfont les contraintes de cohérence topologique suivantes :

- non orientation : G est non orienté
- arcs adjacents : chaque brin est la source exactement de $n+1$ liaisons respectivement étiquetées de α_0 à α_n
- cycles : pour tous indices i et j tels que $0 \leq i \leq i+2 \leq j$, il existe un cycle [2] étiqueté par $\alpha_i\alpha_j\alpha_i\alpha_j$ dans G à partir de chacun de ses brins.

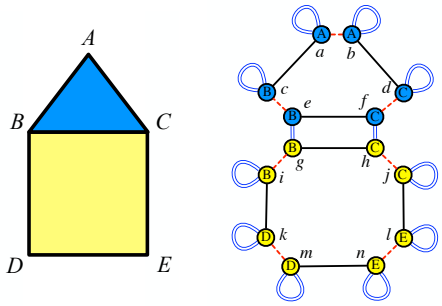
Cette définition et plus précisément la troisième contrainte permet d'assurer que l'objet est une quasi-variété. De plus, ces contraintes restent valides pour le bord de l'objet en utilisant les liaisons qui bouclent sur les brins du bord.

Comme pour les cartes orientées, des plongements (informations géométriques ou autres selon l'application visée) peuvent être ajoutés à la structure topologique. Pour contrôler la cohérence des plongements, on les associe obligatoirement à une orbite, que cet orbite représente une cellule ou pas. Ces plongements sont représentés par des étiquettes placées sur les brins de la G-carte comme défini dans [Bel12].

1. Soient n une dimension topologique et Π une famille d'informations π de type τ. Un graphe $G = (V, E, s, t, \alpha, \Pi)$ de dimension n et d'information Π est défini par un ensemble de brins V (les nœuds) et des liaisons E (les arcs), les fonctions source $s : E \to V$, cible $t : E \to V$ et d'étiquetage topologique $\alpha : E \to [0..n]$ tel que pour toute liaison $e \in E$, $s(e)$ est le brin source de e et $t(e)$ son brin cible et $\alpha(e)$ est sa dimension topologique, et des fonctions d'information $\pi : v \to \tau$ pour tout π de Π tel que pour tout brin de V, $\pi(v)$ est l'information π du brin v.

2. Un chemin est une suite de liaisons $e_1...e_k$ tel que pour tout $1 < i < k-1$ $t(e_i) = s(e_{i+1})$. Le chemin admet une source $s(e_1)$, une cible $s(e_k)$ et une étiquette topologique $\alpha(e_1)\alpha(e_1)...\alpha(e_k)$. Le chemin est un cycle si sa source et sa cible sont identiques $s(e_1) = t(e_k)$.

Pour qu'un plongement soit cohérent, une contraintes doit être satisfaite : il faut assurer que, pour tout plongement π porté par une orbite $\langle o \rangle$, tous les brins de cette orbite portent la même information. Un plongement se définit formellement sous la forme : $\pi : \langle orb \rangle \rightarrow \tau$, où π est le nom du plongement, $\langle orb \rangle$ est le type d'orbite auquel il est associé, et τ est le type de l'information. On note Π la famille des plongements π. Une carte généralisée est dite *plongée* lorsqu'on lui associe un ou plusieurs plongements π.



(a) (b)

FIGURE 3.16 – Représentation d'un objet 2D avec des plongements.

Quelques exemples de plongement sont donnés sur la figure 3.16. Le plongement géométrique `point` : $\langle \alpha_1, \alpha 2 \rangle \rightarrow Point$ définit la position de chaque sommet (A, B, C, D et E) dans un espace géométrique de dimension 2D/3D et est rattaché aux orbites $\langle \alpha_1, \alpha_2 \rangle$ correspondant aux sommets pour un objet 2D. Pour satisfaire la contrainte de cohérence de plongement, tous les brins du même sommet partagent la même position. Par exemple, sur la figure 3.16(b), les brins c, e, g et i partagent la même position B de type $Point$. De même, le plongement `color` : $\langle \alpha_0, \alpha_1 \rangle \rightarrow RGB$ permet de définir la couleur des faces, sous la forme d'un triplet $(rouge, vert, bleu)$ en l'associant aux orbites faces $\langle \alpha_0, \alpha_1 \rangle$. Tous les brins de la même face partagent la même couleur. Par exemple, sur la figure 3.16(b) les brins a, b, c, d et f appartiennent à la même face et partagent la même couleur, dans le cas présent le bleu.

Toute information que l'on souhaite attacher à une G-carte se traduit par un plongement, associé à un type d'orbites. Il convient alors de définir les orbites les plus adéquates pour porter l'information. Il peut s'agir des cellules ou d'orbites plus spécifiques comme, en 2D, les arêtes de face $\langle \alpha_0 \rangle$, les sommets de face $\langle \alpha_1 \rangle$, les extrémités d'arête $\langle \alpha_2 \rangle$, les composantes connexes $\langle \alpha_0, \alpha_1, \alpha_2 \rangle$, etc. En 3D, les définitions des orbites changent et de nouvelles orbites apparaissent : par exemple, les arêtes de faces correspondent aux orbites $\langle \alpha_0, \alpha_3 \rangle$ et les orbites $\langle \alpha_0 \rangle$ sont des arêtes de face de volume (*i.e.* les arêtes d'une seule face dans un seul volume). Néanmoins, certains éléments ne correspondent pas forcément à un type d'orbites. C'est le cas par exemple des diagonales de face. Dans ce cas, puisqu'on ne peut pas associer une information aux diagonales, il faut l'associer aux orbites qui englobent

ou qui permettent de retrouver les diagonales en question comme par exemple les orbites faces $\langle \alpha_0, \alpha_1 \rangle$.

Le choix d'un modèle topologique dépend du type d'objets à représenter dans l'application visée et de la nature des opérations à réaliser. On ne peut pas dire qu'il existe un modèle meilleur qu'un autre. Néanmoins, on peut dire qu'un modèle est plus adapté pour une classe d'objets ou bien pour une utilisation dans une application particulière. Nous avons choisi de travailler avec un modeleur cellulaire parce que, comme déjà mentionné dans la section précédente, nous voulons créer un cadre général pour simuler différents types d'objets avec différents types de maillages, y compris des maillages hétérogènes. La famille des cartes combinatoires est bien adaptée. Parmi les cartes combinatoires, nous avons choisi les G-cartes, tout d'abord, parce que c'est un modèle général qui permet de représenter une large classe d'objets et qu'il est défini mathématiquement avec des contraintes de cohérence. Mais ce choix est principalement motivé par l'ensemble des orbites utilisables, car elles sont plus atomiques avec les cartes généralisées qu'avec les cartes orientées. Par exemple, dans une 2-G-carte, on peut distinguer une arête de face (orbite de type $\langle \alpha_0 \rangle$), d'une extrémité d'arête (orbite de type $\langle \alpha_2 \rangle$). Au contraire, dans une carte orientée, l'orbite brin désigne à la fois les arêtes de faces et les extrémités d'arêtes. Il n'est donc pas possible de différencier les plongements associés aux arêtes des faces de ceux associés aux extrémités d'arêtes. De même, les opérations qui s'appliquent aux arêtes de faces sont indistinguables de celles qui s'appliquent aux extrémités d'arêtes. Surtout, il n'existe pas d'orbite correspondant aux extrémités des arêtes de faces. Donc pour lui associer une information, il convient d'utiliser deux plongements, le premier qui porte l'information de l'origine de l'arête de face, et l'autre qui porte celle de l'autre extrémité. Bien entendu, dans ce cas la cohérence des plongements ne peut plus être assurée formellement. Les G-cartes nous offrent ainsi la gamme d'orbites la plus vaste, d'où notre choix, mais le passage au cartes orientées pourra se faire, au prix de compromis et/ou de conventions. Finalement, une autre raison pour laquelle nous utilisons les G-cartes est l'utilisation d'un langage à base des règles qui est basé sur les G-cartes. L'homogénéité de ce modèle simplifie la définition des opérations avec les règles. Dans la partie suivante, nous présentons ce langage et la manière de l'utiliser pour créer des opérations de manipulation des objets.

3.4 Modélisation à base de règles : Jerboa

Les langages à base de règles ont été utilisés pour la modélisation géométrique des objets et l'animation. Parmi les langages les plus connus se trouvent les *L-systèmes* [PL90]. Ils ont été introduits pour modéliser le développement et la croissance des plantes. Un exemple de modélisation d'une plante est donné sur la figure 3.17. Les langages utilisés en informatique graphique ne sont pas limités à la modélisation de plante. Ils ont été utilisés dans plusieurs autres domaines, comme

FIGURE 3.17 – Modélisation d'une plante avec un L-system.

par exemple, la reconstruction de bâtiments [VAB10]. Ils ont également été utilisés dans un contexte topologique dans [TGM+09] pour modéliser la structure interne du bois, la croissance des feuilles et pour modéliser la structure interne des fruits [BTG15]... Quelle que soit l'application exploitant les L-systèmes, un ensemble limité d'opérations spécialisées est défini et codé de manière *ad hoc*. Ces dernières sont appliquées au moyen de certains types de règles et grammaires. Ainsi, chaque domaine d'application utilise un dialecte de L-système différent.

Au contraire, Jerboa définit les opérations topologiques et géométriques sous la forme de transformations de graphes qui modifient directement la structure topologique des objets et leurs plongements. Il est générique en dimension et en plongement et reste indépendant du domaine d'application. Comme un de nos objectifs est de proposer une plate-forme générale pour la simulation physique, l'utilisation d'un langage à base de règles paraît intéressant pour un tel travail exploratoire. En effet, ces langages permettent le prototypage rapide et facile des opérations. Outre sa généricité, le principal avantage de Jerboa est la vérification syntaxique des règles lors de leur création. Cette vérification permet de garantir la préservation de contraintes de cohérence topologique et de plongement des objets lors de leur transformation (*i.e.* lors de l'application des règles). Ainsi, des objets construits et modifiés uniquement par des règles sont garantis cohérents.

3.4.1 Le langage Jerboa

Une règle Jerboa se présente sous la forme suivante : $L \rightarrow R$. L est le membre gauche de la règle et correspond au motif à filtrer (c'est à dire le sous-graphe à reconnaître dans l'objet). R est le membre droit, il correspond au motif transformé (c'est-à-dire le sous-graphe qui remplace L dans l'objet). Les règles de transformation de graphe sont définies par des graphes qui décrivent l'objet avant et après une modification. Un exemple de règle est donné sur la figure 3.18. Cette règle permet d'insérer un sommet sur une arête du bord (car comme on peut le remarquer sur la règle, les liaisons α_2 sont libres).

L'application de la règle présentée dans la partie haute de la figure 3.18 se fait en associant les nœuds de partie gauche (L) de la règle sur les nœuds du graphe

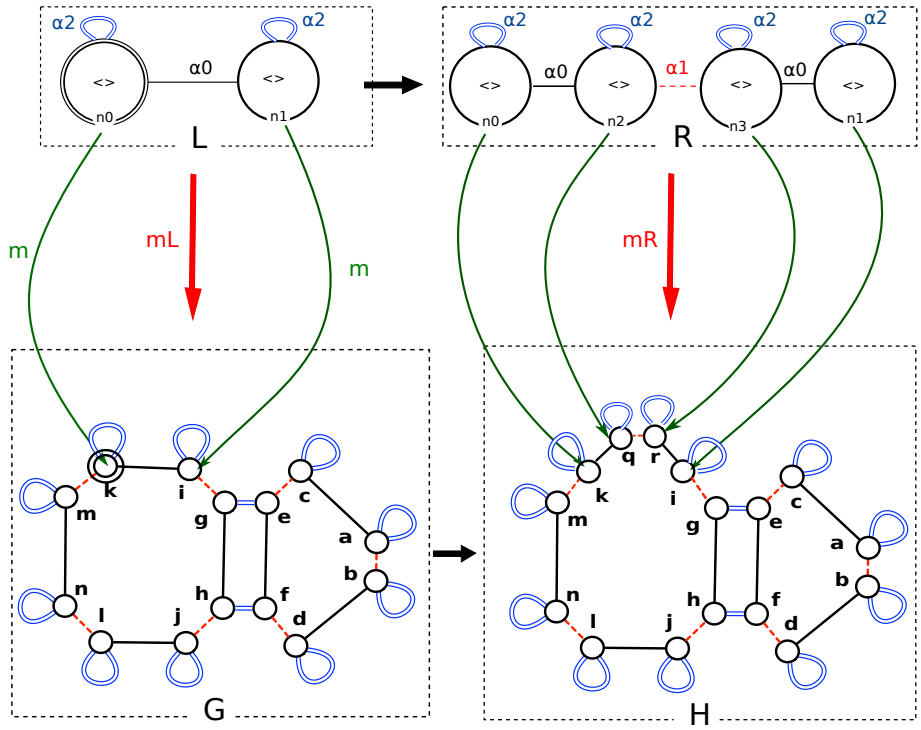


FIGURE 3.18 – Exemple d'application d'une règle de transformation de graphe.

G. Cette association se fait à l'aide d'un morphisme de filtrage [3]. Par exemple ici, dans le membre gauche, le nœud `n0` est associé au brin k du graphe G et le nœud `n1` au brin i. Dans le cas où il existe un morphisme de filtrage, la règle s'applique en remplacent alors le motif filtré dans L par le motif décrit dans le membre droit qui sert ici à insérer un sommet sur une arête du bord, et pour cela il ajoute deux nouveaux nœuds (`n2` et `n3`) qui sont associés respectivement aux brins q et r du graphe H (graphe transformé) par le morphisme de filtrage mR. S'il n'existe pas de morphisme de filtrage mL alors le motif L n'est pas reconnu dans le graphe G (le filtrage échoue) la règle ne s'applique pas et ainsi aucune transformation n'est appliquée.

Il faut noter que pour appliquer une règle, il convient à préciser son lieu d'application en construisant le morphisme de filtrage. Pour cela, dans le membre de gauche de la règle, on marque par un double cercle les nœuds à fournir. Ces nœuds s'appellent **les nœuds d'accroche** (*hook*) et sont les paramètres topologiques de la règle. Lors de l'application d'une règle Jerboa, chaque nœud d'accroche est associé à un brin de l'objet. Si chaque composante connexe du graphe gauche de la

3. Un morphisme $m : L \to G$ est une fonction qui transporte les brins et les liaisons en préservant la structure du graphe et son étiquetage. Pour toute liaison e de L, $s(m(e)) = m(s(e))$, $t(m(e)) = m(t(e))$ et $\alpha(e) = \alpha(m(e))$ et pour tout brin v de L et tout plongement π, $\pi(e) = \pi(m(e))$. m est un morphisme de filtrage s'il est injectif. C'est à dire, pour toutes liaisons e et $e\prime$ de L, $m(e) = m(e\prime)$ si seulement si $e = e\prime$ et pour tout brin v et $v\prime$ de L, $m(v) = m(v\prime)$ si seulement si $v = v\prime$.

règle possède un nœud d'accroche, alors le morphisme (quand il existe) est unique. Et la règle peut ainsi être appliquée sans ambiguïté. Sur la figure 3.18, le nœud d'accroche `n0` est associé au brin k, et le morphisme de filtrage est représenté par les flèches vertes. Notons que si l'on essaye d'appliquer cette même règle sur l'arête centrale, par exemple en associant le nœud `n0` au brin g, alors le filtrage échouera, car le brin g n'a pas de boucle α_2 contrairement au nœud `n0`.

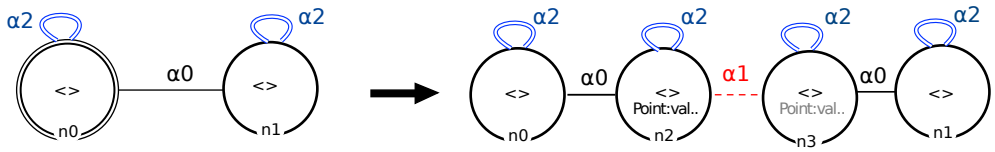


FIGURE 3.19 – Ajout de plongement pour calculer la position du sommet à insérer.

En complément de la transformation topologique, les règles permettent de transformer les plongements. Par exemple, dans la figure 3.19, la position du nouveau sommet est définie par le plongement `point` par une expression.

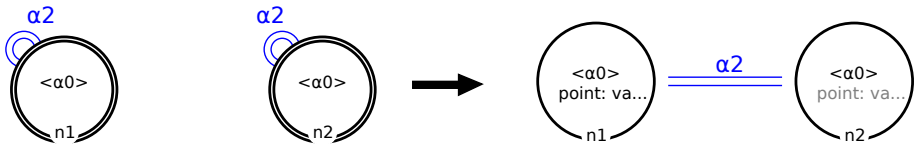


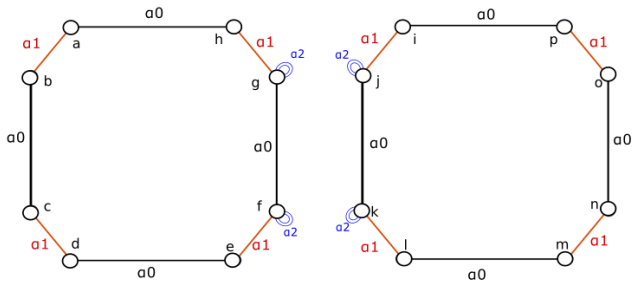
FIGURE 3.20 – Règle de couture de deux faces le long d'une arête.

Comme indiqué précédemment, il existe des règles qui contiennent plus qu'un nœud d'accroche dans le membre gauche comme par exemple dans la règle de couture présentée sur la figure 3.20 qui s'applique sur deux arêtes et permet de coudre deux faces le long d'une arête. Aussi, la règle contient deux nœuds d'accroche, car les deux arêtes sont indépendantes. D'autre part, chaque nœud est ici étiqueté par les orbites $\langle\alpha_0\rangle$, chaque nœud filtre donc une orbite $\langle\alpha_0\rangle$ complète. Ainsi le nœud `n1` filtre les brins g et f, et `n2` filtre les brins j et k.

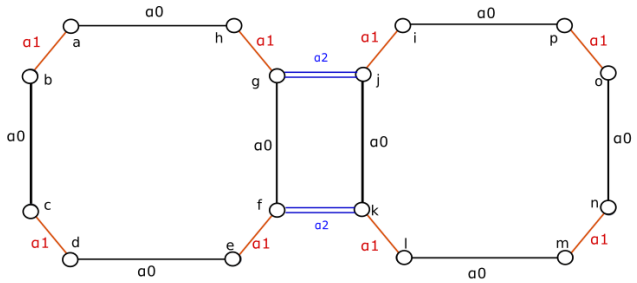
Nous détaillons dans la suite les principes utilisables dans Jerboa pour définir une règle, d'abord dédiée aux changements topologiques, et par la suite, dédiée aux changements de plongement. Nous donnons une brève description de l'outil Jerboa, de son éditeur et quelques exemples de règles.

3.4.1.1 Transformations topologiques

Pour définir des opérations pour différents types et formes d'orbites, le langage à base de règles est doté de *variables topologiques* [PACL08]. Pour cela, les nœuds sont étiquetés par des orbites, ce qui permet de désigner, en un coup, tous les brins de l'orbite en question, lors de l'instanciation de la règle.



(a) Avant l'application de la règle



(b) Après l'application de la règle

FIGURE 3.21 – Exemple d'application de la règle de couture de deux faces.

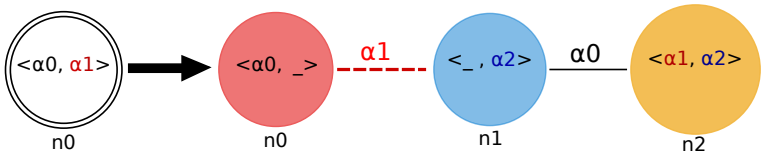
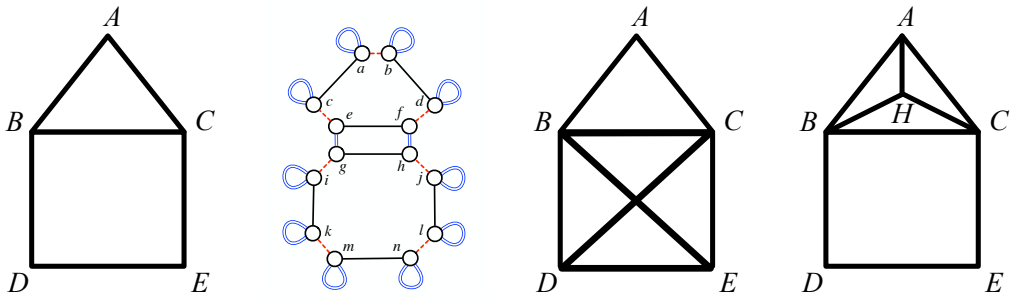


FIGURE 3.22 – Règle de triangulation d'une face.

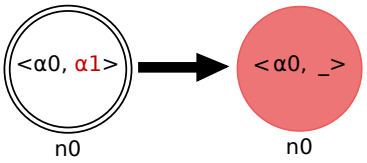
Par exemple, la triangulation d'une face peut être définie par la règle présentée sur la figure 3.22. Dans le membre gauche, un seul nœud nommé `n0` définit toute la face à filtrer. Il est étiqueté avec l'orbite $\langle \alpha_0, \alpha_1 \rangle$, qui définit les faces en 2D.

L'application de cette règle sur le brin g de la G-carte présentée sur la figure 3.23(b) instancie la règle pour une face quadrangulaire et fournit le résultat de la figure 3.23(c). De même, son application sur le brin a instancie la règle pour une face triangulaire (voir le résultat sur la figure 3.23(d)). Ainsi la règle est instanciée avant d'être appliquée. Pour cela, sa variable topologique `n0` (le nœud d'accroche) est substituée par l'orbite d'application, $\langle \alpha_0, \alpha_1 \rangle(g)$ ou $\langle \alpha_0, \alpha_1 \rangle(a)$ dans notre exemple. Les autres nœuds de la règle sont instanciés par la même orbite, mais à renommage et/ou suppression des liaisons près.

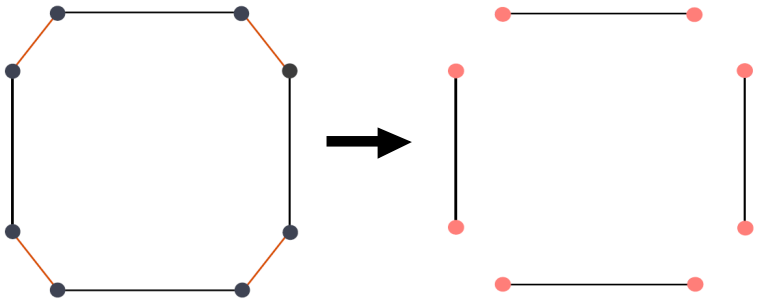


(a) Objet (b) G-carte (c) Résultat sur un rectangle (d) Résultat sur un triangle

FIGURE 3.23 – Application de la triangulation d'une face.



(a) Règle

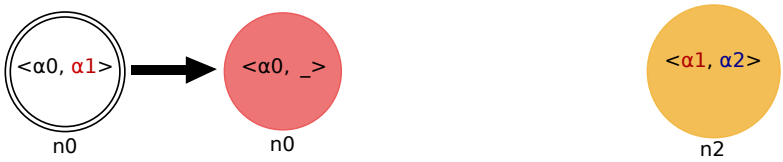


(b) Instanciation

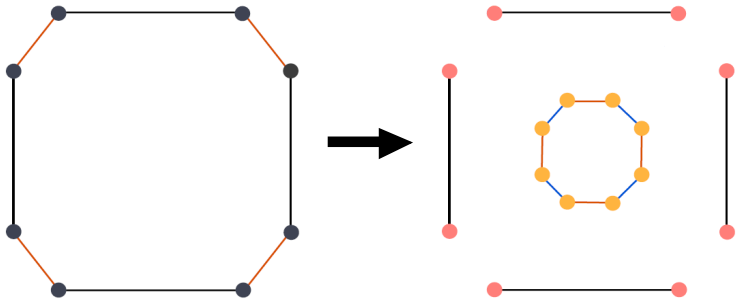
FIGURE 3.24 – Instanciation du nœud `n0` de la règle de triangulation d'une face.

Commençons tout d'abord par le nœud `n0` (en magenta) du membre droit de la règle. Il est étiqueté par $\langle \alpha_0, - \rangle$, ce qui signifie que les liaisons α_0 de la face filtrée sont préservées alors que les liaisons α_1 sont supprimées (ce qui est noté par le caractère « _ »). La sous-règle correspondante (figure 3.24(a)) a donc pour instanciation la règle de transformation de la figure 3.24(b). Son application a ainsi pour effet de découdre les arêtes de la face filtrée.

De la même façon, les liaisons peuvent être renommées. En se référant à la même règle de triangulation, le nœud `n2` (nœud jaune) est étiqueté par l'orbite $\langle \alpha_1, \alpha_2 \rangle$. Cela indique que les liaisons α_0 (resp. α_1) de l'orbite d'instanciation sont renommées en α_1 (resp. α_2). Ce qui permet de construire le sommet dual (constitué de brins jaunes) de la face filtrée au départ comme montré par la figure 3.25. Remarquons que lorsqu'un même nœud (comme `n0`) apparaît à gauche et à droite de la règle,



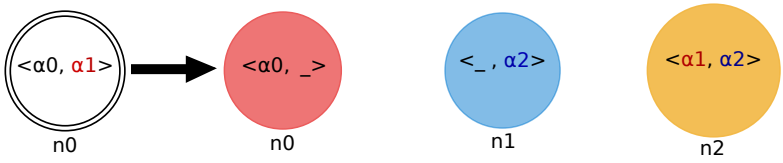
(a) Règle



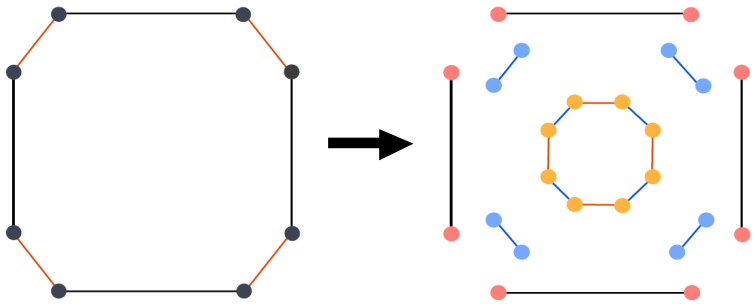
(b) Instanciation

FIGURE 3.25 – Instanciation du nœud `n2` de la règle de triangulation d'une face.

les brins filtrés sont préservés lors de la transformation. Quand un nœud n'apparaît qu'à droite de la règle (comme `n2`), alors la transformation ajoute de nouveaux nœuds. Et au contraire lorsque un nœud n'apparaît qu'à gauche de la règle, la transformation supprime des brins.



(a) Règle

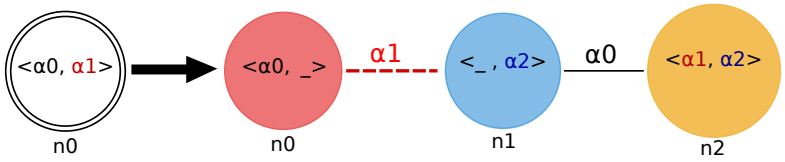


(b) Instanciation

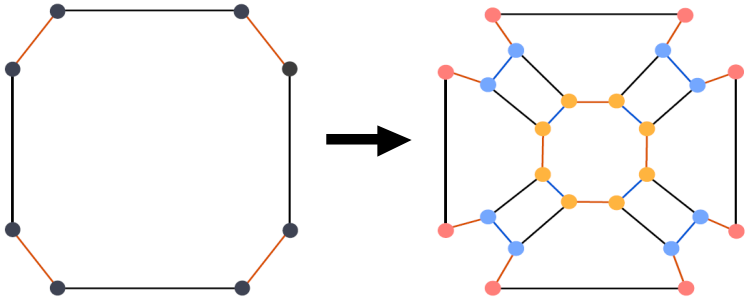
FIGURE 3.26 – Instanciation du nœud `n1` de la règle de triangulation d'une face.

Un nœud peut présenter à la fois des liaisons supprimés et d'autre renommées. C'est le cas pour le nœud `n1` (nœud bleu) qui est étiqueté par $\langle -, \alpha_2 \rangle$, les liaisons α_0 de l'orbite filtrée sont donc supprimées et les liaisons α_1 sont renommées en α_2

comme montré par la figure 3.26.



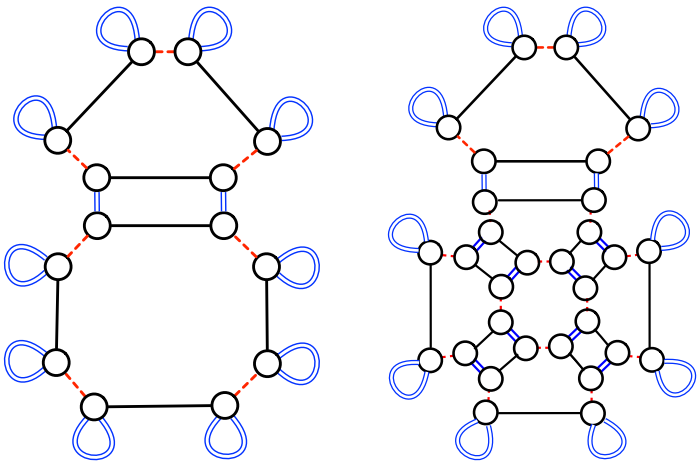
(a) Règle



(b) Instanciation

FIGURE 3.27 – Ajout des liaisons explicites entres les instances.

Enfin, les différents nœuds sont reliés par des liaisons dites *explicites*, lors de l'instanciation, ces liaisons sont dupliquées pour relier deux à deux chaque brins instance des nœuds. Par exemple et comme montré sur la figure 3.27, la liaison α_0 lie deux à deux les brins instances du nœud `n2` (en jaune) avec les brins instances du nœud `n1` (en bleu). De même, les brins instance de `n0` sont liés avec ceux de `n1` par les liaisons α_1. Finalement, la règle instanciée peut être appliquée sur la G-carte 3.28(a)), pour obtenir la G-carte transformée (figure 3.28(b)).



(a) G-carte (b) G-carte transformée

FIGURE 3.28 – Face rectangulaire triangulée.

L'éditeur de Jerboa vérifie automatiquement la préservation de la cohérence

topologique de l'objet lors de la création de la règle. En effet, dans [Pou09] des conditions syntaxiques sont définies pour garantir cette préservation. Par exemple, pour garantir la contrainte d'arcs incidents, tous les nœuds ajoutés au membre droit doivent posséder toutes les liaisons de α_0 jusqu'à α_n. En particulier, le nœud `n2` qui est ajouté à la règle 3.27 dispose bien de ses trois liaisons α_0, α_1 et α_2. La liaison α_0 est explicite entre les nœuds `n1` et `n2`. Les liaisons α_1 et α_2 sont implicite et portées par le nœud lui-même. Les autres contraintes de cohérence topologique (contrainte de cycle et celle de non orientation) sont également vérifiées.

3.4.1.2 Transformation des plongements

Comme expliqué précédemment, il faut ajouter des plongements à la structure topologique pour définir l'objet avec sa géométrie et ses propriétés physiques. Pour cela, le langage de règles permet de calculer les plongements pour les objets transformés [Bel12]. Rappelons que chaque plongement est porté par un type bien défini d'orbites. Nous poursuivons cette partie avec la même règle de triangulation précédente complétée par la transformation des plongements (voir figure 3.29). Le nœud `n2` du membre droit porte le plongement `point` qui permet de calculer les cordonnées du barycentre associé au nouveau sommet. Les trois nœuds du membre droit portent le plongement `color` qui colorie les nouvelles faces créées avec un mélange de la couleur de la face filtrée avec celle de la face voisine. Les nœuds `n0` et `n1` ne portent pas de plongement `point`, car les points des sommets filtrés restent inchangés. Ainsi, la règle complétée transforme l'objet comme montré par la figure 3.30 quand on l'applique sur le carré bleu.

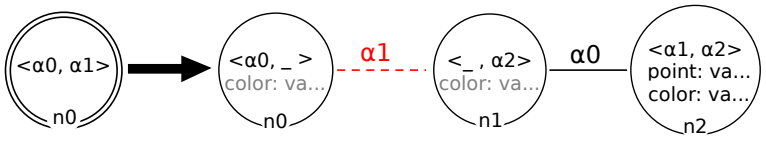


FIGURE 3.29 – Règle de triangulation et coloration d'une face.

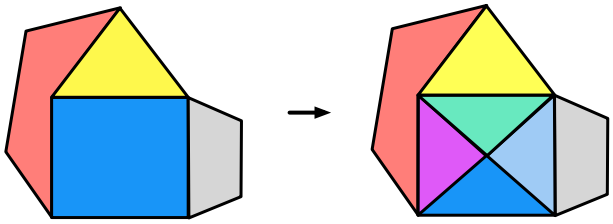


FIGURE 3.30 – Exemple d'application de la règle 3.29.

Cependant, la version courante de Jerboa utilisée dans cette thèse n'offre pas de langage pour spécifier la transformation des plongements. Cette dernière est donc écrite directement dans le langage cible, pour manipuler et calculer ces plongements, Jerboa offre différentes méthodes. Par exemple, pour calculer la couleur de chaque

nouvelle face triangulaire, nous faisons la moyenne entre la couleur de la face initiale et la face voisine comme on peut le voir dans l'application de la règle sur la figure 3.30. Tout d'abord, pour accéder au plongement d'un nœud filtré, ou plus exactement au plongement de ses instances (qui sont des brins de l'objet filtré), Jerboa utilise la méthode `ebd` qui est générique en fonction du plongement `T`. Cette méthode a comme prototype `<T> T ebd(String name)` (avec `T` le type du plongement et `name` le nom du plongement). Ainsi, par exemple, pour accéder à la couleur de la face initiale, il faut utiliser l'expression : `n0().<Color> ebd("color")` qui retourne le plongement `color` des brins filtrés par le nœuds `n0` qui est de type `Color`. De même pour `n0().<Point> ebd("point")`, permet d'accéder au plongement point du brin courant instance de `n0`.

Une autre méthode, très utilisée, existant dans la bibliothèque Jerboa est la méthode `JerboaNode alpha(int i)` qui permet d'accéder au brin voisin selon la liaison α_i. Par exemple, pour accéder à la face voisine du brin courant, nous utilisons l'expression `n0.alpha(2)`, et l'expression `n0.alpha(2).<Color> ebd("color")` permet alors d'accéder à la couleur de la face voisine.

La bibliothèque Jerboa possède d'autres méthodes comme la `collecte`, qui permet de construire la collection des brins d'une orbite donnée. Ces différentes méthodes sont utilisées pour construire les expressions de plongement, qui sont des portions de code écrit en Java ou C++. En complément des méthodes de Jerboa, les expressions de plongements utilisent les méthodes statiques fournies par les classes de plongement. Par exemple, dans la règle de triangulation 3.29, pour calculer la moyenne des couleurs des faces, nous utilisons la méthode `middle` définie dans la classe de plongement `color` qui calcule la moyenne de deux couleurs données. L'expression du plongement portée par le nœud `n2` de la règle s'écrit alors de la façon suivant : `value = Color.middle(n0().<Color>ebd("color"), n0().alpha(2).<Color>ebd("color"));`.

Pour éviter de saisir plusieurs fois la même expression de plongement lors de l'écriture d'une règle, Jerboa propage automatiquement l'expression définie pour un nœud sur les nœuds de la même orbite support du plongement. Par exemple, dans la règle présentée sur la figure 3.29, comme `n0`, `n1` et `n2` appartiennent à la même orbite face $\langle \alpha_0, \alpha_1 \rangle$, le plongement couleur d'une face est défini pour le nœud `n2` (l'expression est noire) et propagé aux nœuds `n0` et `n1` (l'expression est alors grise).

Pour récapituler, les règles créées par Jerboa diffèrent grandement par leur objectifs. Il existe des règles qui apportent une modification de la structure (par exemple l'ajout d'un sommet et la couture de deux faces), on parle alors de *règle topologique*. Dans ces règles, on peut ajouter/supprimer des brins et/ou modifier/ajouter des liaisons. Mais il faut noter qu'il existe des règles qui permettent de modifier uniquement des plongements sans apporter de modification à la structure topologique comme par exemple l'ajout d'une couleur à une face et la modification de la position d'un sommet. La règle présentée sur la figure 3.31 permet de changer uniquement la forme des quadrilatères, car le membre gauche de la règle filtre explicitement un quadrilatère, en fournissant une nouvelle position à un des sommets

(sommet du coin supérieur gauche dans notre exemple). Un exemple d'application est donnée sur la même figure. Nous utilisons beaucoup ce type de règles dans notre travail, plusieurs autres exemples des règles qui ne modifient que les plongements sans modification topologique sont alors données dans les chapitres 5 et 6

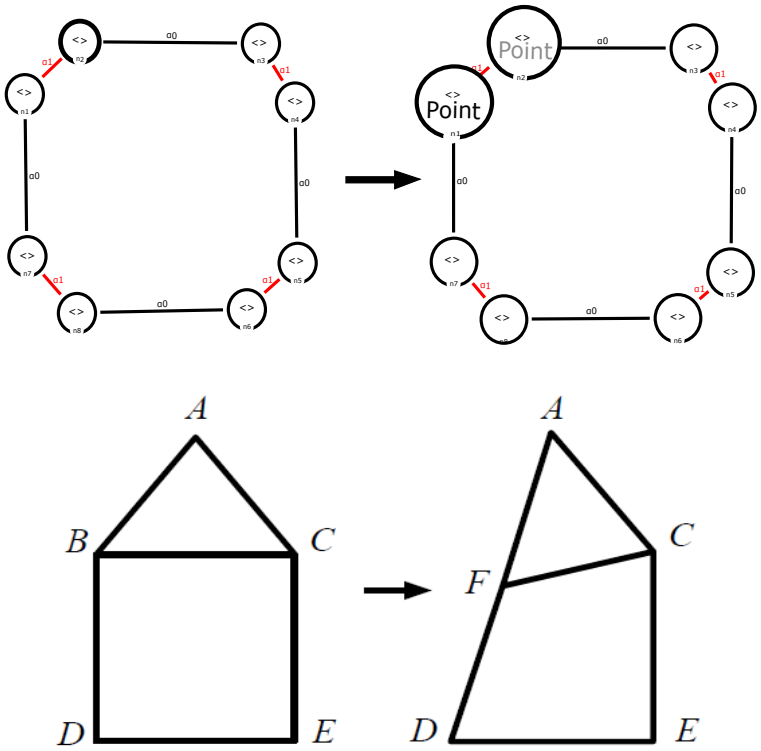


FIGURE 3.31 – Règle de changement de position d'un sommet d'un quadrilatère et son application.

Souvent, nous combinons à la fois des changements d'un plongement et de la topologie. Il est également possible de procéder à des changements simultanés de plongements différents. Par exemple, plutôt que de définir une règle comme celle présentée sur la figure 3.31 pour modifier la position d'un sommet et une autre pour modifier la couleur de la face quadrilatère, il est possible de combiner les deux, comme montré sur la figure 3.32, en définissant les deux plongements simultanément dans la même règle.

3.4.1.3 L'outil Jerboa

Jerboa est une bibliothèque qui permet de générer des modeleurs. Comme montré sur la figure 3.33, cet outil est composé : d'un *éditeur de modeleur* qui permet de générer le noyau de modeleur dédié correspondant, d'un noyau qui comprend le moteur d'application des règles et permet ainsi d'exécuter toutes les opérations spécifiées par les règles, et d'un visualiseur qui permet de réaliser facilement l'interface graphique des noyaux des modeleurs générés. Comme illustré sur la figure 3.33,

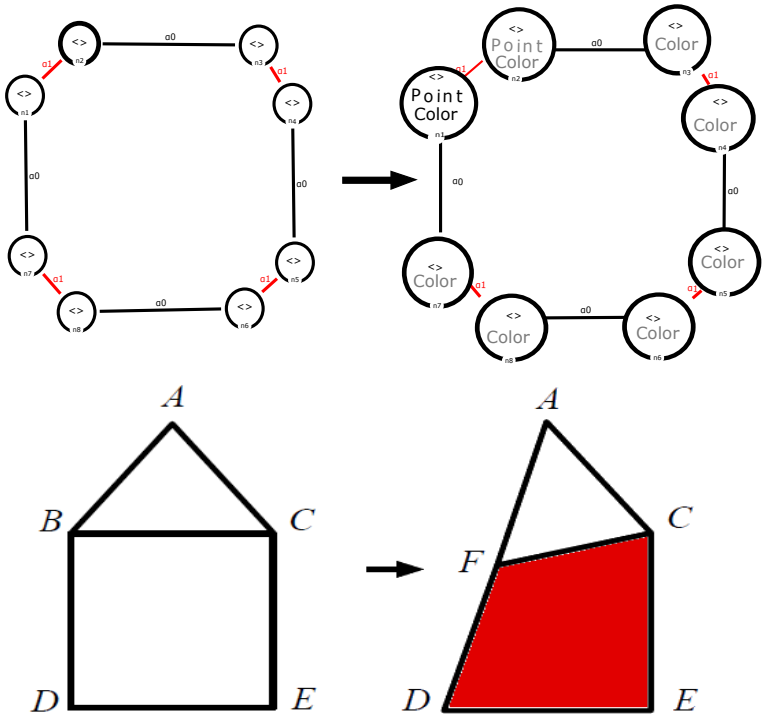


FIGURE 3.32 – Règle de changement de la forme et du couleur d'un quadrilatère et son application.

un utilisateur Jerboa est spécialiste de modélisation géométrique et développe, avec cet outil, un modeleur dédié. Il utilise l'éditeur de Jerboa, crée les classes de plongements et les règles qu'il faut pour son modeleur. C'est ce type d'utilisation que nous avons eu durant notre travail. L'utilisateur final de l'application est spécialiste du domaine cible, mais n'a pas besoin de connaître Jerboa, car il utilise directement le modeleur dédié via son interface graphique.

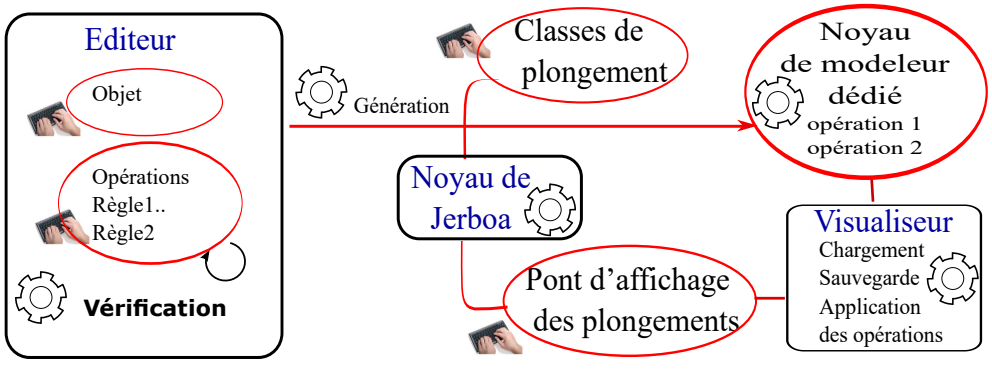


FIGURE 3.33 – Schéma d'utilisation de Jerboa.

Un premier prototype de Jerboa a été écrit par Thomas Bellet [BPA+10]. En-

suite en 2014, une version plus avancée, écrite en JAVA, a été introduite par Belhaouari *et al.* [BALB14]. Dans cette thèse, nous utilisons la version écrite en JAVA. De plus, il existe une autre version de Jerboa écrite en C++, qui est en cours de développement, introduite par Gauthier *et al.* [GBA15].

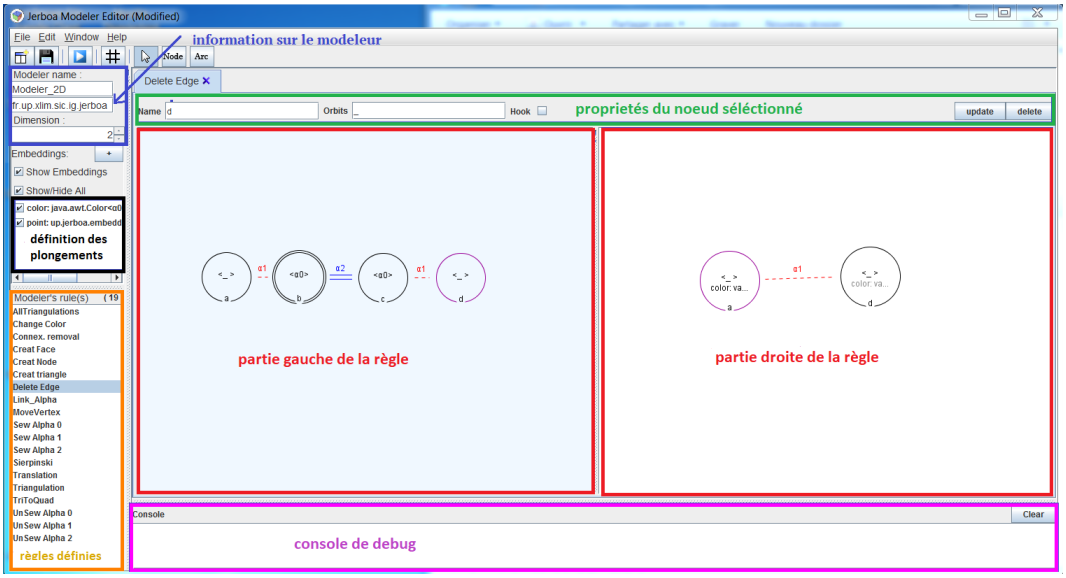


FIGURE 3.34 – Interface de l'éditeur de Modeleur de Jerboa.

Présentons en premier lieu l'éditeur de Jerboa. Pour créer un modeleur, il faut tout d'abord lui donner un nom et définir sa dimension topologique dans la partie liée aux informations sur le modeleur comme montré sur la figure 3.34 par le cadre bleu. L'étape suivante consiste à créer les plongements nécessaires et à les associer à des orbites de la structure topologique dans la zone appelée définition de plongements (voir cadre noir figure 3.34).

Après cette étape, l'utilisateur peut créer les opérations de son modeleur, sous la forme de règles (cadres rouges sur la figure 3.34). À la création d'une opération, l'interface de la figure 3.35 doit être renseignée. Outre le nom de l'opération et un commentaire, il est possible d'ajouter une pré-condition qui réduit les cas d'application de la règle. Par exemple, nous pouvons préciser que la règle de triangulation ne s'applique que sur les faces rouges ou que sur les faces à quatre cotés ou plus. Il est aussi possible d'ajouter à la règle des *extra-parameters*(pré-traitement), qui sont les paramètres de plongement de la règle, comme par exemple le vecteur de translation pour une règle de translation d'une composante connexe. Mais ce champ peut aussi permettre d'injecter du code qui est exécuté entre la phase de filtrage du membre gauche celle de transformation du membre droit. Nous l'utiliserons donc régulièrement pour effectuer des calculs communs aux différents plongements et ainsi optimiser l'exécution des transformations des plongements. Plusieurs exemples seront donnés ultérieurement dans le chapitre 6.

L'éditeur de Jerboa permet de vérifier automatiquement, lors de la création des

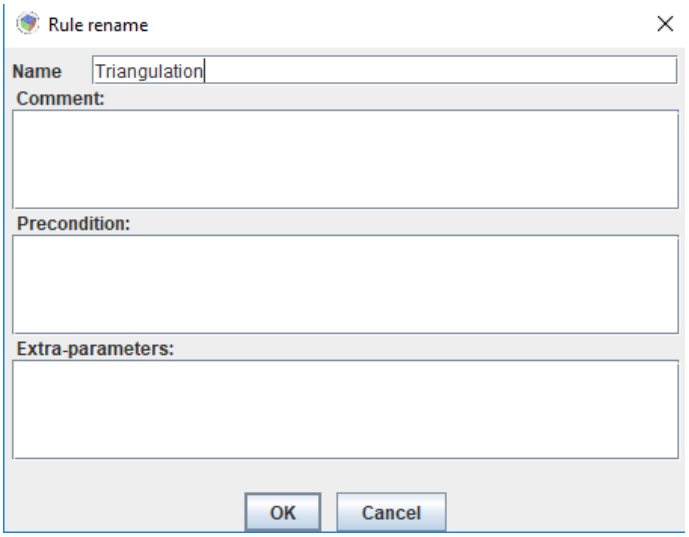


FIGURE 3.35 – Interface de modification d'une règle.

règles, la syntaxe graphique et les conditions de préservation de la cohérence topologique. Certaines vérifications sont aussi effectuées sur les expressions de plongements qui sont ensuite créés de manière cohérente par le moteur d'application des règles. S'il y a des erreurs, elles sont affichées dans la console (cadre en magenta figure 3.34). Finalement le moteur de règles de Jerboa assure que les règles préservent la cohérence des objets. Il s'agit d'une aide précieuse pour le développeur qui accélère grandement la mise au point des modeleurs dédiés.

3.4.2 Exemples de règles : création d'une surface en damier

Nous introduisons dans cette partie, un exemple d'utilisation du langage à base de règles pour la création d'une surface rectangulaire découpée en damier et quelques règles à appliquer pour faire des modifications topologiques et/ou modification des plongements sur cette surface.

Pour créer cette surface, nous avons conçu quatre règles qui sont appliquées successivement. Il s'agit de (1) la création d'un sommet, (2) la création d'une arête, (3) la création d'une face et finalement (4) la subdivision de face. Après l'application successive des trois premières règles puis plusieurs itérations d'application de la dernière règle, nous obtenons le damier. Dans la suite nous présentons ces règles une par une.

- Règle de création d'un sommet

Tout d'abord nous construisons une règle pour créer un sommet, comme le montre la figure 3.36. Comme c'est une règle de création, le membre gauche de la règle est vide. Elle crée un sommet libre en α_0, α_1, α_2 et α_3, il n'est donc constitué que d'un seul brin. Néanmoins, il faut introduire sa position initiale. Pour

cela, nous avons créé un plongement `point` de type vecteur 3D initialisé à une valeur donnée. Si l'application utilise d'autres plongements, il faut penser à les indiquer explicitement dans le nœud, afin de les initialiser (au besoin, en fixant une valeur par défaut si le plongement en question n'est pas encore calculable à ce stade de la construction). Cette règle peut aisément être étendue pour ajouter d'autres informations de plongements comme des propriétés physiques (masse, ressort...).

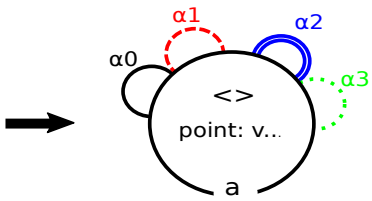


FIGURE 3.36 – Création d'un sommet.

- Règle de création d'une arête

À partir du sommet déjà créé, nous construisons une arête. Comme montré sur la figure 3.37, nous filtrons dans la partie gauche de la règle le sommet créé précédemment et nous ajoutons dans la partie droite de la règle un nouveau sommet. Pour créer une arête, qui a comme extrémités les deux sommets créés, nous ajoutons une liaison α_0 entre les deux. Après la construction de notre arête, nous définissons les valeurs des plongements associés au sommet nouvellement créé représenté par le nœud `b`.

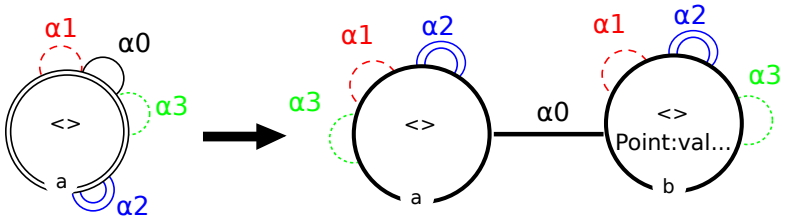


FIGURE 3.37 – Règle de création d'une arête.

Nous associons à ce sommet un nouveau plongement `point`. Là encore cette règle peut aisément être étendue à d'autres plongements.

- Règle de création d'une face

Nous proposons de créer une face rectangulaire grâce à l'extrusion de l'arête déjà créée par la règle précédente. Le principe est illustré sur la figure 3.38. À gauche, l'arête filtrée est l'arête précédente. À droite chaque brin filtré est dupliqué afin de construire successivement l'arête de départ (copie a), les arêtes de coté (copies b et c) et l'arête finale (copie d) qui est isomorphe à l'arête initiale.

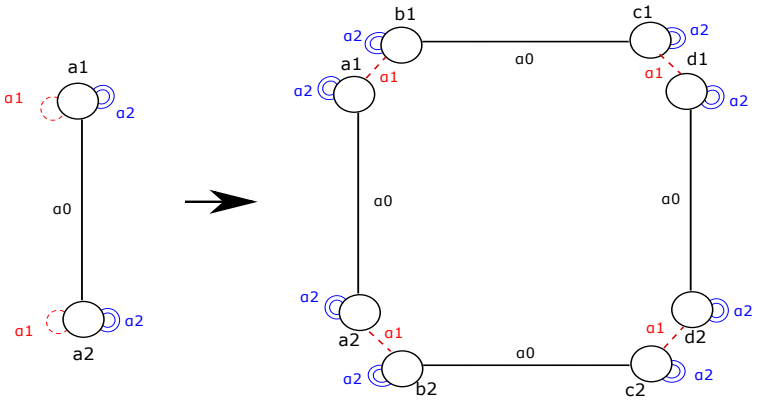


FIGURE 3.38 – Principe de création d'une face.

La règle de création est donnée sur la figure 3.39. Grâce à l'utilisation des variables topologiques, les brins de même lettre mais de numéros différents (figure 3.38) sont générés par un même nœud dans la règle (figure 3.39). Par exemple, les brins b_1 et b_2 sont créés par le nœud `b` de la règle. Comme les brins b_1 et b_2 ne sont pas liés entre eux (figure 3.38) le nœud `b` a sa liaison α_0 supprimée (figure 3.39).

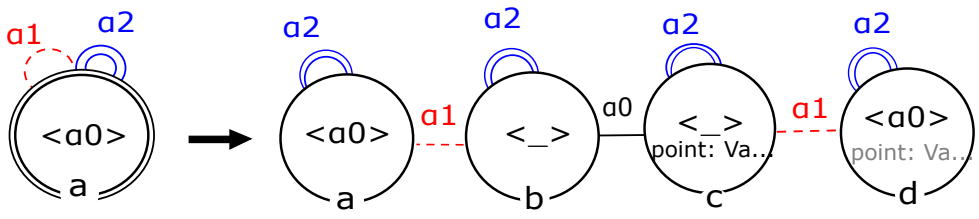


FIGURE 3.39 – Règle de création d'une face.

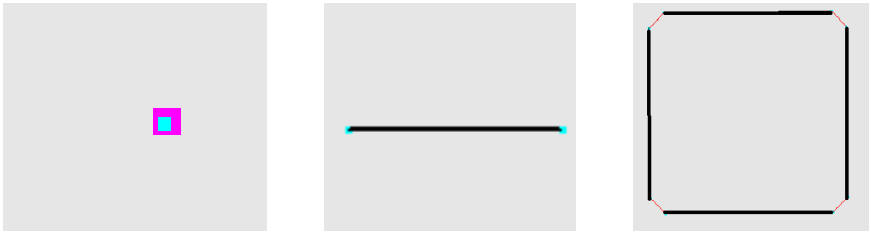
Dans la partie gauche de la règle, nous filtrons l'arête précédemment créée (le nœud `a` avec son orbite $\langle\alpha_0\rangle$) et dans la partie droite, nous créons (de chaque côté)

une nouvelle arête (nœuds `b` et `c`, liés par α_0) qui lui est adjacente via une liaison explicite α_1.

Le nœud `c` (et `d`) correspond aux nouveaux sommets (un de chaque côté), extrémités des nouvelles arêtes. Son plongement `point` est ici précisé. Il est défini simplement par translation des points d'origines. Le nœud `d` correspond à une duplication de l'arête initiale, où la liaison α_0 est préservée. Cette arête est liée aux sommets du nœud `c` par un lien α_1. Puisque le plongement `point` des sommets est déjà précisé dans `c`, l'expression est propagée sur le nœud `d`.

Rappelons que s'il y a d'autres plongements, il faut ajouter les expressions des plongements adéquates pour chaque nœud afin de préserver la cohérence de l'objet construit. L'éditeur de Jerboa vérifie que tous les plongements sont effectivement définis.

La figure 3.40 présente le résultat après l'application successive de la règle de création d'un sommet, puis celle de création d'une arête et enfin la création d'une face.



(a) Création d'un sommet (b) Creation d'une arête (c) Creation d'une face

FIGURE 3.40 – Résultat de l'application de trois règles successives pour créer une face.

- Règle de subdivision d'une face

L'opération de subdivision consiste à diviser notre face rectangulaire en quatre sous-faces rectangulaires d'égales dimensions. Ceci revient à insérer un sommet au milieu de chaque arête et au centre de la face, puis à former les quatre nouvelles faces à partir de ces sommets. Il est possible de ré-appliquer cette opération sur les sous-faces de façon itérative, pour découper la surface initiale en damier, comme montré sur la figure 3.41.

La figure 3.42 illustre comment la subdivision peut être définie en exploitant les variables topologiques. Comme pour la règle de création de face, les brins de même lettre, mais de chiffres différents (par exemple tous les a_i) correspondent au même nœud dans la règle que nous allons définir. En pratique, pour chaque brin des sommets initiaux a_i, il faut insérer un nouveau sommet b_i au milieu de son arête incidente. Puis, il faut ajouter les arêtes qui relient les sommets nouvellement créés, à partir de c_i vers le centre de la face initiale d_i.

Remarquons que ce même schéma de subdivision peut être utilisé sur des faces non quadrilatère, comme le montre la figure 3.43

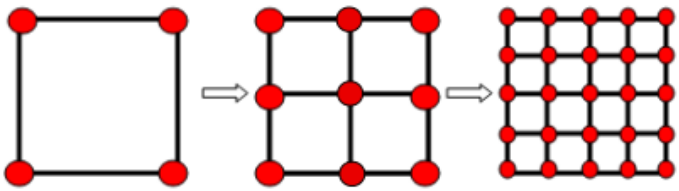


FIGURE 3.41 – Subdivisions successives d'une surface rectangulaire.

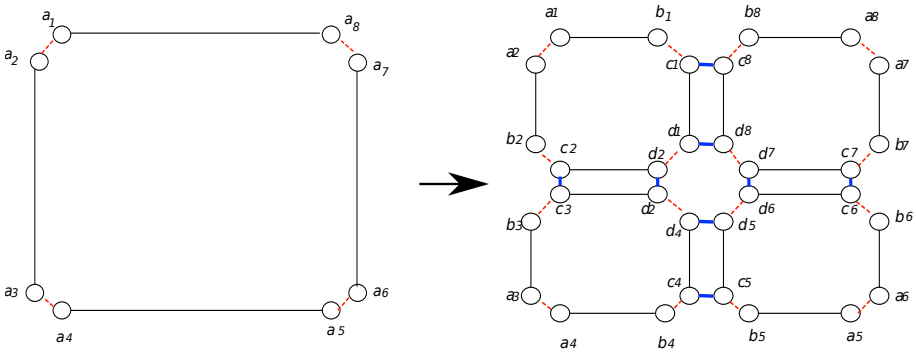


FIGURE 3.42 – Principe de la subdivision avec Jerboa.

La figure 3.44 illustre la règle de subdivision utilisée. Pour pouvoir appliquer le schéma de subdivision à toutes les faces (rectangulaires ou non) d'un objet, le membre de gauche filtre toute la composante connexe de l'objet 2D. Dans le membre de droit, nous supprimons tout d'abord les liaisons α_0 des faces initiales afin de découper les arêtes en deux. Ceci se traduit, dans le nœud `a` par une étiquette comprenant le symbole « _ » à la place de α_0. Nous construisons ensuite le nœud `b` (*i.e.* tous les brins b_i) et on leur attribue comme valeur de plongement `point`, le milieu de l'arête initiale de la face. Ensuite il faut connecter ce nœud `b` au nœud `a` par des liaisons α_0 pour construire des arêtes (*i.e.* on construit des arêtes entre les paires de brins a_i et b_i). Puis nous ajoutons un autre nœud `c` qui est une copie

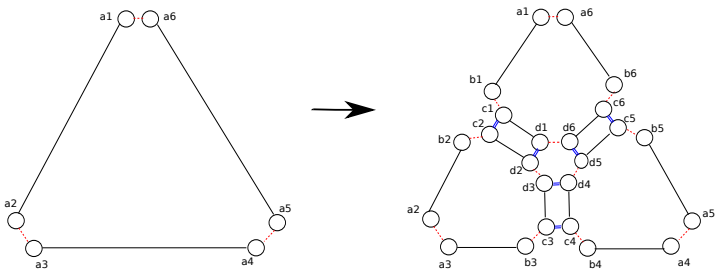


FIGURE 3.43 – Subdivision d'un triangle avec Jerboa.

du nœud `b` et qu'on relie à ce dernier par une liaison α_1. Enfin, il faut construire le sommet central qui est le centre de la face initiale. Nous créons alors un nœud `d` qui a comme valeur de plongement `point`, les cordonnées du centre. Ce dernier nœud est lié au nœud `c` par une liaison α_0. En exploitant le principe de renommage des liaisons dans l'étiquette du nœud `c`, les brins c_i sont liés entre eux par α_2 selon le schéma initial en α_0 des brins a_i. *Idem*, les brins d_i sont liés entre eux par α_2 et α_1 selon le schéma qui liait initialement les brins a_i entre eux par α_0 et α_1.

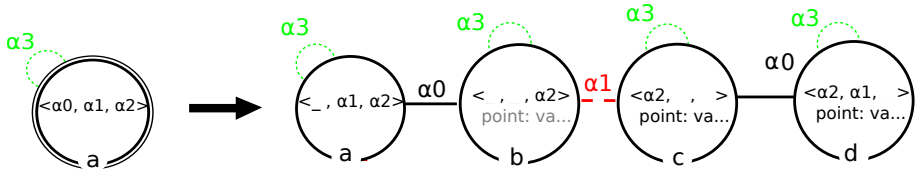
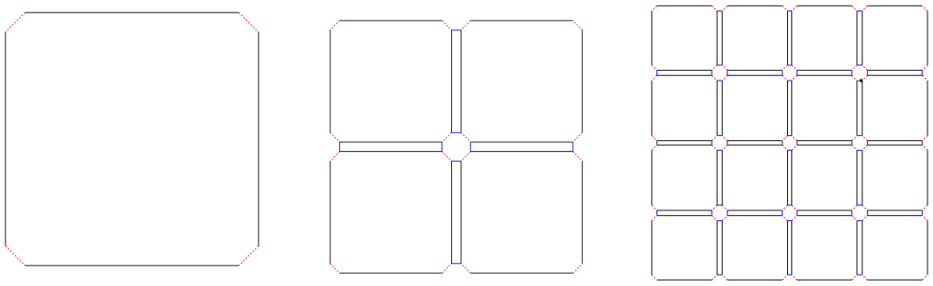


FIGURE 3.44 – Règle de subdivision des faces d'un objet connexe.

La figure 3.45 montre le résultat obtenu après une première, puis une seconde application de la règle. Il faut noter le fait que la règle s'applique à toute la composante connexe en un coup, c'est à dire sur toutes les faces en même temps. Nous ne sommes donc pas confrontés au problème du lien α_2 entre une arête de face qui a été subdivisée et une arête de face adjacente qui ne l'est pas encore (cas de la « jonction en T »).



(a) Face filtrée (b) Face avec une subdivision (c) Face après deux subdivisions

FIGURE 3.45 – Face subdivisée.

Pour pouvoir enchaîner les règles (trois règles de création de la surface, suivies d'un certain nombre de subdivisions), il est possible de les regrouper dans un script prenant la forme d'un programme Java. Notons que la prochaine version de Jerboa fournira un langage dédié pour définir plus facilement les scripts.

3.4.3 Autre exemples de règles

Dans cette partie, nous montrons d'autres exemples de règles dont nous nous sommes servis pour réaliser nos travaux. Nous avons montré, dans la section précédente, qu'il est possible de créer une face par extrusion. Néanmoins, il est également

possible de créer des faces en une seule règle comme montré sur la figure 3.46. Il s'agit d'une règle de création, le membre gauche est donc vide.

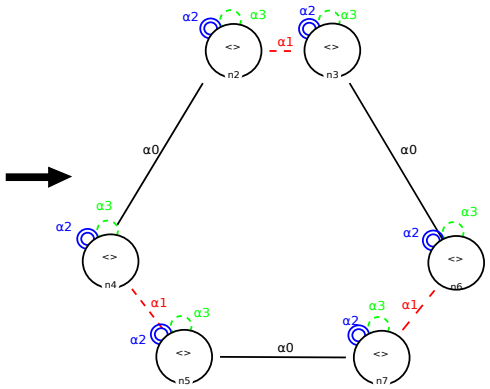


FIGURE 3.46 – Règle de création d'un triangle.

Au contraire, lorsque le membre droit de la règle est vide, alors il s'agit d'une règle de suppression. Par exemple, la règle de la figure 3.47 permet de supprimer une face isolée.

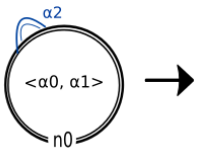


FIGURE 3.47 – Règle de suppression d'une face isolée

En effet, on reconnaît l'orbite de face $\langle \alpha_0, \alpha_1 \rangle$ en étiquette de nœud. En outre, la règle impose que la face soit libre en α_2 pour toutes ses arêtes, elle est donc obligatoirement isolée.

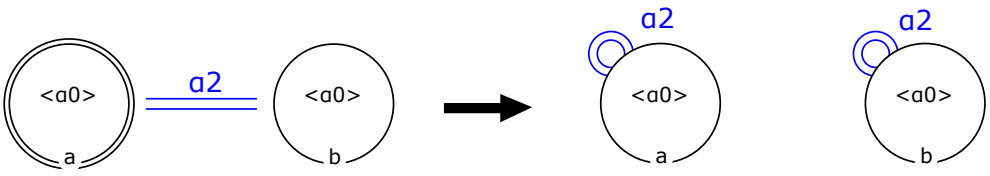


FIGURE 3.48 – Règle de décousure d'une arête

Pour isoler une face, il faut découdre toutes ses arêtes. La règle présentée sur

la figure 3.48, permet la décousure d'une arête. En effet dans le membre gauche de la règle, nous avons filtré explicitement deux arêtes de face, représentées par les deux nœuds `a` et `b` liées par une liaison α_2 (*i.e* une arête commune à deux faces adjacentes). Alors que dans le membre droit, nous supprimons la liaison α_2 entre les deux arêtes de face pour obtenir deux arêtes libres en α_2. Après l'application de cette règle sur toutes les arêtes d'une face, cette dernière devient isolée et par la suite, nous pouvons appliquer la règle de suppression d'une face isolée présentée sur la figure 3.47 comme le montre la figure 3.49.

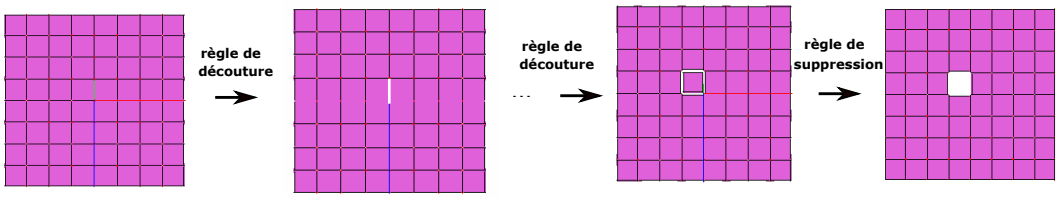


FIGURE 3.49 – Application successive de la règle d'isolement puis de la suppression d'une face.

Il faut noter que les deux règles 3.47 et 3.48 créées précédemment pour isoler une face et la supprimer, peuvent être réunies en une seule règle. Cette dernière est présentée sur la figure 3.50. Cette règle, comme on peut le remarquer, permet de filtrer dans le membre gauche une face, représentée par le nœud `a` dans la règle, avec toutes les arêtes des faces adjacentes (représentées par le nœud `b` dans la règle), alors que le membre droit permet de supprimer uniquement la face sélectionnée et de mettre à jour les liaisons α_2 des faces voisines.

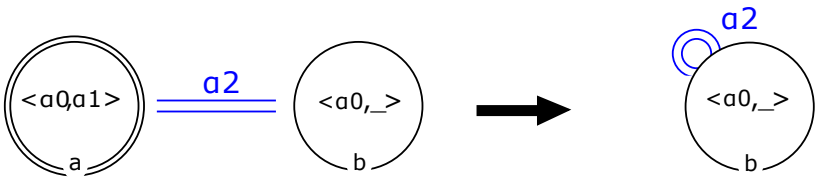


FIGURE 3.50 – Règle d'isolement et suppression d'une face

Avec Jerboa, il est également possible de mettre à jour un même plongement, pour toutes les orbites qui le portent. Par exemple, on peut demander de mettre à jour la position de tous les sommets d'une composante connexe, pour, par exemple, effectuer un déplacement ou une déformation d'un objet connexe. Ce type de règle prend toujours la même forme. Les deux motifs de la règle ne contiennent qu'un seul nœud qui est étiqueté par l'orbite composante connexe, à savoir l'orbite $\langle \alpha_0, \alpha_1, \alpha_2 \rangle$ en 2D et $\langle \alpha_0, \alpha_1, \alpha_2, \alpha_3 \rangle$ en 3D. Le nœud du membre droit contient le plongement à mettre à jour. Ce type de règle permet de gagner en temps de calcul car tous les plongements sont recalculés en une seule application de règle. Par exemple, la règle présentée sur la figure 3.51, permet de mettre à jour les positions de tous les

sommets d'une composante connexe en 3D.

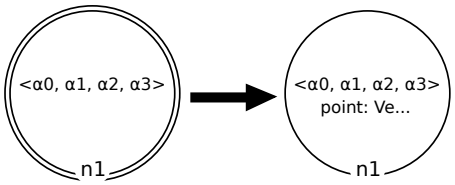


FIGURE 3.51 – Règle de mise à jour des positions des sommets dans toute une composante connexe 3D.

Un exemple d'application de cette règle, dans le cas particulier d'une rotation, sur la surface en damier précédemment créée est donné sur la figure 3.52

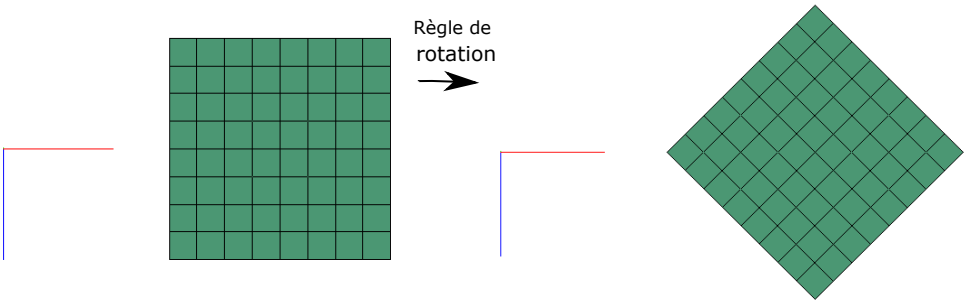


FIGURE 3.52 – Exemple d'application de règle de rotation de toute la composante connexe.

Pour obtenir le même résultat, il est possible d'écrire une règle qui filtre un brin et met à jour le plongement voulu. On applique alors cette règle, au travers d'un script, à tous les sommets de l'objet (ce script est équivalent à une boucle *for*). Cette approche est cependant moins efficace, en termes de temps de calcul, en général.

Cette section avait pour but de montrer les diverses formes que peuvent prendre les règles dans Jerboa. Il existe des règles de création ou de suppression des éléments et aussi des règles qui filtrent explicitement des cellules. Il existe également des règles plus génériques, dont les nœuds sont étiquetés par des variables topologiques et d'autres qui sont conçues pour n'effectuer que des mises à jour de plongement ou que des modifications topologiques. De même, il y a des règles qui filtrent toute une composante connexe et d'autres qui utilisent des scripts pour les appliquer sur toute la composante connexe. Pour cela, lors de la conception d'une règle, il faut réfléchir à son application, voire à sa généralisation (en type d'orbite, voire en dimension) afin de trouver la forme la plus pertinente qu'elle peut revêtir. Car souvent, plusieurs

possibilités co-existent, mais à des degrés divers de généricité et/ou d'efficacité.

3.5 Conclusion

Dans ce chapitre, nous avons présenté différents types de modèle topologique existants dans la littérature. Nous nous sommes focalisés sur les modèles cellulaires et en particulier sur ceux à représentation implicite des cellules qui offrent une solution robuste pour les opérations de modifications topologiques. En effet, ces dernières permettent en outre de garantir la propriété de quasi-variétés, indispensable à la simulation physique des objets maillés. Pour cela nous avons choisi de travailler avec le modèle topologique des G-cartes parce que c'est le modèle le plus atomique. Il nous fournit un nombre important d'orbites qui permettent de stocker les informations applicatives, dans notre cas liées à la mécanique, dont nous avons besoin. De plus ce modèle est exploité par le langage à base de règles Jerboa que nous avons utilisé. Ce langage permet de prototyper facilement un nombre très important d'opérations dans un temps réduit, car il permet de garantir la cohérence topologique de l'objet. Comme nous le détaillerons par la suite, il nous facilite la création d'un cadre général pour la simulation physique.