Représentation et manipulation de structures topologiques dans un langage fonctionnel

Antoine SPICHER — Olivier MICHEL

IBISC, FRE 2873 CNRS, Université d'Évry
Tour Evry-2, 523 Place des Terrasses de l'Agora
91000 Évry, France

RÉSUMÉ. *Le* calcul spatial *est un domaine de recherche qui vise à développer des formalismes, des langages et des architectures matérielles qui permettent d'appréhender la notion d'espace en informatique. Nous présentons* MGS*, un langage de programmation déclaratif dédié à la représentation et à la manipulation de structures spatiales arbitrairement complexes. Afin d'atteindre ce but, nous avons mis en place un formalisme de calcul non conventionnel fondé sur deux notions, les* collections topologiques*, une nouvelle structure de données reposant sur la notion de* complexe cellulaire *développée en topologie algébrique, et les* transformations*, une forme originale de définition par cas de fonctions pour manipuler par réécriture locale des collections topologiques. Nous illustrons ce formalisme en décrivant la représentation à l'aide des collections topologiques d'une classe particulière d'espaces, les* quasi-variétés*, et en implantant avec des transformations un algorithme classique de raffinement de maillages.*

ABSTRACT. Spatial computing *is a research field that is aimed at developing formalisms, languages and architectures in order to take into account the notion of space in computer science. We present* MGS*, a programming language dedicated to the representation and the handling of arbitrary complex spatial organizations. To achieve this goal, we define an unconventional formalism based on two notions:* topological collections*, a new data structure relying on the notion of* cellular complex *developed in algebraic topology, and* transformations*, an original kind of case-based definition of functions to handle topological collections by rewriting. We illustrate this formalism by describing the representation in terms of topological collections a particular class of spaces, the* quasi-manifolds*, and by implementing with transformations a standart mesh refinement algorithm.*

MOTS-CLÉS : *Système de réécriture, collection topologique, transformation,* MGS

KEYWORDS: *Rewriting system, topological collection, transformation,* MGS

RSTI - TSI – 26/2007. Langages applicatifs, pages 1169 à 1196

1. Introduction

Le *calcul spatial* (DeHon *et al.,* 2006) est un domaine de recherche qui vise à développer des formalismes, des langages et des architectures matérielles qui permettent de prendre en compte la notion d'espace en informatique, que ce soit comme une ressource (par exemple pour implanter des circuits) ou comme résultat (par exemple en CAO). Plus particulièrement, ce domaine cherche à manipuler de manière *intentionnelle* des objets *spatialement structurés*.

La manipulation intentionnelle de ces objets permet de les considérer comme un « tout » et non à travers les éléments qui les constituent. Ce type de manipulation s'oppose à une manipulation *extensionnelle* faisant apparaître des schémas explicites de parcours des éléments de la structure.

On entend par objets spatialement organisés, des systèmes composés d'entités localisées et organisées par une relation de *voisinage*. Cette relation correspond à une forme de « proximité » entre les éléments qui constituent l'objet, proximité entraînant la possibilité pour ces entités d'interagir entre-elles.

Ce domaine trouve un grand nombre d'applications dans divers champs d'activité. La CAO est bien évidemment un domaine d'application privilégié du calcul spatial mais il en existe bien d'autres.

Cette problématique est par exemple abordée par le *calcul amorphe* (Abelson *et al.,* 2000) : un milieu amorphe est un support de calcul composé d'un très grand nombre d'entités qui coopèrent de façon dynamique, irrégulière et défaillante. La programmation d'un tel support demande de passer de la spécification d'un objectif global à la définition d'un programme local à chaque entité. Plusieurs langages ont déjà été développés pour des applications dédiées (GPL (Coore, 1991), OSL (Nagpal, 2001), AML (Beal, 2005)…).

On peut également citer l'étude et la modélisation de phénomènes d'*auto-organisation* ou d'*auto-assemblage*. Il s'agit de processus créant de façon incrémentale des structures spatiales complexes. On trouve dans la nature un grand nombre d'exemples de systèmes auto-assemblés : de la cristallisation en physique jusqu'aux processus de morphogénèse en biologie. Il n'existe cependant dans ce domaine aucune théorie générale et unificatrice. Néanmoins, de nombreuses applications concrètes existent comme le calcul par pavage de tuiles d'ADN (Rothemund *et al.,* 2004), l'assemblage de robots configurables (Klavins, 2002), la formation de nanomatériaux intelligents (Goldstein *et al.,* 2005).

Le calcul spatial n'est pas seulement restreint à ces nouveaux supports de calcul, mais peut également offrir un point de vue nouveau dans des domaines plus anciens. En analyse numérique par exemple, la simulation de champs physiques demande la manipulation d'objets spatialement organisés et arbitrairement complexes, comme pour la résolution d'équations aux dérivées partielles. Le point de vue pris en physique discrète avec le calcul extérieur discret (Hirani, 2003, Desbrun *et al.,* 2006)

par exemple, amène à considérer la modélisation des systèmes physiques comme du calcul spatial.

Pour finir cette série d'exemples, on peut trouver des applications dans des domaines comme l'intelligence artificielle. Par exemple, l'agrégation spatiale est un formalisme fournissant un mécanisme pour transformer une entrée numérique en une suite de descriptions de plus haut niveau en spécifiant des métriques, des relations de voisinages et des relations d'équivalence sur les données de plus bas niveau (Yip *et al.*, 1996).

Dans le cadre de cet article, basé partiellement sur un premier travail publié dans (Spicher, 2005), nous nous concentons sur des domaines où l'espace apparaît comme le résultat d'un calcul ou comme une des données du problème à résoudre.

Le projet MGS[1] développe un langage éponyme dédié aux applications liées au calcul spatial. Les langages dédiés, souvent déclaratifs et organisés autour d'un petit noyau, proposent de nouvelles abstractions et notations orientées vers un domaine d'application particulier, rendant la programmation et la réutilisation de code plus faciles. Avec MGS, nous proposons de représenter dans un cadre générique différents types d'organisations spatiales, et de les manipuler simplement et uniformément. Pour cela, nous avons développé les *collections topologiques*, une notion abstraite fondée sur le concept de *complexe cellulaire* provenant de la topologie algébrique et qui permet de représenter des espaces de façon combinatoire, et les *transformations*, une forme originale de définition de fonctions par cas adaptée à la réécriture locale de collection topologique.

La section 2 définit formellement les collections topologiques comme des espaces engendrés par des complexes cellulaires et décorés par des valeurs. Elle décrit également le paradigme de calcul proposé par les transformations, et plus particulièrement les *transformations de patch* dédiées à la modification de la topologie des collections.

La section 3 présente une implantation d'un type particulier de collection topologique fondé sur la notion de *G-carte*. Les G-cartes sont une structure de données qui permet la représentation des quasi-variétés, une classe d'espaces topologiques. Elles sont utilisées en CAO pour la description et la manipulation efficace d'objets solides.

La section 4 clôt cet article en proposant un exemple d'utilisation des transformations de patch. Nous proposons une implantation en MGS du raffinement de maillage, une opération importante dans le domaine de la CAO qui consiste à raffiner des objets géométriques. La subdivision est une opération souvent décrite informellement, qui permet de modifier le maillage de telle sorte que l'objet représenté affiche une courbure cohérente. Mais son implantation dans un langage impératif (ou fonctionnel)

1. Le projet MGS, acronyme de « encore un **M**odèle **G**énéral de **S**imulation » a été initié par Jean-Louis GIAVITTO et Olivier MICHEL en 2000 pour prendre en compte la modélisation et la simulation de systèmes dynamiques dont la structure évolue au cours du temps avec comme champ d'application privilégié les problèmes de morphogénèse.

classique est souvent une opération délicate. Son implantation immédiate et directe en MGS valide l'expressivité des notions développées.

2. Formalisation

Dans cette section, nous développons les notions de *collection topologique* et de *transformation* qui vont respectivement nous permettre de représenter et de manipuler des espaces.

2.1. *Représentation de l'espace*

2.1.1. *Complexe cellulaire*

La *topologie algébrique* est un domaine des mathématiques étudiant les espaces topologiques en leur associant un objet algébrique (groupe, espace vectoriel, etc.) dont les propriétés servent à déterminer un certain nombre d'invariants caractérisant la topologie de l'espace initial. Notre utilisation de la topologie algébrique se restreint au concept de *complexe cellulaire abstrait*. Un complexe cellulaire abstrait est une construction formelle qui spécifie un espace de façon combinatoire à l'aide d'objets plus simples, les *cellules topologiques*, chacune représentant de façon abstraite une partie de l'espace considéré. Chaque cellule est caractérisée par une *dimension*, la dimension de la partie représentée ; la structure de l'espace, correspondant à la partition en cellules topologiques, est considérée à travers une relation dite d'*incidence* liant deux cellules « voisines » dans la partition.

Définition 1 (Complexe cellulaire abstrait) *Soit S un ensemble arbitraire. Soit $\preceq$ un ordre partiel (une relation binaire réflexive, transitive et antisymétrique) sur les éléments de S. Soit une fonction $dim : S \to \mathbb{N}$ assignant un entier positif ou nul noté $dim(\sigma)$ à chaque élément $\sigma \in S$ tel que si $\sigma \prec \tau$ alors $dim(\sigma) < dim(\tau)$.*

Le triplet $\mathcal{K} = (S, \prec, dim)$ est un complexe cellulaire abstrait. *Les éléments de S sont appelés les* cellules *du complexe $\mathcal{K}$; en particulier, si $dim(\sigma) = n$ pour $\sigma \in S$ alors n est la* dimension *de σ et σ est appelé* n-cellule . *La relation $\preceq$ est appelée* relation d'incidence *; on dira de deux cellules en relation par $\preceq$ qu'elles sont* incidentes.

On dira qu'un complexe cellulaire abstrait est de dimension finie *s'il existe un entier N tel que $\forall \sigma \in S,\ dim(\sigma) \leq N$. Le plus petit entier N vérifiant cette propriété est appelé la* dimension *du complexe $\mathcal{K}$.*

Une n-cellule représente un espace élémentaire de dimension n. En particulier, les 0-cellules sont des sommets, les 1-cellules des arcs, 2-cellules des faces, 3-cellules des volumes, etc. Ainsi, les graphes sont des exemples de complexes cellulaires de dimension 1.

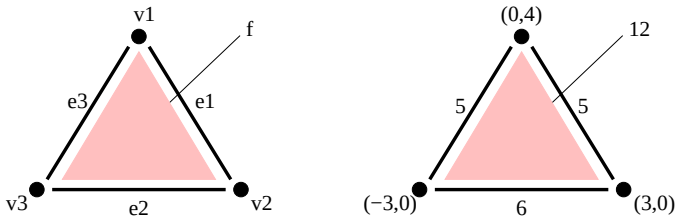


Figure 1. *Sur la figure de gauche, un exemple de complexe cellulaire : il est composé de trois* 0*-cellules (*v_1*,* v_2*,* v_3*), de trois* 1*-cellules (*e_1*,* e_2*,* e_3*), et d'une* 2*-cellule* f*. Le bord de* f *est constitué de ses cellules incidentes* v_1*,* v_2*,* v_3*,* e_1*,* e_2 *et* e_3*. Plus particulièrement, les trois arcs sont les faces de* f*, et par conséquent,* f *est une coface de* e_1*,* e_2 *et* e_3*. Sur la figure de droite, des données sont associées aux cellules topologiques : des positions pour les sommets, des longueurs pour les arcs et une aire pour* f

Outre la représentation d'une partition possible de l'espace euclidien en cellules topologiques, la relation d'incidence permet de parcourir de proche en proche les cellules du complexe. Nous utilisons en particulier deux opérateurs de voisinage fondés sur les notions de *face* et de p*-voisinage*.

Définition 2 (Voisinages) *Soit* $\mathcal{K}$ *un complexe cellulaire, et* σ *une cellule de* $\mathcal{K}$*. On appelle* face *de* σ *une cellule* τ *de* $\mathcal{K}$ *telle que :*

$$\tau \prec \sigma \quad \textit{et} \quad dim(\tau) = dim(\sigma) - 1$$

On dit également que σ *est une* coface *de* τ*. Cette relation est notée* $\tau < \sigma$*.*

Soient n *et* p *deux entiers. Deux* n*-cellules de* $\mathcal{K}$*,* σ_1 *et* σ_2*, sont dites* p-voisines *s'il existe une* p*-cellule* τ *de* $\mathcal{K}$ *telle que* σ_1 *et* σ_2 *sont toutes les deux incidentes à* τ*.*

L'incidence est à rapprocher de la notion de bord. Par exemple, si deux sommets v_1 et v_2 sont les faces d'un même arc e (autrement dit v_1 et v_2 sont 1-voisines et $v_1 < e > v_2$), ils constituent le bord de l'arc e. La figure 1 de gauche donne un exemple de complexe cellulaire. Nous ne détaillerons pas plus ces notions, mais le lecteur intéressé peut approfondir ce sujet avec (Munkres, 1984, Giavitto *et al.*, 2002).

2.1.2. *Collection topologique : décoration de l'espace*

Un complexe cellulaire définit uniquement la structure d'un espace. A partir de cette structure, on définit une *collection topologique* comme l'association de données aux cellules topologiques du complexe. Cette façon de procéder se rapproche de la notion de *champ de données* où les données sont indexées par les éléments de $\mathbb{Z}$ (Lisper, 1993, Hillis *et al.*, 1986). Les champs de données utilisés en parallélisme (data-field) correspondent à une généralisation des régions arbitraires de $\mathbb{Z}^n$ et généralisent la

notion de tableau. La notion de collection topologique permet d'aller plus loin en considérant des espaces plus structurés et/ou plus complexes que $\mathbb{Z}^n$, représentés par des complexes cellulaires.

Définition 3 (Collection topologique) *Soient $\mathcal{K} = (S, \prec, dim)$ un complexe de dimension N et V un ensemble arbitraire de valeurs. On définit une* collection topologique *sur $\mathcal{K}$ à valeurs dans V, comme une fonction partielle de S dans $\mathcal{V}$. On note $C(\mathcal{K}, V)$, l'ensemble des collections topologiques sur $\mathcal{K}$ dans V.*

Une telle collection peut être représentée par une somme formelle associant une valeur à chaque élément d'un complexe :

$$c \in C(\mathcal{K}, \mathcal{V}), \sigma \in \mathcal{K} \quad c(\sigma) = v_\sigma \textit{ s'écrit } c = \sum_{\sigma \in \mathcal{K}} v_\sigma.\sigma$$

La figure 1 donne un exemple de collection topologique.

2.2. *Manipulation de l'espace*

Les collections topologiques permettent de représenter un large spectre d'espaces. Pour les manipuler, nous mettons en place une structure de contrôle, appelée *transformation*, permettant la définition par cas de fonctions sur les collections topologiques. Les transformations reposent sur la notion de *réécriture locale de sous-collection* : chaque cas correspond à une règle de réécriture *filtrant* une sous-partie de la collection pour la modifier localement.

2.2.1. *Fonctions définies par cas*

Dans les langages de type ML (comme Haskell ou OCaml), il est possible de définir des fonctions par cas sur des constantes. Ce principe de définition se rapproche d'une structure d'aiguillage. La fonction *factorielle* s'écrit ainsi en OCaml :

```
let rec fact = function
  | 0 -> 1
  | x -> x*fact(x-1)
```

Cette fonction est spécifiée par deux cas suivant la valeur de son argument. Le premier cas filtre la constante `0` ; ainsi, si l'argument de la fonction est `0`, l'évaluation de l'expression `1` sera déclenchée. La seconde partie de cette définition absorbe un argument de n'importe quelle valeur (ici, `x` ne pourra jamais prendre la valeur `0` qui est filtrée par la première règle). La variable `x` joue un rôle de *joker*, permettant la référence à la valeur de l'argument dans l'expression en partie droite.

Dans les langages Haskell et OCaml, la spécification des motifs ne se limite pas au filtrage de constantes (l'entier `0` dans notre exemple). La définition par cas des

fonctions permet notamment l'expression de motifs filtrant des objets plus complexes que de simples valeurs scalaires : les *types de données algébriques*. Ces valeurs permettent la représentation de structures définies inductivement. C'est le cas des arbres binaires localement complets non vides étiquetés aux feuilles :

```
type Ab = Leaf of int
        | Node of Ab * Ab ;;
```

Les arbres binaires sont soit des feuilles (`Leaf`), soit des nœuds (`Node`). `Leaf` et `Node` sont les *constructeurs* du type `Ab`. Ces constructeurs sont paramétrés par des attributs permettant leur décoration par des valeurs du langage. Ici, on associe un entier à chaque feuille et un couple d'arbres binaires à chaque nœud (représentant les fils droit et gauche). On utilise alors les constructeurs pour définir de nouveaux motifs expressifs spécifiant la structure des éléments à filtrer et des variables de motif pour référencer les attributs qui leur sont associés. Par exemple, on définit par cas la somme des valeurs associées aux feuilles d'un arbre de type `Ab` de la façon suivante :

```
let rec sum = function
  | Leaf n -> n
  | Node(l, r) -> sum(l) + sum(r)
```

Les définitions par cas présentent de nombreux avantages : elles facilitent le raisonnement équationnel sur les fonctions et fournissent un mécanisme concis et expressif pour la définition de fonctions. Néanmoins, étendre ce mécanisme à des types de données non algébriques (comme les graphes définis par une triangulation de Delaunay par exemple) reste une question ouverte.

2.2.2. *Réécriture locale de sous-collection*

Par leur définition inductive, les valeurs des types de données algébriques correspondent à une organisation hiérarchique ; il s'agit en effet d'arbres formels dont les valeurs sont construites par une racine colorée par un constructeur et pouvant exhiber un certain nombre de fils suivant la définition du type. Lors du filtrage tel qu'il vient d'être décrit, les différents cas de la définition des fonctions correspondent aux différents constructeurs sur lesquels les racines sont définies. Ainsi, pour la fonction `sum`, deux cas se présentent correspondant à une feuille ou à un nœud de l'arbre. Le filtrage consomme entièrement l'arbre, les sous-arbres étant nommés par des variables du filtre puis traités par des appels récursifs le cas échéant.

Contrairement aux types de données algébriques, il n'est pas possible en général de représenter les collections topologiques par une racine unique permettant de filtrer toute la structure de données. Cependant, en partant d'un des éléments de la collection puis en suivant la relation d'incidence, il est possible de sélectionner une partie de la collection topologique. Là où le filtrage des structures de données algébriques est *global* dans le sens où toute la structure est filtrée, le filtrage des collections topologiques est *local* : une sous-partie est sélectionnée.

Les *transformations* étendent en ce sens la définition par cas de fonctions aux collections topologiques. Dans une transformation, chaque cas est décrit par une règle de réécriture $\alpha \rightarrow \beta$:

– la partie gauche α est un *motif* qui sélectionne par filtrage une sous-partie de la collection, appelée *sous-collection*,

– la partie droite β est une expression qui calcule une nouvelle collection à substituer en lieu et place de la sous-collection filtrée par α.

Une transformation est définie par un ensemble de règles de réécriture et son application sur une collection topologique correspond aux trois opérations suivantes :

1) *application d'une règle* : cette étape retourne un couple (sous-collection filtrée, sous-collection calculée) ; le premier élément est issu du filtrage respectant le motif de la règle, et le second correspond à l'évaluation de la partie droite de la règle. L'enjeu déterminant de l'application d'une règle consiste à spécifier de façon élégante une sous-collection, c'est-à-dire à développer un langage de spécification de motifs concis et expressif,

2) *stratégie d'application des règles* : une transformation étant composée de plusieurs règles de réécriture, une *stratégie d'application des règles* consiste à choisir quelle règle doit être appliquée. Elle se présente sous la forme d'un algorithme utilisé lors de l'application des règles pour choisir quelle règle doit s'appliquer lorsque plusieurs motifs sont susceptibles de filtrer la même sous-collection,

3) *reconstruction* : en fonction de la liste des couples (sous-collection filtrée, sous-collection calculée), les sous-collections calculées sont réarrangées les unes par rapport aux autres pour construire, de façon cohérente, la collection résultant de l'application de la règle.

Nous décrivons en détail dans cet article les *transformations de patches* (simplement appelées *patches* dans la suite), un type de transformations dédiées à la spécification de modifications topologiques. Nous nous plaçons également dans le cadre de la stratégie d'application des règles *maximale parallèle* : intuitivement, les règles s'appliquent un nombre maximum de fois de façon disjointe. Après l'étape de filtrage, on assure qu'il n'existe pas de sous-collection non filtrée qui puisse être filtrée par l'un des motifs des règles de la transformation. Lorsque deux motifs filtrent une même partie de la collection, une priorité est donnée à l'une des règles. En pratique, l'ordre choisi est l'ordre dans lequel les règles sont spécifiées. Après la sélection des sous-collections à faire évoluer, l'application des règles se fait en parallèle. Cette stratégie trouve une forte motivation dans le cadre de la simulation puisqu'elle supporte l'idée que les sous-parties d'un système évoluent en parallèle et de manière indépendante. Elle a notamment été utilisée dans les systèmes de Lindenmayer, des systèmes de réécriture maximale parallèle de chaînes (Lindenmayer *et al.*, 1992, Rozenberg *et al.*, 1992), qui ont grandement inspiré les développements du projet MGS.

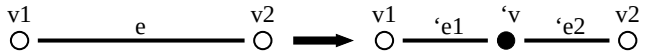


Figure 2. *Insertion d'un sommet sur un arc*

2.2.3. *Transformation de patches*

Les *patches* sont destinés à modifier la structure des collections topologiques. Un patch est un ensemble ordonné de règles de réécriture :

```
patch id = {
  Pattern => Exp ;
  ...
} ;;
```

Le langage de filtre *Pattern* a été introduit pour spécifier les motifs. En revanche, la partie droite des règles est une simple expression du langage, permettant d'effectuer des calculs élaborés suivant la sous-collection filtrée par le motif. MGS propose un ensemble très complet d'opérateurs facilitant la création de nouvelles collections.

Afin d'illustrer la syntaxe des règles de réécriture, nous considérons la modification topologique décrite figure 2 : un arc nommé `e`, dont les faces sont appelées `v1` et `v2`, est filtré et remplacé par deux nouveaux arcs `e1` et `e2` ainsi qu'un nouveau sommet `v` ; l'arc `e1` est bordé par les sommets `v1` et `v`, et l'arc `e2` par les sommets `v2` et `v`. Cette opération permet d'insérer un nouveau sommet sur un arc.

Les motifs

Les motifs décrivent la liste des cellules topologiques à filtrer. Elles sont caractérisées par une dimension, ainsi que la liste partielle de leurs faces et de leurs cofaces. On définit ainsi un ensemble de contraintes sur les cellules topologiques composant la sous-collection filtrée ainsi que sur la relation d'incidence qui les relie les unes aux autres. Voici la grammaire des motifs de patch :

Pattern

m ::= $c \mid c\ o\ m$

Op

o ::= $\varepsilon \mid$ `<` $\mid$ `>`

Clause

c ::= `x:[dim=`exp_d`, faces=`exp_f`, cofaces=`exp_{cf}`,` exp_b`]`
$\mid$ `~x:[dim=`exp_d`, faces=`exp_f`, cofaces=`exp_{cf}`,` exp_b`]`

où ε représente le mot vide. Un motif (*Pattern*) est une liste finie de *clauses* séparées par des opérateurs (*Op*) ; une clause (*Clause*) correspond à un élément à filtrer. Ces

clauses sont caractérisées par plusieurs informations optionnelles qui vont contraindre la recherche d'une sous-collection :

– `x` est une variable de motif qui permet de faire référence à la cellule filtrée par la clause n'importe où dans la règle. Les motifs de patch ne sont pas linéaires. Il est possible d'utiliser une variable qui n'est définie que plus loin dans le motif. Néanmoins, un nom correspond à une et une seule cellule filtrée. Si deux clauses partagent le même identificateur, elles filtrent la même cellule ; les prédicats des deux clauses doivent être vérifiées,

– l'expression exp_d associée au champ `dim` s'évalue en un entier indiquant la dimension de la cellule filtrée par la clause,

– les expressions exp_f et exp_{cf} associées respectivement aux champs `faces` et `cofaces` sont des expressions qui s'évaluent en des séquences de cellules topologiques. On peut utiliser ici les variables de motif ou faire directement référence à des cellules particulières de la structure. Ces séquences contraignent la relation d'incidence que doivent respecter les cellules filtrées. Elles ne sont pas exhaustives ; une cellule peut posséder des (co)faces en plus de celles imposées par le motif,

– une dernière expression optionnelle peut être ajoutée à cette liste. Il s'agit de l'expression exp_b dont l'évaluation doit retourner `true`. Cela permet par exemple d'imposer des propriétés à la valeur associée à la cellule filtrée par la clause,

– pour autoriser certains éléments à être filtrés plusieurs fois (par des occurrences différentes du même motif ou de motifs différents), nous introduisons dans le motif un opérateur unaire « ~ ». Celui-ci signifie que la clause correspond à une cellule répondant toujours aux contraintes structurelles imposées par le motif, mais qui ne sera pas considérée comme filtrée après la recherche de l'instance. Autrement dit, l'élément sera filtré mais *non consommé* par l'étape de filtrage ; il pourra alors être à nouveau sélectionné lors de la recherche des occurrences de filtre suivantes. Les deux règles de consommation sont les suivantes :

1) toute cellule peut être filtrée sans être consommée (clause dont l'identificateur est précédé par l'opérateur ~) ;

2) seules des cellules non consommées peuvent être consommées (clause dont l'identificateur **n**'est **pas** précédé par l'opérateur « ~ »).

Les opérateurs de *`Op`* correspondent à du sucre syntaxique. L'opérateur infixe binaire `<` (resp. `>`) contraint l'élément filtré par son opérande gauche à être une face (resp. une coface) de la cellule filtrée par l'opérande droite. Par exemple, le motif `v:[...] < e:[...]` (qui peut également s'écrire `e:[...] > v:[...]`) signifie que la cellule filtrée par `v` est une face de celle filtrée par `e`. En d'autres termes, ce motif est équivalent à

```
v:[..., cofaces=e] e:[..., faces=v]
```

L'absence d'opérateur entre deux clauses représente l'absence de contrainte sur la relation d'incidence qui lie `v` et `e`. Le motif correspondant à la partie gauche de la règle de la figure 2 s'écrit alors de la manière suivante :

```
~v1 < e:[ dim = 1 ] > ~v2
```

L'opérateur de non consommation est utilisé ici pour ne pas consommer les sommets `v1` et `v2`. En effet, l'opération décrite par la figure 2 décrit la réécriture de l'arc `e` ; les sommets ne sont pas réécrits mais uniquement utilisés pour fournir un *contexte* lors de la définition en partie droite des deux nouveaux arcs.

Partie droite d'une règle

La partie droite d'une règle est une expression. Elle doit spécifier une nouvelle sous-collection qui se raccrochera en lieu et place de la collection filtrée. La partie droite décrit donc un ensemble de cellules topologiques anciennes (correspondant aux cellules filtrées et conservées dans la nouvelle sous-collection) ou nouvelles (correspondant aux cellules nouvellement créées), ainsi que la nouvelle relation d'incidence entre ces cellules. La syntaxe utilisée est la suivante :

Rhs

r ::= $e \mid e\ r$

RhsElement

e ::= `x:[val=`exp_v`]`
| $symb$`:[dim=`exp_d`, faces=`exp_f`, cofaces=`exp_{cf}`, val=`exp_v`]`

Cette spécification (*Rhs* pour *Right-hand-side*) de la sous-collection à réécrire correspond à une liste d'éléments (*RhsElement*) de deux types :

1) les éléments filtrés et conservés en partie droite sont référencés *via* les identificateurs `x` qui ont permis de les nommer en partie gauche de la règle ; cette première construction permet de mettre à jour la valeur qui leur est associée,

2) de nouvelles cellules topologiques peuvent être introduites dans le nouveau complexe cellulaire. Ces éléments n'existent pas avant l'étape de reconstruction. Pour les identifier, nous utilisons donc des symboles ($symb$). Très similaires aux atomes en LISP, les symboles correspondent syntaxiquement à un identificateur précédé d'une apostrophe inversée. Ce symbole peut apparaître ailleurs dans la partie droite : il fait alors référence à la cellule qui sera créée à l'application de la règle. Chaque nouvel élément est caractérisé par sa dimension exp_d, la liste de ses faces exp_f (pouvant faire apparaître à la fois d'anciens éléments à travers l'utilisation des variables de filtre, et des nouveaux éléments en utilisant les symboles), la liste de ses cofaces exp_{cf} et la valeur qui lui est associée exp_v.

En suivant cette syntaxe, la règle schématisée figure 2 s'écrit de la façon suivante en MGS :

```
~v1 < e:[ dim = 1 ] > ~v2 =>
  ‘e1:[ dim = 1,               faces = (^v1,‘v),
        cofaces = cofaces(^e), val = ... ]
  ‘v :[ dim = 0,               faces = seq:(),
        cofaces = (‘e1,‘e2),   val = ... ]
  ‘e2:[ dim = 1,               faces = (^v2,‘v),
        cofaces = cofaces(^e), val = ... ]
```

Les éléments présents en partie droite sont des nouvelles cellules créées dont on réfère à l'aide des symboles ‘v, ‘e1 et ‘e2. On note que les éléments filtrés par v1 et v2 n'étant pas consommés, ils ne sont pas réécrits en partie droite de la règle. Les cofaces de ‘e1 et ‘e2 sont celles de e de telle sorte que si l'arc e borde une cellule de dimension 2, les cellules spécifiées par ‘e1 et ‘e2 bordent également cette cellule après l'application. Notons dans cette règle l'utilisation de l'opérateur de position « ^ » sur les variables de filtre. Le filtrage fait correspondre à ces variables des éléments de la collection, c'est-à-dire des couples (valeur v, position σ). Lorsqu'une variable de filtre est utilisée dans une expression, elle réfère à la valeur v. La position σ est en dénotée à l'aide de l'opérateur de position.

La sémantique formelle des patches n'est pas détaillée dans cet article ; le lecteur intéressé pourra se référer à (Spicher, 2006) pour une sémantique naturelle des patches.

3. Intégration des G-cartes dans MGS

Le formalisme décrit dans la section précédente a donné naissance à un langage de programmation fonctionnel appelé MGS dont nous avons implanté un interprète[2]. Dans cette implantation, nous nous sommes intéressés en particulier à un type de collection topologique : les *quasi-variétés*. Il s'agit d'une classe d'espaces topologiques qu'il est possible de représenter par une structure de données dédiée : les *cartes généralisées* (plus simplement *G-cartes*) (Lienhardt, 1991).

L'implantation de ce type de collections topologiques dans l'interprète MGS est l'objet de plusieurs motivations :

– les quasi-variétés reposent sur la notion de complexe cellulaire que nous utilisons dans la formalisation des collections topologiques les rendant adaptées au point de vue topologique adopté dans MGS ;

– les G-cartes sont une représentation implicite des complexes cellulaires décrivant les quasi-variétés ; la traduction des G-cartes en collections topologiques n'est donc pas immédiate et constitue une bonne validation de la généralité des notions introduites en MGS ;

2. L'interprète MGS, ainsi que l'intégralité de ses sources, est libre de droit et disponible *via* l'adresse suivante : http://mgs.ibisc.univ-evry.fr.

– les G-cartes ont fait l'objet d'une implantation optimisée à travers le développement du modeleur graphique MOKA[3]. Nous proposons une intégration des G-cartes dans le cadre d'un langage déclaratif en se fondant sur la bibliothèque des G-cartes développée pour MOKA.

Nous commençons par présenter brièvement les quasi-variétés et la structure de G-carte qui permet de les représenter. Nous voyons ensuite comment les G-cartes peuvent être considérées du point de vue des collections topologiques dans le langage déclaratif MGS.

3.1. *Définition formelle des G-cartes*

Les quasi-variétés sont une classe d'espaces topologiques particulière qui correspond aux objets topologiques de dimension n obtenus par l'assemblage de n-cellules liées le long de $(n-1)$-cellules. Dans ce cadre, une $(n-1)$-cellule ne peut pas appartenir au bord de plus de deux n-cellules différentes.

Les G-cartes sont un modèle topologique reposant sur la notion de complexe cellulaire abstrait et qui permet de représenter la topologie des quasi-variétés orientables ou non, avec ou sans bord (Lienhardt, 1994). Ce modèle utilise astucieusement la contrainte de bord imposée par la définition des quasi-variétés pour représenter de façon implicite et compacte la relation d'incidence entre les cellules topologiques formant l'espace considéré. Intuitivement, considérons un complexe cellulaire $\mathcal{K}$ de dimension n. On appelle *tuple* une séquence $(\sigma_n, \sigma_{n-1}, \ldots, \sigma_1, \sigma_0)$ de cellules de $\mathcal{K}$ telles que pour tout i, σ_i est une i-cellule et $\sigma_i < \sigma_{i+1}$. Par exemple, dans le complexe de la figure 3, (f_0, e_1, v_2) est un tuple alors que (f_0, e_5, v_2) n'en est pas un (e_5 n'étant incident ni à f_0 ni à v_2). Deux tuples sont dits *i-adjacents* s'ils sont égaux ou s'ils diffèrent uniquement pour la cellule de dimension i. Par exemple, dans la figure 3, (f_0, e_1, v_2) et (f_0, e_2, v_2) sont 1-adjacents. La structure de G-carte correspondant à $\mathcal{K}$ représente alors l'ensemble des tuples de $\mathcal{K}$ ainsi que leurs propriétés d'adjacence à travers des entités abstraites, appelées *brins*, liées les unes aux autres par des applications de l'ensemble des brins dans lui-même.

Définition 4 (G-carte) *Une carte généralisée de dimension $n \geq 0$ est une algèbre $\mathcal{G} = (\mathcal{B}, \alpha_0, \ldots, \alpha_n)$ telle que :*

– $\mathcal{B}$ est un ensemble de brins,

– $\alpha_0, \ldots, \alpha_n$ sont des involutions[4] de $\mathcal{B}$ dans $\mathcal{B}$, et

– $\alpha_i \circ \alpha_j$ est une involution pour $0 \leq i < i+2 \leq j \leq n$.

3. Une présentation du projet MOKA est disponible à l'url suivante : `http://www-sic.univ-poitiers.fr/moka/`

4. Une involution f est une application telle que $f = f^{-1}$.

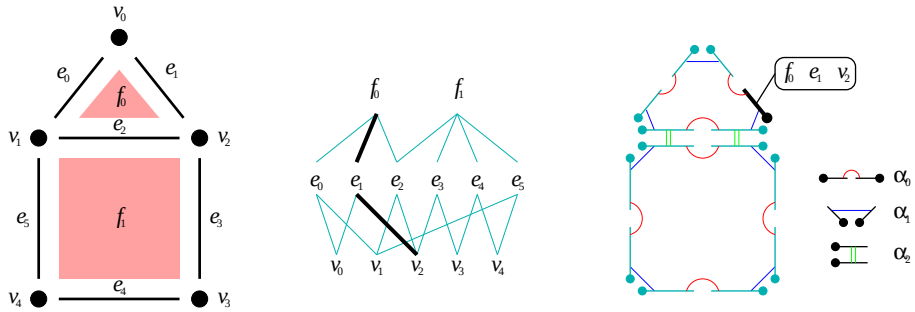


Figure 3. *Complexe cellulaire, graphe d'incidence et G-carte : la figure de gauche décrit un complexe cellulaire de dimension 2. La figure au centre décrit le graphe d'incidence (diagramme de Hasse de l'ordre induit par la relation d'incidence) du complexe cellulaire. La figure de droite décrit la représentation du complexe cellulaire sous forme de G-carte. On note en gras le tuple* (f_0, e_1, v_2) *dont le brin correspondant est mis en évidence dans la G-carte*

Dans la figure 3, le complexe est traduit sous la forme d'une G-carte de dimension 2. Les involutions α_i spécifient la i-adjacence : on vérifie par exemple que les brins correspondants aux tuples (f_0, e_1, v_2) et (f_0, e_2, v_2) sont liés par α_1, les deux tuples étant 1-adjacents. La bijection entre G-cartes et quasi-variétés résulte des propriétés d'involution des α_i et des $\alpha_i \circ \alpha_j$ imposées par la définition des G-cartes (Lienhardt, 1994).

3.2. *G-cartes et collections topologiques*

L'introduction des G-cartes dans l'interprète MGS correspond à la définition d'une nouvelle classe de collections topologiques dénotée `qmf`. Les collections topologiques de type `qmf` sont donc restreintes à des espaces de positions représentés par des complexes cellulaires pouvant être décrits par des G-cartes. Nous proposons d'implanter les collections `qmf` en utilisant la structure de données de G-carte mais en conservant à l'esprit que cette structure de données n'est pas accessible depuis l'interprète où seule la relation d'incidence et les cellules topologiques doivent être manipulées. Tout l'enjeu de cette implantation est donc de rendre opaque l'utilisation des G-cartes.

3.2.1. *Cellule topologique et G-carte*

Afin de réaliser la traduction d'une G-carte $\mathcal{G}$ en le complexe cellulaire $\mathcal{K}$ qu'elle représente, il est nécessaire de retrouver les cellules de $\mathcal{K}$ à partir de $\mathcal{G}$. Intuitivement, une cellule σ de $\mathcal{K}$ est représentée implicitement par l'ensemble des brins dont le tuple associé contient σ. En supposant $b \in \mathcal{B}$ un tel brin, l'ensemble des brins contenant σ est accessible au moyen d'une *orbite*. Une orbite est définie par une séquence d'invo-

lutions $\langle \alpha_{i_1}, \ldots, \alpha_{i_k} \rangle$; soit b un brin de $\mathcal{B}$, on définit de façon inductive l'ensemble $\langle \alpha_{i_1}, \ldots, \alpha_{i_k} \rangle(b)$ des brins accessibles à partir de l'orbite $\langle \alpha_{i_1}, \ldots, \alpha_{i_k} \rangle$ par :

$$\left\{ \begin{array}{l} b \in \langle \alpha_{i_1}, \ldots, \alpha_{i_k} \rangle(b) \\ b' \in \langle \alpha_{i_1}, \ldots, \alpha_{i_k} \rangle(b) \end{array} \right. \quad \Rightarrow \quad \forall\, 1 \leq j \leq k, \quad \alpha_{i_j}(b') \in \langle \alpha_{i_1}, \ldots, \alpha_{i_k} \rangle(b)$$

Ainsi, l'ensemble des brins représentant une cellule σ de dimension i est donné par l'orbite $\langle \alpha_0, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n \rangle$ et un des brins b dont le tuple associé contient σ. Cette orbite permet d'emprunter toutes les involutions excepté α_i pour parcourir les brins accessibles à partir de b. En interdisant α_i, tous les brins considérés correspondent à des tuples possédant la même cellule de dimension i, à savoir σ. Dans la figure 3, l'orbite $\langle \alpha_1, \alpha_2 \rangle$ à partir du brin (f_0, e_1, v_2) parcour les brins (f_0, e_2, v_2), (f_1, e_2, v_2) et (f_1, e_3, v_2). Ces tuples sont les seuls à contenir v_2. En revanche, le parcours de l'orbite $\langle \alpha_0, \alpha_2 \rangle$ à partir de (f_0, e_1, v_2) ne retourne que (f_0, e_1, v_0), l'unique autre brin contenant e_1. Ainsi, un couple $(b, i) \in \mathcal{B} \times \mathbb{N}$ permet d'identifier la cellule σ de dimension i telle que le tuple associé à b contient σ.

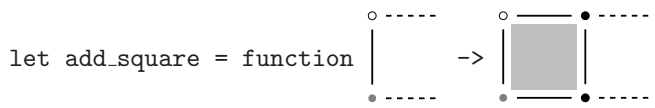
Nous utilisons ce couple pour représenter en mémoire les cellules topologiques des collections `qmf` rendant opaque l'utilisation des G-cartes. Le programmeur MGS a alors accès à ces cellules à l'aide d'un type abstrait `ncell` fournit par l'interprète. De façon interne, nous avons interfacé la bibliothèque de gestion des G-cartes de MOKA avec notre implantation OCaml de l'interprète. Les brins sont accessibles en OCaml *via* un type abstrait `dart` (voir les *custom blocks* dans (Leroy *et al.*, 2004, page 254)) que nous avons défini ; les cellules topologiques sont alors simplement représentées par un couple de type `dart * int`.

3.2.2. *Des G-cartes dans un langage déclaratif*

La présence des objets de type `ncell` n'est pas suffisante : l'interprète fournit également un certain nombre de primitives permettant de spécifier des complexes cellulaires complets, c'est-à-dire de définir la relation d'incidence entre les cellules topologiques.

La construction des complexes est similaire à la paramétrisation d'objets géométriques par le λ-calcul proposée dans (Dufourd *et al.*, 2002). Il y est présenté une intégration des G-cartes dans un λ-calcul typé qui facilite énormément la composition des objets géométriques. Les G-cartes et leurs opérations deviennent alors des expressions fonctionnelles.

Nous nous plaçons dans un contexte où le point de vue « brins et involutions » est caché pour ne laisser apparaître que les cellules topologiques. Les primitives proposées dans l'interprète sont, dans ce contexte, de plus haut niveau que celles de (Dufourd *et al.*, 2002). Il est alors possible d'utiliser la puissance d'expression d'un langage fonctionnel pour construire de nouveaux objets (par notamment la composition de fonctions et la récursivité). Dans l'exemple suivant, on propose de construire une ligne de n carrés (vus comme des 2-cellules à 4 côtés). Le programme suivant schématise comment faire :



La fonction `add_square` prend en argument un complexe cellulaire (encodé dans une G-carte) référencé par un arc (les cercles gris et vides représentent le bord de cet arc, les lignes pointillées représentent la partie restante du complexe cellulaire). Elle construit un nouveau complexe identique au précédent (la partie du complexe figurant en pointillés n'est pas modifiée) auquel est ajouté un nouveau carré construit à partir de l'arc passé en paramètre. Elle retourne alors un des côtés de ce nouveau complexe à partir duquel un nouvel appel à la fonction construira un autre carré. La fonction `line` utilise ce principe en appelant de façon récursive `add_square` pour construire une ligne de n carrés à partir d'un arc distingué originel (pour $n = 0$). Nous ne détaillons pas les primitives de l'interprète concernant la construction de G-cartes ; nous invitons toutefois le lecteur intéressé à parcourir (Spicher, 2006).

3.2.3. *Les collections topologiques `qmf`*

Les collections topologiques de type `qmf` sont simplement des tables d'associations applicatives attachant une valeur quelconque du langage aux cellules topologiques de type `ncell`.

4. Le raffinement de maillage

Dans cette section, nous proposons d'utiliser la programmation par règles offerte par les patches pour spécifier un algorithme spécifique à la CAO : le raffinement de maillage. En appliquant ces transformations sur les collections `qmf`, nous cherchons à montrer que le formalisme développé dans le projet MGS permet d'implanter de façon élégante et expressive des algorithmes non triviaux sur des structures de données complexes.

La définition et la génération de courbes et de surfaces lisses spécifiées à partir d'un petit nombre de points de contrôle est un problème fondamental en modélisation géométrique. Une approche possible se fonde sur l'idée de *subdivision* qui consiste à remplacer itérativement une représentation grossière par une représentation plus fine. Introduits par Chaikin en 1974 (Chaikin, 1974) les algorithmes de subdivision pour les courbes et les surfaces se sont depuis multipliés. Si ces algorithmes peuvent se décrire de manière très intuitive par des opérations locales agissant sur un point et ses voisins, leur formulation et leur implantation sont souvent compliquées par les notations indexées (vecteurs et tableaux) habituellement utilisées dans les programmes et la réindexation impliquée par les changements topologiques. Cette complication

justifie le développement d'approches implicites, *i.e.*, qui ne font pas référence à des séquences indexées de points. Cependant, la définition d'un cadre à la fois déclaratif et implicite n'a réellement abouti que pour la subdivision de courbes (Prusinkiewicz *et al.*, 2003).

Ce type d'algorithme est donc un bon test pour valider l'expressivité de MGS. L'exemple développé ci-dessous illustre l'utilisation des patches pour la spécification d'opérations topologiques sur des maillages de surfaces plongées en dimension 3. L'approche est complètement déclarative (les modifications topologiques sont spécifiés par des règles locales qui correspondent à la description informelle de l'algorithme) ce qui répond par la positive à une question posée dans (Smith *et al.*, 2004) : est-il possible de programmer de manière déclarative les opérations de subdivision d'un maillage au-delà de la dimension 1.

4.1. *La subdivision de Loop*

Les algorithmes de subdivision génèrent à la limite des surfaces lisses en itérant des subdivisions de maillages polygonaux. Dans (Zorin, 2000), on trouve une description détaillée des processus de subdivision utilisés pour la modélisation et l'animation. Les algorithmes de subdivision sont spécifiés localement à l'aide de *masques* ; il s'agit de complexes cellulaires décrivant une partie d'un maillage centrée sur un élément à raffiner (un arc ou une face) pour lequel un nouveau sommet est créé. Les coordonnées de ce sommet sont déterminées par une combinaison affine des positions des sommets apparaissant dans le masque. Les propriétés de continuité de la surface obtenue en itérant jusqu'à la limite l'algorithme de subdivision, diffèrent selon le masque utilisé. La qualité du masque dépend de ces propriétés : on cherche à construire des surfaces aussi lisses que possible (C^1-continues ou C^2-continues partout par exemple). Il est difficile de réaliser cette propriété sur des maillages arbitraires, ceux-ci présentant des irrégularités situées en des points singuliers. Un masque est donc choisi suivant le type de maillage ou de propriétés à vérifier.

Nous nous intéressons à la subdivision de Loop (Loop, 1987). Cet algorithme de raffinement :

– fonctionne par *insertion de sommets* : à chaque application de l'algorithme un sommet est inséré sur chaque arc du maillage pour le diviser en deux arcs ; les anciens sont conservés, et les nouveaux sommets sont liés entre eux par de nouveaux arcs ;

– s'applique sur et génère des *maillages triangulaires* ;

– est *approximant* : les anciens sommets voient leur position dans l'espace modifiée et les sommets insérés sont placés en fonction de la position des anciens sommets dans le maillage initial (les subdivisions approximantes s'opposent aux subdivisions *interpolantes* qui n'affectent pas la position des anciens sommets).

L'algorithme est décrit d'une part, par la modification du maillage qu'il génère et d'autre part, par le placement géométrique des sommets du maillage.

Modification topologique. La modification du maillage est fondée sur la subdivision polyédrique présentée figure 4. Un sommet est inséré sur chaque arc ; les triangles sont raffinés en 4 triangles plus petits.

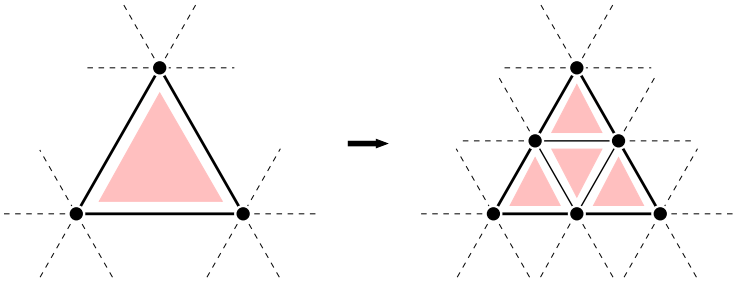


Figure 4. *Algorithme de Loop : modification topologique*

Placement géométrique. Les masques de Loop fournissent les informations nécessaires pour positionner les nouveaux sommets en fonction des coordonnées des anciens, et pour déplacer les anciens sommets. Ils sont décrits figure 5. A gauche, les coordonnées du nouveau sommet v inséré sur l'arc bordé par v_1 et v_2 (figuré en gras sur l'image) sont données par la somme pondérée des coordonnées de sommets voisins suivante :

$$\frac{3}{8}v_1 + \frac{3}{8}v_2 + \frac{1}{8}v_3 + \frac{1}{8}v_4$$

A droite, les coordonnées d'un ancien sommet v du maillage sont recalculées en fonction de la position de ses sommets 1-**voisins** dans le maillage initial par :

$$(1 - \mathrm{k}\beta)v + \sum \beta v_i \quad \text{avec } \beta = \frac{1}{k}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{k}\right)^2\right)$$

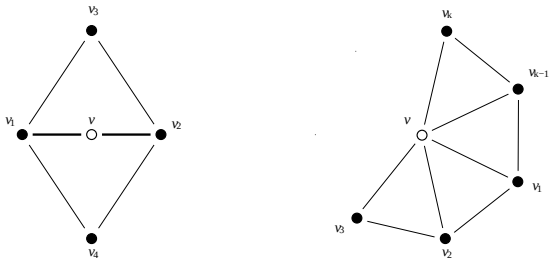


Figure 5. *Algorithme de Loop : modification géométrique*

4.2. *Représentation des objets géométriques en MGS*

Les objets géométriques sont représentés en MGS par les collections topologiques fondées sur la notion de G-carte que nous avons vues précédemment.

Nous souhaitons représenter des surfaces composées de polygones. Il s'agit donc de complexes cellulaires de dimension 2. Les 2-cellules (appelées également *faces*) sont décorées par le symbole `'face`. Les arcs (1-cellules) sont décorés par le symbole `'edge` ; on remarque en particulier que la manipulation de solides impose que chaque arc soit bordé par deux faces (les arcs situés aux bords sont incidents à une seule face ; nous ne les considérons pas dans notre implantation).

Les sommets portent une information plus structurée, représentée par l'enregistrement MGS (équivalent à une structure en C) suivant :

```
record vertex = { x:float, y:float, z:float, n:int } ;;
```

dont les champs `x`, `y` et `z` codent les coordonnées du sommet qu'ils décorent. L'entier `n` encode la génération à laquelle le nœud a été créé. En effet, il nous faut distinguer les anciens et les nouveaux sommets. On utilise alors ce champ ; nous supposons pour cela l'existence d'un compteur `gen` contenant l'entier correspondant à la génération courante. Un ancien sommet aura un champ `n` strictement inférieur à `gen`. On prendra soin d'incrémenter le compteur `gen` après chaque application de l'algorithme de subdivision.

Pour simplifier la description des programmes MGS, nous supposons l'existence d'une fonction `addCoord` sommant des coordonnées de deux points, et d'une variable globale `O` correspondant à l'origine.

4.3. *Subdivision de Loop en MGS*

L'implantation de l'algorithme de Loop est faite en trois étapes.

1) La sauvegarde des anciennes coordonnées : les sommes pondérées font référence aux positions des sommets dans le maillage initial. Nous programmons le déplacement des sommets suivant le masque de Loop de droite sur la figure 5, mais nous sauvegardons les anciennes coordonnées pour le calcul de la position des nouveaux sommets.

```
patch even_vertex = {
  v:[dim=0] =>
    let s = ccellsfold(addCoord, O, v, 1) and b = β in
      v:[val={ ox = v.x, x = (1-k*b)*v.x + beta*s.x, ...}]
};;
```

Ce patch est constitué d'une seule règle filtrant un sommet `v` pour mettre à jour sa décoration. Les variables `s` et `b` correspondent respectivement à la somme des coor-

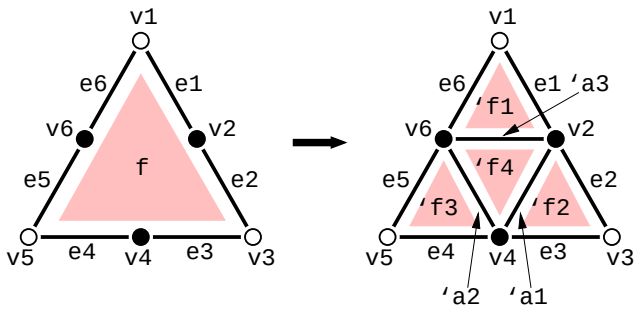


Figure 6. *Création des 4 triangles raffinés à partir d'un hexagone dans la subdivision de Loop*

données des sommets 1-voisins de `v` et au coefficient β du masque de Loop. La primitive `ccellsfold(`f`,`z`,`σ`,`i`)` récupère la liste $(\sigma_1, \ldots, \sigma_n)$ des i-voisines de σ et calcule la valeur $f(\sigma_1, (\ldots f(\sigma_n, z)\ldots))$. Les anciennes coordonnées sont conservées dans les champs `ox`, `oy` et `oz` alors que les champs `x`, `y` et `z` sont mis à jour.

2) L'insertion des nouveaux sommets :

```
patch odd_vertex = {
  ~v1 < e:[dim=1] > ~v2
  ~v1 < ~e13 > ~v3 < ~e23 > ~v2
  ~v1 < ~e14 > ~v4 < ~e24 > ~v2
  =>
    'v:[ dim = 0, cofaces = ('e1,'e2), val = v ]
    'e1:[ dim = 1, ..., val = 'edge ]
    'e2:[ dim = 1, ..., val = 'edge ]
} ;;
```

Ce patch est proche du patch présenté en exemple section 2.2.3. Le motif est étendu pour prendre en compte les sommets `v3` et `v4` qui apparaissent dans le masque de Loop à gauche figure 5. Seul l'arc `e` est consommé pour être supprimé et remplacé. Les autres éléments filtrés servant de contexte, ils ne sont pas consommés.

3) La création des triangles raffinés : le patch précédent divisant chaque arc en deux, les triangles du maillage initial sont transformés en hexagone. Les 2-cellules sont alors détruites pour être substituées par les nouveaux triangles raffinés.

```
patch subdivideFace = {
  f:[ dim = 2, faces = (^e1,^e2,^e3,^e4,^e5,^e6) ]
  ~v1 < ~e1 > ~v2:[ v2.n == gen ] < ~e2 >
  ~v3 < ~e3 > ~v4:[ v4.n == gen ] < ~e4 >
  ~v5 < ~e5 > ~v6:[ v6.n == gen ] < ~e6 > ~v1
  =>
    'a1:[ dim = 1, faces = (^v2,^v4), val = 'edge ]
```

```
    ‘a2:[ dim = 1, faces = (^v4,^v6), val = ‘edge ]
    ‘a3:[ dim = 1, faces = (^v6,^v2), val = ‘edge ]

    ‘f1:[ dim = 2, faces = (‘a1,^e2,^e3), val = ‘face ]
    ‘f2:[ dim = 2, faces = (‘a2,^e4,^e5), val = ‘face ]
    ‘f3:[ dim = 2, faces = (‘a3,^e6,^e1), val = ‘face ]
    ‘f4:[ dim = 2, faces = (‘a1,‘a2,‘a3), val = ‘face ]
}
```

Ce patch est composé d'une seule règle qui est schématisée figure 6. On note la présence des tests vi`.n == gen` pour différencier les anciens des nouveaux sommets.

La figure 7 présente les résultats d'applications de l'algorithme de Loop générés par l'interprète MGS.

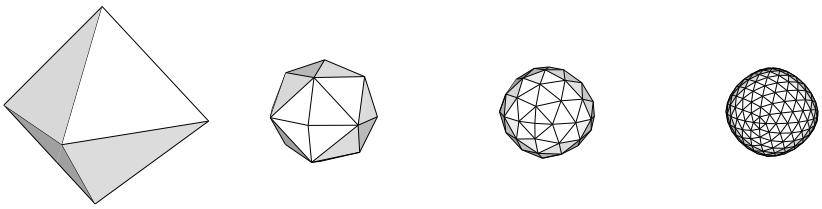


Figure 7. *Résultats de l'application de l'algorithme de Loop sur un diamant. De gauche à droite, l'état initial puis 3 itérations de l'algorithme. Ces images ont été générées par l'interprète MGS*

5. Conclusion

Dans cet article, nous avons introduit des concepts permettant la représentation et la manipulation de structures topologiques arbitrairement complexes. Les *collections topologiques* sont des nouvelles structures de données correspondant à des champs de données indexés par des espaces topologiques discrets. Ces espaces sont représentés par des *complexes cellulaires abstraits*, un outil mathématique issu de la *topologie algébrique*. Pour manipuler de façon concise et expressive de telles structures de données, les *transformations* développent le concept de *réécriture locale* de collections topologiques. Il s'agit d'une extension des définitions par cas de fonction : une transformation permet de spécifier une modification locale d'une collection topologique à l'aide de règles de réécriture dont les motifs sont construits à partir de la relation d'incidence des complexes cellulaires. Nous avons présenté en détails une classe particulière de transformations, les *patches*, fondées sur un langage de spécification de motifs original et autorisant la modification de la structure des complexes cellulaires.

Ces concepts ont été intégrés dans un langage fonctionnel. Bien que la notion de collection topologique soit indépendante d'un paradigme de programmation (qu'il soit impératif, objet, fonctionnel, ...), la prise en compte des transformations dans un cadre fonctionnel est naturelle et bien adaptée : la réécriture locale des éléments de la collection topologique est en accord avec le paradigme fonctionnel qui procède par calcul d'une nouvelle valeur résultat, en fonction d'une valeur argument, sans effet de bord.

Afin d'illustrer nos propos, nous nous sommes penchés sur une application des collections et des transformations dans le cadre de la CAO. Tout d'abord, nous avons cherché à spécifier des espaces topologiques appelés *quasi-variétés* à l'aide d'un type particulier de collections topologiques. Les quasi-variétés peuvent être représentées par des *G-cartes*, un modèle topologique. Les G-cartes reposent sur la notion de complexe cellulaire et sont en cela adaptées au point de vue topologique que nous adoptons avec les collections. Malgré la représentation implicite de la structure qu'elles offrent, nous avons utilisé les G-cartes pour l'implantation de cette classe d'espaces. Finalement, nous avons illustré l'utilisation des transformations pour spécifier *par règles* une opération largement utilisée en CAO : le raffinement de maillage. Avec cette implantation, nous montrons qu'il est possible de décrire au moyen de règles de réécriture des processus topologiques complexes de façon déclarative, expressive et concise.

Bien que nous nous soyons attachés à des problématiques touchant le domaine de la CAO, les concepts développés dans le projet MGS peuvent être appliqués dans beaucoup d'autres domaines. Nous les avons par exemple utilisés pour spécifier la propagation d'une onde dans un milieu amorphe (Rauch, 2003), l'auto-assemblage par accrétion ou par découpage de fractales (Giavitto *et al.,* 2006), l'évolution de la densité de particules baignant dans un fluide dans un milieu arbitrairement complexe (Egli *et al.,* 2004), ou encore dans la représentation de connaissances à l'aide de la q-analyse (Valencia *et al.,* 1998) pour revenir aux exemples concernant le calcul spatial présentés dans l'introduction de cet article. Ces travaux nous ont également conduits à l'utilisation de la réécriture pour la modélisation et la simulation de systèmes complexes. L'état du système est décrit par une collection topologique, et sa dynamique est spécifiée à l'aide des transformations. Nous avons en particulier développé un modèle pour simuler le processus de neurulation (Spicher *et al.,* 2005), phénomène important de l'embryogénèse. Le langage MGS a également été utilisé dans la modélisation de la croissance du méristème apical de l'arabette (Reuille *et al.,* 2006).

Les travaux futurs portent sur la compilation des étapes de filtrages et de reconstruction. Par exemple, l'implantation actuelle des patches sur les G-cartes, notamment pour la reconstruction, consiste à transformer la G-carte en un complexe cellulaire abstrait (dont le pouvoir d'expressivité inclut celui des G-cartes). La structure ainsi transformée est facilement manipulable mais on perd les avantages algorithmiques et les performances d'une structure de données telle que les G-cartes. De plus, la transformation d'une structure en une autre est extrêmement coûteuse. Une optimisation possible consisterait à traduire un motif filtrant des cellules topologiques en un motif équivalent filtrant directement un ensemble de brins.

Plus généralement, du fait de sa richesse, l'algorithme de filtrage des motifs des patches est déterminant pour les performances d'un programme MGS. En effet, trouver une instance d'un motif consiste en l'énumération de l'espace des sous-collections vérifiant les contraintes imposées par le motif. Plusieurs voies peuvent êtres suivies pour optimiser cette recherche. Les deux techniques suivantes sont envisagées :

1) la progression dans l'espace des sous-collections se faisant au rythme des échecs successifs de la recherche des instances, il est essentiel d'énumérer les éléments du motif de telle sorte que l'échec se produise au plus tôt. Commencer le filtrage par les éléments les plus contraints est donc une optimisation possible,

2) les motifs peuvent présenter des éléments de symétries. Celles-ci peuvent être utilisées pour réduire l'espace de recherche et réutiliser des résultats intermédiaires. Elles peuvent amener plus généralement à développer une algèbre de filtre fournissant une aide à la compilation (par exemple pour l'équivalence entre filtres).

Nous travaillons également sur la définition d'une sémantique formelle décrivant les collections et les transformations. Ces recherches sont particulièrement orientées sur la définition d'une sémantique des transformations fondée sur une analogie entre les homomorphismes de structures de données et les formes différentielles de la géométrie algébrique. En effet, les collections topologiques peuvent être vues comme des *chaînes topologiques* (Munkres, 1984, Giavitto *et al.*, 2002, Spicher, 2006). Elles permettent à la fois d'associer des valeurs aux cellules topologiques mais elles fournissent également une structure de groupe facilitant la manipulation de ces espaces. Dans ce cadre de la topologie algébrique, les homomorphismes de chaînes sont appelés *cochaînes* et sont utilisés pour définir des formes différentielles sur des espaces discrets. L'objectif de ce rapprochement entre collections topologiques et formes différentielles est de définir un ensemble minimal d'opérateurs, et l'algèbre associée, sur les transformations. Cette algèbre permettra dans le cadre de la simulation de systèmes complexes, l'expression générique de schémas d'interactions indépendants de la topologie du système. On pense par exemple à la généralisation de l'opérateur Laplacien rencontré en physique pour la modélisation de processus de diffusion. Sa généricité permet de combiner des modèles de diffusion discrets *et* continus. Cette perspective conduit naturellement à la mise en œuvre de simulations hybrides (temps et espace continus/discrets) ou multimodèles.

6. Bibliographie

Abelson H., Allen D., Coore D., Hanson C., Homsy G., Knight T. F., Nagpal R., Rauch E., Sussman G. J., Weiss R., « Amorphous Computing », *CACM : Communications of the ACM*, 2000.

Beal J., « Amorphous Medium Language », *Large-Scale Multi-Agent Systems Workshop - AAMAS 2005*, 2005.

Chaikin G., « An algorithm for high speed curve generation », *Computer Graphics and Image Processing*, vol. 3, p. 346-349, 1974.

Coore D., Botanical Computing : A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer, PhD thesis, Massachusetts Institute of Technology, 1991.

DeHon A., Giavitto J.-L., Gruau F., « Computing Media and Languages for Space-Oriented Computation », Seminar, 2006.

Desbrun M., Kanso E., Tong Y., « Discrete ifferential forms for computational modeling », *Discrete differential geometry : an applied introduction*, Schröder, SIGGRAPH'06 course notes, p. 39-54, 2006.

Dufourd J.-F., Luther S., « Parametrizing geometric objects using λ-calculus », *Proceedings of the 18th spring conference on Computer graphics*, ACM Press, p. 185-194, 2002.

Egli R., Stewart N. F., « Chain models in computer simulation », *Math. Comput. Simul.*, vol. 66, n° 6, p. 449-468, 2004.

Giavitto J.-L., Michel O., « The Topological Structures of Membrane Computing », *Fundamenta Informaticae*, vol. 49, p. 107-129, 2002.

Giavitto J.-L., Spicher A., *Systems Self-Assembly : multidisciplinary snapshots*, Elsevier, chapter Simulation of self-assembly processes using abstract reduction systems, 2006.

Goldstein S. C., Campbell J., Mowry T. C., « Programmable Matter », *IEEE Computer*, vol. 38, n° 6, p. 99-101, 2005.

Hillis W. D., Steele G. L., « Data Parallel Algorithms », *Communications of the ACM*, vol. 29, n° 12, p. 1170-1183, December, 1986.

Hirani A. N., Discrete exterior calculus, PhD thesis, California Institute of Technology, 2003.

Klavins E., « Automatic Synthesis of Controllers for Distributed Assembly and Formation Forming », *ICRA*, IEEE, p. 3296-3302, 2002.

Leroy X., Doligez D., Garrigue J., Rémy D., Vouillon J., *The Objective Caml system, Documentation and user's manual*, release 3.09 edn. 2004, `http://caml.inria.fr/pub/docs/manual-ocaml/index.html`.

Lienhardt P., « Topological models for boundary representation : a comparison with n-dimensional generalized maps », *Computer-Aided Design*, vol. 23, n° 1, p. 59-82, 1991.

Lienhardt P., « N-Dimensional Generalized Combinatorial Maps and Cellular Quasi-Manifolds », *Journal on Computational Geometry and Applications*, vol. 4, n° 3, p. 275-324, 1994.

Lindenmayer A., Jürgensen H., « Grammars of development : discrete-state models for growth, differentiation, and gene expression in modular organisms », G. Rozenberg, A. Salomaa (eds), *Lindenmayer Systems, Impacts on Theoretical Computer Science, Computer Graphics and Developmental Biology*, Springer Verlag, p. 3-21, February, 1992.

Lisper B., « On the relation between functional and data-parallel programming languages », *Proc. of the 6th. Int. Conf. on Functional Languages and Computer Architectures*, ACM, ACM Press, June, 1993.

Loop T. L., *« Smooth subdivision surfaces based on triangle »*, Master's thesis, University of Utah, August, 1987.

Munkres J., *Elements of Algebraic Topology*, Addison-Wesley, 1984.

Nagpal R., Programmable Self-Assembly : Constructing Global Shape using Biologically-inspired Local Interactions and Origami Mathematics, PhD thesis, Massachusetts Institute of Technology, 2001.

Prusinkiewicz P., Samavati F. F., Smith C., Karwowski R., « L-System Description of Subdivision Curves », *International Journal of Shape Modeling*, vol. 9, n° 1, p. 41-59, 2003.

Rauch E., « Discrete, Amorphous Physical Models », *International Journal of Theoretical Physics*, vol. 42, n° 2, p. 329-348, Februray, 2003.

Reuille P. B. (de), Bohn-Courseau I., Ljung K., Morin H., Carraro N., Godin C., Traas J., « Computer simulations reveal properties of the cell-cell signaling network at the shoot ape x in Arabidopsis », *PNAS*, vol. 103, n° 5, p. 1627-1632, 2006.

Rothemund P. W. K., Papadakis N., Winfree E., « Algorithmic self-assembly of DNA Sierpinski triangles », *PLoS Biol*, vol. 2, n° 12, e424, 2004. www.plosbiology.org.

Rozenberg G., Salomaa A., *Lindenmayer Systems*, Springer, Berlin, 1992.

Smith C., Prusinkiewicz P., Samavati F., « Local specification of surface subdivision algorithms », *Applications of Graph Transformations with Industrial Relevance (AGTIVE 2003)*, vol. 3062 of *LNCS*, Springer, p. 313-327, 2004.

Spicher A., « Représentation et manipulation de structures topologiques dans un langage fonctionnel », O. Michel (ed.), *Journées Francophones des Langages Applicatifs (JFLA'2005)*, INRIA, p. 113-128, 2005.

Spicher A., Transformation de collections topologiques de dimension arbitraire. Application à la modélisation de systèmes dynamiques., PhD thesis, Université d'Évry, 2006.

Spicher A., Michel O., « Declarative modeling of a neurulation-like process », *Sixth International Workshop on Information Processing in Cells and Tissues (IPCAT'05)*, York, p. 304-317, August, 2005. revised and extended version to be published in Biosystems.

Valencia E., Giavitto J.-L., « Algebraic Topology for Knowledge Representation in Analogy Solving. », *European Conference on Artificial Intelligence (ECAI'98)*, Brighton, p. 23-28, August, 1998.

Yip K., Zhao F., « Spatial aggregation : Theory and applications », *Journal of Artificial Intelligence Research*, vol. 5, p. 1-26, august, 1996.

Zorin D., « Subdivision zoo », *Subdivision for modeling and animation*, Schröder, Peter and Zorin, Denis, p. 65-104, 2000.

Article reçu le 17 mai 2006
Accepté après révisions le 15 février 2007

Antoine Spicher *est actuellement post-doctorant au Laboratoire Lorrain de Recherche en Informatique et ses Applications (LORIA) dans l'équipe MAchine Intelligente Autonome (MAIA) où il cherche à analyser la notion d'espace dans les systèmes multi-agents et les automates cellulaires, et ses applications en biologie synthétique. Ses travaux font suite à une thèse effectuée au laboratoire Informatique, Biologie Intégrative et Systèmes complexes (IBISC) de l'université d'Évry qu'il a soutenue en 2006.*

Olivier Michel *est actuellement maître de conférences à l'université d'Evry et membre du laboratoire Informatique, Biologie Intégrative et Systèmes complexes (IBISC). Ses activités de recherche concernent l'étude et le développement de langages de programmation non conven-*

tionnels. Ces langages intègrent à chaque fois des structures de données et de contrôle originales pour faciliter le développement et la simulation d'une classe particulière de systèmes dynamiques : les systèmes dynamiques à structure dynamique. Ces systèmes sont très fréquents en biologie où la structure de l'espace des phases ne peut être fixée a priori *car elle dépend de l'évolution du système.*

ANNEXE POUR LE SERVICE FABRICATION

A FOURNIR PAR LES AUTEURS AVEC UN EXEMPLAIRE PAPIER
DE LEUR ARTICLE ET LE COPYRIGHT SIGNE PAR COURRIER
LE FICHIER PDF CORRESPONDANT SERA ENVOYE PAR E-MAIL

1. ARTICLE POUR LA REVUE :
   RSTI - TSI – 26/2007. Langages applicatifs

2. AUTEURS :
   Antoine SPICHER — *Olivier* MICHEL

3. TITRE DE L'ARTICLE :
   Représentation et manipulation de structures topologiques dans un langage fonctionnel

4. TITRE ABRÉGÉ POUR LE HAUT DE PAGE MOINS DE 40 SIGNES :
   Réécriture de collections topologiques

5. DATE DE CETTE VERSION :
   29 novembre 2007

6. COORDONNÉES DES AUTEURS :

   – adresse postale :
   IBISC, FRE 2873 CNRS, Université d'Évry
   Tour Evry-2, 523 Place des Terrasses de l'Agora
   91000 Évry, France
   – téléphone : 01 60 87 39 00
   – télécopie : 01 60 87 37 89
   – e-mail : aspicher@ibisc.univ-evry.fr

7. LOGICIEL UTILISÉ POUR LA PRÉPARATION DE CET ARTICLE :
   LaTeX, avec le fichier de style `article-hermes.cls`,
   version 1.2 du 17 May 1995.

8. FORMULAIRE DE COPYRIGHT :
   Retourner le formulaire de copyright signé par les auteurs, téléchargé sur :
   `http://www.revuesonline.com`

SERVICE ÉDITORIAL – HERMES-LAVOISIER
14 rue de Provigny, F-94236 Cachan cedex
Tél : 01-47-40-67-67
E-mail : revues@lavoisier.fr
Serveur web : http://www.revuesonline.com