Un générateur de noyaux de modeleurs

M. Poudret, A. Arnould, Y. Bertrand

Laboratoire SIC / Université de Poitiers

poudret,arnould,bertrand@sic.univ-poitiers.fr

Résumé : *La majorité des modeleurs géométriques sont basés sur un modèle topologique fixe, adapté à un domaine particulier (conception mécanique, géologie, architecture, etc.). Devant la multiplicité de ces domaines, les chercheurs ont besoin de toujours plus de modeleurs, dont le temps de développement peut être très élevé. Néanmoins, il existe des analogies entre les modèles topologiques. En s'appuyant sur elles, nous nous proposons de créer un générateur de noyaux de modeleurs. Cet outil doit générer le code d'un noyau de modeleur à partir de la description d'un modèle topologique et de ses opérations. Les mois de développement seront ainsi transformés en quelques jours de choix ou mise au point du modèle. L'article présente les mécanismes à la base du Méta-Modeleur, puis en propose une implantation.*

Mots-clés : modélisation géométrique à base topologique, modeleur, ensemble semi-simplicial, carte généralisée, méta-modeleur

1 Introduction

Les modeleurs géométriques sont des logiciels de création et de manipulation d'objets géométriques. Ils utilisent une représentation (un modèle) fixe des objets, choisie en fonction du domaine d'application visé. Ce sont des logiciels complexes dont le coût de développement peut représenter de nombreuses années/hommes.

Dans un environnement de développements autour de la modélisation géométrique, les modèles utilisés sont nombreux car les natures et dimensions des objets représentés varient en fonction des applications visées (villes, bâtiments, animations, couches géologiques, etc.). De ce fait, les infographistes consacrent beaucoup de temps à développer de nouveaux modeleurs.

Cependant, des points communs existent entre les modèles. À partir de leur étude, nous souhaitons concevoir un générateur de code de noyaux de modeleurs, que nous appelons Méta-Modeleur. La simple description du modèle et de ses opérations sera transformée automatiquement en noyau de modeleur. Les mois de développement seront ainsi transformés en jour de choix ou mise au point du modèle.

Dans cet article, nous nous intéressons aux modeleurs géométriques à base topologique. En modélisation géométrique « classique », seules les formes et positions des objets sont représentées. En modélisation topologique, les objets sont caractérisés par leur structure, c'est-à-dire leur décomposition en termes de cellules de différentes dimensions : sommets, arêtes, faces, volumes, etc.

Dans la section 2, nous présentons deux modèles topologiques représentatifs : les cartes généralisées [Lie89] et les ensembles semi-simpliciaux [May67]. La section 3 est consacrée à la conception du Méta-Modeleur : nous y présentons son architecture, la structure de représentation des objets et le traitement des opérations de constructions. L'implantation de notre prototype de Méta-Modeleur est présentée dans la section 4. Enfin, en section 5, nous comparons notre prototype à Moka [VD03], en termes de parcours d'objets volumineux.

2 Deux modèles géométriques à base topologique

Les modèles topologiques utilisés de nos jours appartiennent pour la plupart à la classe des subdivisions d'espace (ou structures cellulaires). Dans celle-ci, un espace de dimension n est partitionné en $n+1$ familles de cellules de dimensions 0 à n munies de relations d'incidences et/ou d'adjacences.

Les modèles que nous traitons dans le cadre du Méta-Modeleur s'inscrivent tous dans cette classe. Néanmoins, au sein de cette unique classe, on distingue les modèles dans lesquels les cellules topologiques sont données explicitement (c'est le cas dans les ensembles semi-simpliciaux), des modèles dans lesquels les cellules sont données par

un parcours des éléments de base du modèle (comme dans les cartes généralisées). En ce sens, les deux modèles que nous avons choisis sont représentatifs de la classe traitée par le Méta-Modeleur.

Toutefois, il convient de préciser que dans certains modèles comme les arêtes ailées [Wei75], les cellules topologiques sont à la fois représentées explicitement et implicitement. Il va de soi que de tels modèles doivent être acceptés, par le Méta-Modeleur.

2.1 Les cartes généralisées

Les cartes généralisées sont une extension des cartes combinatoires [Tut84] [BS85]. Elles permettent de représenter la topologie des quasi-variétés[1] de dimension n, orientables ou non, avec ou sans bord.

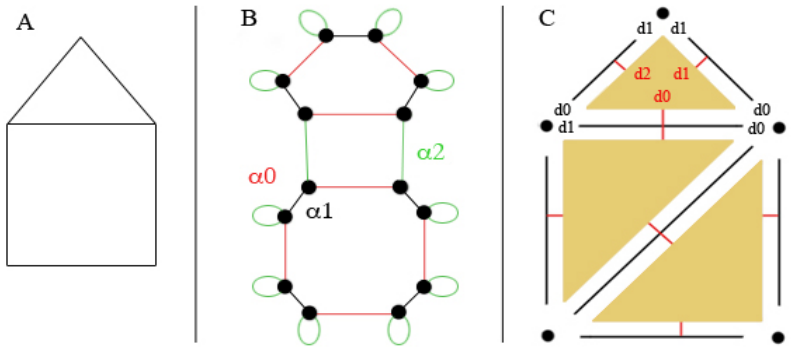

FIG. 1 – Un même objet représenté sous deux modèles distincts.

Intuitivement, une carte généralisée de dimension n (ou n-G-carte) est la donnée d'un ensemble d'éléments abstraits appelés brins, liés entre eux par des involutions[2] (chaque brin étant lié à $n+1$ brins distincts ou non). La cohérence des objets ainsi construits est assurée par la donnée d'une contrainte. Le formalisme mathématique des n-G-cartes est donné en définition 1. La figure 1 montre comment représenter un objet de dimension 2 (figure 1A) à l'aide une 2-G-Carte (figure 1B).

Définition 1 (Cartes généralisées de dimension n) *Une carte généralisée de dimension $n \geq 0$ (ou n-G-carte) est une algèbre $G = (B, \alpha_0, ..., \alpha_n)$, où :*

- *B est un ensemble de brins ;*
- *$\alpha_0, ..., \alpha_n$ sont des involutions sur B ;*
- *$\alpha_i\alpha_j$ est[3] une involution pour tout i, j tels que $0 \leq i < i+2 \leq j \leq n$.*

Les objets sont construits puis modifiés à l'aide d'opérations. Prenons comme exemple l'opération de couture, qui, associée à l'opération d'ajout d'un brin, permet de construire toutes les quasi-variétés.

Coudre deux brins d'une même carte consiste à les lier par une involution. Dans la plupart des cas, l'ajout de ce seul lien ne suffit pas (définition 2). Afin d'assurer la cohérence de l'objet (dernier point de la définition 1), il est souvent nécessaire d'effectuer plusieurs liaisons supplémentaires.

Définition 2 (Couture de brins dans une carte généralisée de dimension n) *Soient $G = (B, \alpha_0, ..., \alpha_n)$ une n-G-carte, b et b' deux brins de B et un entier i tel que $0 \leq i \leq n$. Soit φ un isomorphisme de l'orbite[4] $< \alpha_0, ..., \alpha_{i-2}, \alpha_{i+2}, ..., \alpha_n >(b)$ dans l'orbite $< \alpha_0, ..., \alpha_{i-2}, \alpha_{i+2}, ..., \alpha_n >(b')$ tel que $\varphi(b) = b'$. Soit $G' = (B', \alpha'_0, ..., \alpha'_n)$ la n-G-carte résultat de la i-couture de b et b' dans G. G' est définie par :*

[1] Une quasi-variété de dimension n est un objet de dimension n obtenu par assemblage de n-cellules le long de $(n-1)$-cellules. De plus, une $(n-1)$-cellule ne peut pas appartenir au bord de plus de deux n-cellules.

[2] Une application f est une involution si et seulement si ff est l'identité.

[3] Si β et γ sont des applications de $E \rightarrow E$, nous notons $\beta\gamma$ la composition $\gamma \circ \beta$, et $b\beta\gamma$ l'application de cette composition à un élément b de E.

[4] L'orbite $< \alpha_1, ..., \alpha_k > (b)$ est l'ensemble des brins accessibles à partir de b, par composition d'involutions de $\{\alpha_1, ..., \alpha_k\}$.

- $B = B'$;
- *pour tout* j *tel que* $0 \leq j \leq n$ *et* $j \neq i$, $\alpha'_j = \alpha_j$;
- $\alpha'_i = \begin{cases} \varphi(e) \textit{ si } e \in < \alpha_0, ..., \alpha_{i-2}, \alpha_{i+2}, ..., \alpha_n > (b) \textit{ ;} \\ \varphi^{-1}(e) \textit{ si } e \in < \alpha_0, ..., \alpha_{i-2}, \alpha_{i+2}, ..., \alpha_n > (b') \textit{ ;} \\ \alpha_i(e) \textit{ sinon.} \end{cases}$

2.2 Les ensembles semi-simpliciaux

L'espace de représentation des ensembles semi-simpliciaux est plus vaste que celui des cartes généralisées, puisque l'on ne se limite plus aux quasi-variétés. Il couvre ainsi tout objet géométrique subdivisé.

Un ensemble semi-simplicial est une famille d'objets abstraits appelés simplexes, liés entre eux par des opérateurs de bord pour lesquels il existe des contraintes de cohérence (définition 3). Sur la figure 1C, l'objet 1A est représenté à l'aide du modèle semi-simplicial.

Définition 3 (Ensemble semi-simplicial de dimension n**)** *Un ensemble semi-simplicial de dimension* $n \geq 0$ *est une algèbre* $S = (K, (d_j)_{j=0,...,n})$*, où :*

- $K = \bigcup_{i=0}^{n} K^i$*, avec* K^i *un ensemble fini de simplexes de dimension* i *;*
- *pour tout* $i \geq 1$*, et tout* $0 \leq j \leq i$*,* d_j *est une application* $K^i \rightarrow K^{i-1}$ *appelée opérateur de bord ;*
- *pour tout* i *tel que* $2 \leq i \leq n$*, et pour tout* k *et* j *tels que* $0 \leq k < j \leq i$*, pour tout* $\sigma \in K^i$*,* $\sigma d_j d_k = \sigma d_k d_{j-1}$.

La contrainte de cohérence (dernier point de la définition) peut être reformulée de la manière suivante : pour tout simplexe σ de dimension n (ou n-simplexe), deux $(n-1)$-faces[5] quelconques de σ ont au moins une $(n-2)$-face commune (le bord d'un triangle contient exactement trois sommets).

Dans les ensembles semi-simpliciaux, deux cellules de même dimension sont collées par identification de leurs bords (définition 4). Tout objet géométrique subdivisé est obtenu par composition des opérations d'identification et d'ajout d'un simplexe.

Définition 4 (Identification de simplexes dans un ensemble semi-simplicial de dimension n**)** *Soient* $S = (K, (d_j)_{j=0,...,n})$ *un ensemble semi-simplicial,* σ_1 *et* σ_2 *deux k-simplexes de K tels que si* $k > 0$ *alors* σ_1 *et* σ_2 *ont le même bord.* $S' = (K', (d'_j)_{j=0,...,n})$*, l'ensemble semi-simplicial résultat de l'identification de* σ_1 *et* σ_2 *dans S, est défini par :*

- $K' = K - \{\sigma_1, \sigma_2\} \cup \{\sigma_3\}$ *où* σ_3 *est un nouveau simplexe issu de l'identification de* σ_1 *et* σ_2 *;*
- *pour tout* i *tel que* $0 \leq i \leq k$ *avec* $k \geq 1$*, si* $\sigma_i = \sigma_1 d_i = \sigma_2 d_i$ *alors* $\sigma_3 d'_i = \sigma_i$*, où ;*
- *pour tout* $\sigma \in K$ *et pour tout* $0 \leq i \leq k+1$ *avec* $k \geq 0$*, si (*$\sigma d_i = \sigma_1$ *ou* $\sigma d_i = \sigma_2$*) alors* $\sigma d'_i = \sigma_3$*, sinon* $\sigma d'_i = \sigma d_i$.

3 La conception du Méta-Modeleur

Dans cette partie, nous présentons l'architecture du Méta-Modeleur, puis nous nous appuyons sur les deux modèles topologiques présentés afin de détailler notre réprésentation des objets d'une part, et la génération du code des opérations d'autre part.

[5] σ' est une face de σ s'il existe un opérateur de bord liant σ à σ'.

3.1 L'architecture du Méta-Modeleur

Notre Méta-Modeleur est un générateur de code, et en tant que tel, possède une architecture de compilateur. En entrée, il prend la description d'un modèle topologique et de ses opérations (à gauche sur la figure 2). Il effectue son analyse (en haut, au centre), puis génère le code correspondant aux structures de données ainsi qu'aux opérations (en bas, au centre). Afin de rendre le code généré exploitable, nous devons lui associer une interface de programmation (à droite).

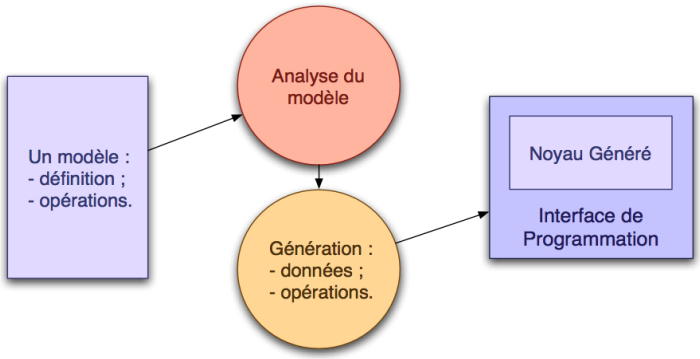

FIG. 2 – L'architecture du Méta-Modeleur.

3.2 Choisir une structure de données

3.2.1 Vers une structure générique

Notre but est de générer une représentation efficace des objets. Cependant, l'efficacité d'une structure de donnée ne dépend pas que de sa représentation, mais aussi de son utilisation. Par exemple ajouter de la redondance en augmentant le nombre de liens ou d'éléments de base peut rendre plus rapide le parcours des objets, mais ralentira leur modification (car les éléments à mettre à jour seront plus nombreux). L'efficacité d'une représentation ne peut donc pas être évaluée *a priori*, mais seulement en fonction de ses opérations de manipulation et de leur fréquence d'utilisation. Pour le Méta-Modeleur, nous avons donc choisi une représentation raisonnablement efficace (voir le paragraphe 5) sans prétendre pour autant quelle soit optimale.

Dans le Méta-Modeleur, nous avons choisi d'exploiter les similitudes entre les différents modèles topologiques pour proposer une structure de donnée générique unique que nous spécialisons en fonction du modèle fourni par l'utilisateur. Ainsi le code produit pour les données est particulièrement simple (voir le paragraphe 4.1).

Si l'on considère l'ensemble des modèles topologiques connus, on note des similitudes dans leur architecture. Quel que soit le modèle étudié, nous avons toujours des éléments de base (éventuellement étiquetés par une dimension) ainsi que des applications permettant de passer d'un élément à un autre (nous parlons de liens entre ces éléments). Bien sur, la nature des éléments de base (brins, simplexes, etc.), la nature des liens (involutions, opérateurs de bord, etc.) et le nombre de liens attachés à un élément de base varient sensiblement d'un modèle topologique à un autre. La structure de donnée générique (voir la définition 5) que nous proposons est donc constituée d'un ensemble d'éléments de base étiquetés par leur dimension et l'ensemble de leurs voisins indexés par les liens (l'ensemble des liens dépend uniquement de la dimension de l'élément de départ).

Définition 5 (Structure topologique générique) *La structure topologique générique de dimension n est un couple $T = (E, L)$, où :*

- $E = \bigcup_{i=0}^{n} E^i$, *avec E^i un ensemble fini d'éléments de base de dimension i ;*
- $L = \bigcup_{i=0}^{n} L^i$, *où L^i est l'ensemble des liens des éléments de E^i, c'est-à-dire tel que tout lien l de L^i est une application de E^i vers E^j ($0 \leq j \leq n$).*

3.2.2 Validation de la structure générique

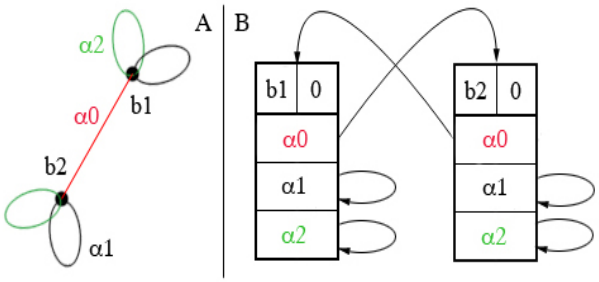

FIG. 3 – Application de la structure générique aux 2-G-cartes.

Dans un premier temps, nous allons utiliser notre structure générique avec les cartes généralisées (voir partie 2.1). Dans le cas d'une carte généralisée de dimension n, nous instancions notre structure topologique générique de la manière suivante :

- $E = B$, est l'ensemble des brins de la carte, notons que dans une carte généralisée les brins ont tous la même dimension (0 par défaut) ;
- $L = \{\alpha_0, ..., \alpha_n\}$, est l'ensemble de liens correspondants aux involutions.

La figure 3A donne un exemple de 2-G-carte dans lequel deux brins b_1 et b_2 sont liés par α_0 pour former une arête. Sur la figure 3B, nous avons deux brins b_1 et b_2 auxquels on attache l'étiquette (la dimension) par défaut : « 0 », et l'indexage des liens par α_0, α_1 et α_0. Les liens eux-mêmes sont représentés par les flèches sortant de chacun des index. Il est donc facile de stocker une carte généralisée à l'aide de notre structure générique.

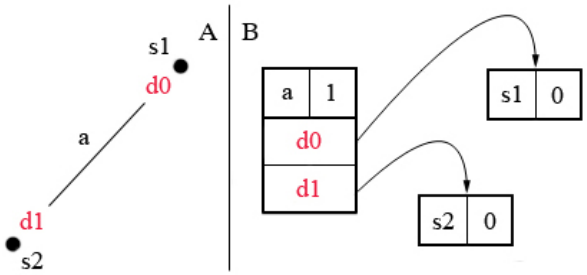

FIG. 4 – Application de la structure générique aux ensembles semi-simpliciaux de dimension 2.

Si nous traitons les ensembles semi-simpliciaux de dimension n, nous instancions notre structure topologique générique de la manière suivante :

- $E = \bigcup_{i=0}^{n} K^i$, avec K^i l'ensemble des simplexes de dimension n ;
- $L = \bigcup_{i=1}^{n} L^i$, où $L^i = \{d_0, ..., d_i\}$ est l'ensemble des opérateurs de bord liant les i-simplexes de K^i aux $(i-1)$-simplexes de K^{i-1}.

Sur la figure 4, nous appliquons la structure générique aux ensembles semi-simpliciaux (voir partie 2.2). Cette figure utilise les mêmes conventions graphiques que précédemment. La figure 4A contient deux 0-simplexes s_1 et s_2 ainsi qu'un 1-simplexe a. L'arête a est liée aux sommets s_1 et s_2 par deux opérateurs de bords d_0 et d_1. Contrairement aux G-cartes, les éléments de base des ensembles semi-simplicaux peuvent être de plusieurs dimensions (0 ou 1 dans notre cas). Les 0-simplexes n'ont pas d'opérateurs de bord ; c'est pourquoi dans la figure 4B, les représentations s_1 et s_2 ne contiennent aucun lien.

3.3 Formaliser les opérations de construction

3.3.1 La reformulation des contraintes de cohérence

Dans la partie 2, nous avons étudié deux exemples d'opérations de construction : la couture de deux brins d'une n-G-carte et l'identification de deux simplexes d'un ensemble semi-simplicial. Dans les deux cas, nous pouvons décomposer l'opérations en deux étapes : 1) les modifications locales ; 2) le rétablissement des contraintes de cohérence. Par exemple, lors d'une couture par α_i, il faut dans un premier temps ajouter localement un lien entre les deux brins à coudre, puis dans un second temps ajouter d'autres liaisons nécessaires à la cohérence de l'objet résultant. L'utilisateur du Méta-Modeleur peut donc se contenter de fournir la partie locale de chaque opération. La cohérence de l'objet est automatiquement rétablie d'après les contraintes du modèle.

Néanmoins, nous n'avons pas de garantie qu'à partir de ces contraintes telles qu'elles sont écrites dans les définitions de la partie 2, il est possible de déduire des algorithmes complets. Cette traduction de formules mathématiques en code opérationnel est un problème difficile auquel nous ne pouvons apporter de réponse directe. Toutefois, nous proposons une manière de générer le code des opérations à partir d'une reformulation plus constructives des contraintes de cohérence.

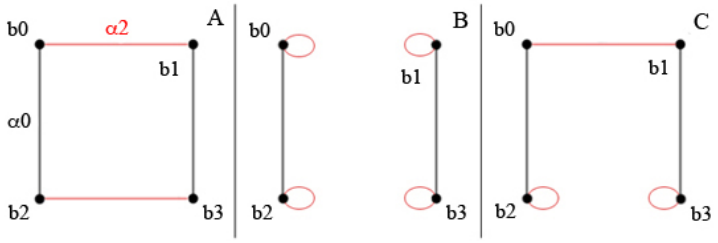

FIG. 5 – La contrainte des G-cartes.

Prenons l'exemple de la contrainte de cohérence des 2-G-cartes : $\alpha_0\alpha_2$ est une involution. Sur la figure 5, les objets A et B respectent la contrainte de cohérence. En effet, pour tout $i \in \{0,1,2,3\}$, $b_i\alpha_0\alpha_2$ est une involution. En C, l'objet est incohérent puisque par exemple : $b_0\alpha_0\alpha_2\alpha_0\alpha_2 = b_1 \neq b_0$. Ce problème peut se produire dans deux cas :

- un lien α_2 vient d'être ajouté entre b_0 et b_1 dans l'objet B. Il faut alors ajouter un lien α_2 entre b_2 et b_3 afin de rétablir la cohérence de l'objet. On se ramène ainsi au cas A. Il est possible de refomuler cette propagation en : si on ajoute un lien α_2 entre deux brins b_0 et b_1 alors il faut ajouter un lien α_2 supplémentaire entre leur image par α_0, b_2 et b_3 dans notre exemple ;
- le lien α_2 vient d'être supprimé entre b_2 et b_3 dans l'objet A. On se ramène alors au cas B en supprimant le lien α_2 entre b_0 et b_1.

Ainsi, les contraintes globales peuvent être exprimées sous la forme de propagations locales. Ces dernières précisent pour chaque lien modifié pour un élément de base donné quelles sont les liens modifiés pour ses différents voisins.

3.3.2 Validation de la reformulation des contraintes

Les équations 3.1 à 3.3 présentent le formalisme retenu pour l'expression locale des contraintes de cohérence. Dans le cas des 2-G-cartes, l'équation 3.1 signifie que si il existe un lien α_2 (que l'on vient d'établir) entre deux brins a et b alors il doit exister un lien α_2 entre $a\alpha_0$ et $b\alpha_0$. Ceci correspond exactement à ce que nous avons introduit dans le paragraphe 3.3.1.

$$(a\alpha_2 = b) \quad \Rightarrow \quad (a\alpha_0\alpha_2 = b\alpha_0) \tag{3.1}$$

$$(a\alpha_0 = b) \quad \Rightarrow \quad (a\alpha_2\alpha_0 = b\alpha_2) \tag{3.2}$$

Il est possible d'exprimer de la même manière les contraintes de cohérence des ensembles semi-simpliciaux de dimension 2 (équations 3.2 et 3.3 où figurent seulement deux des cinq contraintes, les trois autres étant homologues).

Le double indiçage des opérateurs de bords facilite la lecture des contraintes (nous le conservons dans le code généré) : d^i_j lie un simplexe de dimension i à un simplexe de dimension $i-1$, l'indice j est l'indice utilisé dans la définition 3. Notons que l'ajout (ou la suppression) d'un lien peut induire plusieurs changements topologiques (opérateur *et*).

$$(ad^2_2 = b) \quad \Rightarrow \quad (ad^2_1 d^1_1 = bd^1_1) \wedge (ad^2_0 d^1_1 = bd^1_0) \tag{3.3}$$

$$(ad^1_1 = b) \quad \Rightarrow \quad (\forall c\ cd^2_2 = a, cd^2_1 d^1_1 = b) \wedge (\forall c\ cd^2_1 = a, cd^2_2 d^1_1 = b) \wedge (\forall c\ cd^2_0 = a, cd^2_2 d^1_0 = b) \tag{3.4}$$

4 L'implantation du prototype

Dans cette partie, nous proposons une implantation OCaml de notre structure topologique générique ainsi que des contraintes de cohérence.

4.1 Implantation de la structure topologique générique

La structure générique proposée dans la partie 3.2 peut être implantée par un graphe orienté dans lequel les arcs sont indexés par un nom de lien (par exemple « α_0 », « d_0 », etc.). En effet, si nous associons les sommets aux éléments de base et les arcs aux liens, alors pour chaque couple (sommet, étiquette d'arc), il doit exister un unique arc dans le graphe.

Nous souhaitons implanter notre prototype en OCaml [Inr85]. En plus des caractéristiques communes à l'ensemble des langages fonctionnels, OCaml propose une surcouche efficace d'écriture de modules. La puissance de ce surlangage est due à l'utilisation de fonctions particulières permettant d'écrire des modules paramétrés par d'autres modules : les foncteurs. La bibliothèque de graphes généraux OCamlGraph [CFS05] est basée sur l'utilisation de tels foncteurs. Ceci permet de construire facilement des graphes spécialisés tels que ceux dont nous avons besoin dans le cadre du Méta-Modeleur.

Parmi les structures de graphes proposées, nous choisissons les graphes orientés étiquetés. Notons que l'on étiquette à la fois les arcs et les sommets (les étiquettes des sommets contiennent une information utilisée lors de l'affichage des éléments de base). La version impérative de cette structure est la plus performante, le foncteur idoine est *Graph.Imperative.Digraph.AbstractLabeled* dont l'unique paramètre est le module des étiquettes attachées aux sommets. Il renvoie un foncteur (en OCaml, il est facile d'écrire une fonction dont le résultat est une fonction) prenant le module des étiquettes des arcs en entrée. Ce dernier renvoie le module de graphes orientés étiquetés dont le type des étiquettes est décrit par les modules passés en paramètre des foncteurs.

Conformément aux remarques précédentes, la description du modèle topologique doit contenir les informations suivantes (elles sont extraites par l'analyseur syntaxique du Méta-Modeleur) :

- le nom du module des étiquettes attachées au sommet. Ce module n'est pas généré puisqu'il est lié au plongement. Il sera fourni directement par l'utilisateur ;
- le nom des liens ;
- le nom du modèle. Il permet de rendre plus explicite le code généré.

Le code source 1 est le résultat de l'instanciation de notre structure topologique générique dans le cas de cartes généralisées de dimension 2. Il est généré à partir des informations suivantes :

- *Point2D* comme nom du module des étiquettes attachées aux sommets ;
- *ALPHA_0, ALPHA_1* et *ALPHA_2* comme nom des liens ;
- *Gmap2D* comme nom du modèle.

Label (lignes 1 à 7) est le module des étiquettes attachées aux arcs. Ces étiquettes, utilisées comme système d'indexation des liens, sont de type *t* (ligne 2). Dans le cas des 2-G-cartes, il comprend trois constructeurs constants *ALPHA_0 ALPHA_1* et *ALPHA_2*, qui correspondent chacun à un index de lien. En tant que paramètre d'un foncteur, *Label* doit avoir une certaine signature, imposée par les développeurs d'OCamlGraph. Pour cette raison, nous associons à *Label* quatre fonctions :

```
 1 : module Label = struct
 2 :    type t = ALPHA_0 | ALPHA_1 | ALPHA_2
 3 :    let compare = compare
 4 :    let hash = Hashtbl.hash
 5 :    let equal = (=)
 6 :    let default = ALPHA_0
 7 : end ;;
 8 : open Label ;;
 9 :
10 : module Gmap2D =
11 :    (Graph.Imperative.Digraph.AbstractLabeled
12 :       (Point2D))
13 :          (Label) ;;
14 : open Gmap2D ;;
```

Code 1 – Structure générée dans le cas des 2-G-Cartes.

- *compare* (ligne 3) permet de définir un ordre sur les étiquettes ;
- *hash* (ligne 4) est une fonction de hachage ;
- *equal* (ligne 5) est la fonction d'égalité entre deux étiquettes ;
- *default* (ligne 6) est la valeur par défaut d'une étiquette.

4.2 Implantation des contraintes de cohérences

Les contraintes de cohérences sont données au Méta-Modeleur sous forme d'implications (paragraphe 3.3.2), elles sont traduites en listes OCaml (notées entre crochets). Le code source 2 montre la représentation des contraintes des cartes généralisées de dimension 2.

```
1 : let contraintes = [
2 :    (C_alpha_0, [
3 :       ([(C_alpha_2, true)], [(C_alpha_2, true)], C_alpha_0)
4 :    ]) ;
5 :    (C_alpha_1, []) ;
6 :    (C_alpha_2, [
7 :       ([(C_alpha_0, true)], [(C_alpha_0, true)], C_alpha_2)
8 :    ]) ;
9 : ] ;;
```

Code 2 – Contraintes générées dans le cas des 2-G-Cartes.

Chaque élément de la liste exprime les modifications topologiques à effectuer lors de la modification d'un lien. Par exemple, ligne 2, la modification d'une liaison α_0 entre deux brins a et b entraine la modification, ligne 3, de la liaison α_0 entre leurs deux voisins $a\alpha_2$ et $b\alpha_2$.

```
1 : let coudre = fun carte brin_a brin_b involution contraintes ->
2 :    lier carte brin_a brin_b involution ;
3 :    propager carte brin_a brin_b involution contraintes
4 : ;;
```

Code 3 – Couture de deux brins.

L'opération de couture de deux brins est donnée dans le code source 3 (remarquons que ce code est indépendant de la dimension de la carte). La partie locale de l'opération (ligne 2) est donnée par l'utilisateur. Les fonctions *lier*

(ligne 2) et *propager* (ligne 3) sont générées par le Méta-Modeleur, elles sont indépendantes du modèle. *propager* assure la cohérence de l'objet en propageant récursivement les contraintes de cohérence (un marquage des éléments de base assure la terminaison dans tous les cas).

5 Performances

En modélisation géométrique à base topologique, il est fréquent de manipuler des scènes complexes. Dans cette partie, nous montrons l'efficacité de l'implantation de notre structure topologique générique à l'aide d'OCamlGraph. Pour ce faire, nous allons comparer le Méta-Modeleur à un modeleur spécialisé sur des objets volumineux : Moka [VD03]. Ce dernier est un modeleur à base de 3-G-cartes écrits en C++, qui permet de travailler sur de grandes scenes avec des temps de parcours performants.

Nous allons procéder comme suit : dans un premier temps nous utilisons le noyau de Moka afin de créer une 3-G-carte dont nous connaissons la taille en termes de nombre de brins, ensuite nous créons un graphe de taille équivalent à l'aide de la structure de graphe retenue pour l'implantation de notre structure générique. Enfin, nous mesurons le temps de parcours de ces deux structures. Notons que nous n'avons pas de contrôle sur la forme du graphe (il est généré aléatoirement). Néanmoins cela nous suffit puisque nous ne nous intéressons pas à la forme exacte de la structure, mais à sa taille en mémoire.

Le résultat de ce comparatif est donné en figure 6. L'écart entre la courbe d'OCamlGraph et celle de Moka est presque nul. De plus, les deux courbes suivent la même progression linéaire en le nombre de brins (sommets dans le cas du graphe). Les résultats de ce test montrent que l'implantation choisie reste efficace lorsqu'elle est confrontée à de gros objets.

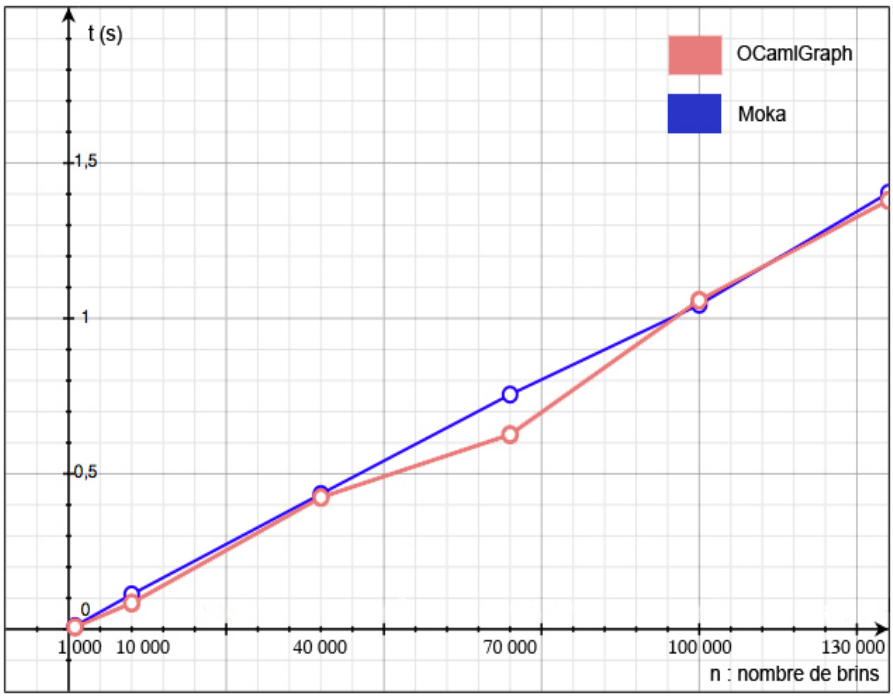

FIG. 6 – Comparaison OCamlGraph/Moka.

6 Conclusion

Nous avons proposé puis implanté nos solutions pour deux des principales difficultés inhérentes à la conception d'un générateur de noyaux de modeleurs (ou Méta-Modeleur). À partir des analogies qui existent entre les différents modèles topologiques, nous avons proposé une structure de données générique pour le stockages des objets. Seule l'instanciation de cette structure la spécialise en fonction du modèle d'entrée. De plus, nous générons le code des opérations de construction à partir d'une reformulation des contraintes de cohérence des modèles, qui permet à l'utilisateur de définir simplement ses opérations.

Notre prototype de Méta-Modeleur offre de bonnes performances, nous pouvons encore les améliorer en spécialisant la représentation du graphe qui implante notre structure topologique générique. En particulier : nous utilisons les étiquettes des arcs comme index pour les liens, mais l'accès au voisin selon un lien donné n'est pas constant (il dépend du lien). Grâce aux foncteurs et aux modules d'OCaml, nous pourrions modifier OCamlGraph et introduire un indexage direct des voisins sans modifier la structure de la bibliothèque et donc celle du Méta-Modeleur.

Au delà du Méta-Modeleur ce travail pourra déboucher sur le développement d'un modeleur générique qui permettrait à la fois la paramétrisation (par différents liens, plongements, opérations, etc.) et la spécialisation (par certains algorithmes et/ou structures de données plus efficaces mais moins généraux). Un tel modeleur (ou bibliothèque de modeleur) permettrait de mutualiser les développements pour des applications très diverses dans des domaines variés. Son architecture novatrice pourrait être mise en oeuvre grâce au langage de module puissant d'OCaml.

Références

[BS85] R. Bryant and D. Singerman. Foundations of the theory of maps on surfaces with boundaries. *Quart. J. Math. Oxford Ser. (2)*, 36(141) :17–41, 1985.

[CFS05] F. Conchon, J.-C. Filliâtre, and J. Signoles. Le foncteur sonne toujours deux fois. In *JFLA'05*, pages 79–93. INRIA, March 2005.

[Inr85] Inria. Objective caml. caml.inria.fr/, 1985.

[Lie89] P. Lienhardt. Subdivision of n-dimensional spaces and n-dimensional generalized maps. In *Annual Symposium on Computational Geometry SCG'89*, pages 228–236, Saarbruchen, Germany, June 1989. ACM Press.

[May67] J.-P. May. *Simplicial Objects in Algebric Topology*, volume 11 of *Van Nostrand Mathematical Studies*. Van Nostrand, premire edition, 1967.

[Tut84] W. Tutte. *Graph Theory*, volume 21 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley, 1984.

[VD03] F. Vidil and G. Damiand. Moka. www.sic.sp2mi.univ-poitiers.fr/moka/, 2003.

[Wei75] K. Weiler. Edge-based data structures for solid modeling in curved-surface environments. *Comput. Graph. Appl.*, 5(1) :21–40, 1975.