Declarative Mesh Subdivision Using Topological Rewriting in MGS

Antoine Spicher[1], Olivier Michel[1], and Jean-Louis Giavitto[2]

[1] Université Paris-Est Créteil, LACL, 61 rue du Général de Gaulle, F-94010 Créteil, France
antoine.spicher@u-pec.fr, olivier.michel@u-pec.fr

[2] CNRS - Université d'Évry, Laboratoire IBISC, 523 place des terrasses de l'Agora, F-91000 Évry, France
giavitto@ibisc.univ-evry.fr

Abstract. Mesh subdivision algorithms are usually specified informally using graphical schemes defining local mesh refinements. These algorithms are then implemented efficiently in an imperative framework. The implementation is cumbersome and implies some tricky indices management. Smith et al. (2004) asks the question of the declarative programming of such algorithms in an index-free way. In this paper, we positively answer this question by presenting a rewriting framework where mesh refinements are described by simple rules. This framework is based on a notion of topological chain rewriting. Topological chains generalize the notion of labeled graph to higher dimensional objects. This framework has been implemented in the domain specific language MGS. The same generic approach has been used to implement Loop as well as Butterfly, Catmull-Clark and Kobbelt subdivision schemes.

1 Introduction

The definition and generation of smooth curves or surfaces specified from a finite and small set of control points is a fundamental problem in geometrical modeling. A possible approach is based on the concept of *mesh subdivision* which consists in iterating the replacement of coarse parts of a mesh by finer ones. Introduced by Chaikin in 1974 [1], subdivision algorithms for curves and surfaces are now a major geometric modeling technique.

Many polygon mesh algorithms operate in a local manner and are intuitively described by local graphical schemes. However, their expression and implementation rely on the use of global arrays of points with the induced indexed notation. This obscures the essence of these algorithms and makes their specification unnecessarily complex, especially for the inevitable reindexations caused by the mesh modifications.

This issue has motivated several attempts to develop implementations of such algorithms that are simultaneously *declarative* (close to the mathematical formulation), *intensional* (abstracting meshes from their implementations) and *coordinate-free* (no explicit index manipulation). These works have succeeded

H. Ehrig et al. (Eds.): ICGT 2010, LNCS 6372, pp. 298–313, 2010.
© Springer-Verlag Berlin Heidelberg 2010

completely in the 1D case [2] and only partially for the 2D case [3]: "The problem of providing a declarative, grammar-like method for specifying subdivision algorithms remains open. Such specification, if possible, may provide the ultimately concise and clear specification of these algorithms".

In [4,5], we have developed a topological collections rewriting formalism for simulation purposes. Topological collections extend the notion of labeled graphs considering higher dimensional cells (w.r.t. points and edges) such as surfaces, volumes, etc. This framework has been validated in the modeling and simulation of morphogenesis [6,7] and self-assembly [8] as well as diagrammatic reasoning involving arbitrary high dimensional spaces [9].

In this paper, we show that the topological collections and the topological rewriting framework suit well the declarative, intensional and coordinate-free specification of mesh subdivision algorithms. In the next section, we present the notion of topological collection and we give a formal definition of topological rewriting. In Section 3 we introduce MGS, an experimental programming language that provides an implementation of the previous concepts. MGS is used in Section 4 to specify the Loop's algorithm, a typical subdivision scheme. The other classical algorithms are illustrated and their complete implementations are given in [10]. We conclude by outlining some links with more traditional approaches in graph rewriting as well as some related and future work.

2 Topological Rewriting

2.1 Topological Collections

A *topological collection* is a weakening of the notion of *topological chain*. The latter is developed in algebraic topology and corresponds to a labeled *cellular complex*. An (abstract) cellular complex is a formal construction that builds a space in a combinatorial way through more simple objects called *topological cells*. Each cell abstractly represents a part of the whole space. The structure of the whole space, corresponding to the partition into topological cells, is considered through the *incidence relationships*, relating two "neighbor" cells in the partition. A topological chain is a function from a cellular complex to a set of labels equipped with some algebraic structure [11]. We can forget some of the technical machinery for topological collection.

Definition 1 (Abstract Cellular Complex). *Let* $(\mathsf{S}_n)_{n\in\mathbb{N}}$ *be a family of disjoint sets of symbols. An element of* S_n *is called an* abstract topological cell of dimension n*, or simply an* n-cell. *If* $\sigma \in \mathsf{S}_n$*,* n *is called the* dimension *of* σ *and is denoted by* $dim(\sigma)$*. We write* S *for the set of cells* $\bigcup_n \mathsf{S}_n$*.*

An abstract cellular complex $\mathcal{K}$ *on* S *is a partially ordered subset of* S*, that is a couple* $(S, \preceq)$ *such that* $S \subset \mathsf{S}$ *and* $\preceq$ *is a partial order over* S *(i.e., a reflexive, transitive and antisymmetric binary relation on* S*) satisfying the following condition:* $\forall \sigma, \tau \in S \quad \sigma \preceq \tau \Rightarrow dim(\sigma) \leq dim(\tau)$*. The relation* $\preceq$ *is called the* incidence relationships *of the complex* $\mathcal{K}$*. A complex* $\mathcal{K}$ *is of finite dimension if the integer* $N = \max\{dim(\sigma) \,|\, \sigma \in S\}$ *is defined. In such a case,* N *is called the* dimension *of* $\mathcal{K}$*.*

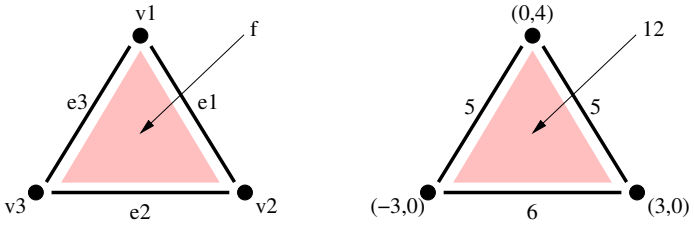

Fig. 1. On the left, an example of cellular complex: it is composed of three 0-cells (v_1, v_2, v_3), of three 1-cells (e_1, e_2, e_3) and of a single 2-cells (f). The boundary of f is constituted of its incident cells v_1, v_2, v_3, e_1, e_2 and e_3. The three edges are the faces of f, and therefore f is a common coface of e_1, e_2 and e_3. On the right, data are associated with topological cells: positions with vertices, lengths with edges and area with f.

An n-cell represents an elementary piece of space of dimension n. In particular, 0-cells are vertices, 1-cells are edges, 2-cells are surfaces, 3-cells are volumes, etc. Thus, graphs are examples of abstract cellular complexes of dimension 1. All set operations are extended to abstract cellular complexes. In particular, we will denote the empty complex $\emptyset$ and use the union $\mathcal{K}_1 \cup \mathcal{K}_2 = (S_1 \cup S_2, \prec_1 \cup \prec_2)$ and the difference of two abstract cellular complexes $\mathcal{K}_1 - \mathcal{K}_2 = (S_1 - S_2, \prec_1 /_{S_1 - S_2})$ where $\prec_1 /_{S_1 - S_2}$ represents the restriction of the relation $\prec_1$ to the elements of $S_1 - S_2$.

Notions of neighborhoods can be defined from the incidence relationships.

Definition 2 (Neighborhoods). *Let $\mathcal{K}$ be a cellular complex and let σ be a cell of $\mathcal{K}$. A cell τ of $\mathcal{K}$ is called* face of σ *iff $(\tau \prec \sigma)$ and $dim(\tau) = dim(\sigma) - 1$. The cell σ is called a* coface of τ. *This relation is denoted by $\tau < \sigma$.*

Let n and p be two integers. Two n-cells of $\mathcal{K}$, σ_1 and σ_2, are p-neighbors *if there exists a p-cell in $\mathcal{K}$ that is commonly incident to σ_1 and σ_2.*

The notion of incidence is close to the notion of *boundary*. For example, if two vertices v_1 and v_2 are the faces of an edge (in other words, v_1 and v_2 are 1-neighbors), they constitute the boundary of that edge. Left of Figure 1 shows an example of cellular complex. We do not detail further these notions. The interested reader should refer to [11,4].

A topological collection is a labeled cellular complex.

Definition 3 (Topological Collection). *Let S be a set of topological cells, $\mathcal{K} = (S, \preceq)$ be a complex of S, and V be an arbitrary set of values. A* topological collection *c over $\mathcal{K}$ with values in V is a partial function from S to V. A topological collection c is represented by a formal sum $\sum_{\sigma \in |c|} v_\sigma.\sigma$ where $v_\sigma = c(\sigma)$.*

We use the following notations: Shape(c) *refers to the complex $\mathcal{K}$, $|c|$ denotes the set of cells $\sigma \in \mathcal{K}$ where $c(\sigma)$ is defined, $C_\mathsf{S}(\mathcal{K}, \mathsf{V})$ denotes the set of topological collections over $\mathcal{K}$ in V, and $C_\mathsf{S}(\mathsf{V})$ denotes the set of topological collections with values in V.*

The indices in notations $C_{\mathsf{S}}(\mathcal{K}, \mathsf{V})$ and $C_{\mathsf{S}}(\mathsf{V})$ refer to the set of topological cells. By convention, when we write a collection c as a sum $c = v_1.\sigma_1 + \cdots + v_p.\sigma_p$, we insist that all σ_i are distinct. Right of Figure 1 gives an example of a topological collection $c = (0,4).v_1 + (3,0).v_2 + (-2,0).v_3 + 5.e_1 + 6.e_2 + 5.e_3 + 12.f$.

2.2 Rewriting Topological Collections

Transforming topological collections using rewriting requires:

- the notion of *sub-collection* (a way to cut out a sub-part of a collection),
- the *extension* of a collection, and
- the *merge* of collections, that is a way to rebuild a collection from the elements resulting of local transformations.

Definition 4 (Sub-collection). *Let c be a collection of $C_{\mathsf{S}}(\mathcal{K}, \mathsf{V})$. A* sub-collection *$s$ of c, is an element of $C_{\mathsf{S}}(\mathcal{K}, \mathsf{V})$ such that $|s| \subseteq |c|$ and $\forall \sigma \in |s|$, $s(\sigma) = c(\sigma)$.*

In other words, a sub-collection of a collection c is a restriction of c to a collection (with the same structure) where only a sub-part of the cells remains labeled. Note that if s is a sub-collection of c, then the collection $c - s$ is defined and is also a sub-collection of c.

Definition 5 (Extension, Collection Matching). *Let c be an arbitrary collection of $C_{\mathsf{S}}(\mathsf{V})$ and let $\mathcal{K}$ be a complex such that* $\mathrm{Shape}(c) \subset \mathcal{K}$ *(i.e. the incidence relationships of the complex underlying c are included in the incidence relationships of $\mathcal{K}$). The extension of c on $\mathcal{K}$, written $c_{|\mathcal{K}}$, is the collection c' of $C_{\mathsf{S}}(\mathcal{K}, \mathsf{V})$ such that $c'(\sigma) = c(\sigma)$ for $\sigma \in |c|$ and c' is left undefined elsewhere.*

We say that a collection c' matches in *a collection $c \in \mathcal{K}$ if* $\mathrm{Shape}(c') \subset \mathcal{K}$ *and ${c'}_{|\mathcal{K}}$ is a sub-collection of c.*

Merging collections with similar shape naturally corresponds to the addition of their formal sum representation. To merge collections with different shapes, we first build a common abstract cellular complex on which we then perform the standard addition. Note that merge is commutative.

Definition 6 (Collections Merge). *Let $c_1 \in C_{\mathsf{S}}(\mathcal{K}_1, \mathsf{V})$ and $c_2 \in C_{\mathsf{S}}(\mathcal{K}_2, \mathsf{V})$ be two topological collections such that $|c_1| \cap |c_2| = \emptyset$. The* merge of c_1 and c_2, *denoted $c_1 \uplus c_2$, is defined by: $c_1 \uplus c_2 = c_{1|\mathcal{K}} + c_{2|\mathcal{K}}$ where $\mathcal{K} = \mathcal{K}_1 \cup \mathcal{K}_2$.*

Remark: if a collection c' matches in a collection c, then c can be written $c = c' \uplus c''$ where c'' is uniquely defined.

Definition 7 (Rewriting Relation). *Let R be a relation over $C_{\mathsf{S}}(\mathsf{V})$.* One step of rewriting generated by *R is the relation $c_1 \rhd_R c_2$ which is true either iff $c_1 = c_2$ or there exists $(l, r) \in R$ such that*

- *$c_1 = c \uplus l$ and $c_2 = c \uplus r$, and*
- *$(\mathrm{Shape}(r) - \mathrm{Shape}(l)) \cap \mathrm{Shape}(c) = \emptyset$.*

In the latter case, the collection l is called the redex *of the couple $c_1 \rhd_R c_2$.*

One step of parallel rewriting generated by R *is the relation $c|\rhd_R c'$ which is true iff there exists a sequence $c = c_1 \rhd_R c_2 \rhd_R \ldots \rhd_R c_p = c'$ such that the redexes l_i of the $c_i \rhd_R c_{i+1}$ are sub-collections of c and are mutually disjoint (that is, $|l_i| \cap |l_j| = \emptyset$ for all $i \neq j$).*

In this definition, the condition $(\mathrm{Shape}(r) - \mathrm{Shape}(l)) \cap \mathrm{Shape}(c) = \emptyset$ stresses the fact that the "new" cells appearing in r w.r.t. l must really be new, even in c. Informally, the definition can be read as follows. When l matches in c_1, the application of (l, r):

1. removes the cells of l (only the elements of c_1 that do not appear in the shape of l remain with their coefficient), and
2. adds the cells of r.

In order to rebuild a collection, r may refer to some cells of $\mathrm{Shape}(c) - \mathrm{Shape}(l)$. These references correspond to the usual notion of *invariant* (or *gluing graph*) in graph rewriting [12]. If there is no matching, then the application of the rule is void and the rewriting is the identity. The parallel application of a relation R can be represented by the following diagram:

$$\begin{array}{ccccccccc} c_1 & = & l_1 & \uplus & \ldots & \uplus & l_n & \uplus & c \\ \downarrow |\rhd_R & & \downarrow \rhd_R & & & & \downarrow \rhd_R & & \downarrow \rhd_R \\ c_2 & = & r_1 & \uplus & \ldots & \uplus & r_n & \uplus & c \end{array}$$

where all the l_i are disjoint.

2.3 Transformation

The use of an explicit relation R to generate a rewriting relation is not very effective. In the following, we propose to generate a relation R from a set of rules $\alpha \to \beta$ where α is a *pattern* that matches a (sub-)collection and β is a *collection expression*. This process, called a *transformation*, relies on the use of *variables* and on the notion of *environment*.

Definition 8 (Variables, Environments and Expressions). *Let $(\mathsf{S}_n^{\mathsf{var}})$ with $n \in \mathbb{N}$, be a family of distinguished symbols. An element x of $\mathsf{S}_n^{\mathsf{var}}$ is called a* (topological) cell variable of dimension n. *The set of all cell variables is written $\mathsf{S}^{\mathsf{var}} = \bigcup_n \mathsf{S}_n^{\mathsf{var}}$. Let $\mathsf{V}^{\mathsf{var}}$ be a set of distinguished values called* value variables.

An environment *(resp. a* cell environment*) is a function from $\mathsf{V}^{\mathsf{var}}$ (resp. $\mathsf{S}^{\mathsf{var}}$) to V (resp. S, such that the dimension of arguments and images match). The set of environments (resp. cell environments) is written Γ_{V} (resp. Γ_{S}).*

Let $\Sigma \subset (\mathsf{V} \cup \mathsf{V}^{\mathsf{var}})^*$ be a distinguished set of words built on V and $\mathsf{V}^{\mathsf{var}}$ called expressions. *We assume that Σ is equipped by a function $\xi : \Sigma \times \Gamma_{\mathsf{V}} \to \mathsf{V}$, called the* evaluation function.

Definition 9 (Rule and Rule Occurrence). *A* pattern *is a topological collection of* $C_{\mathsf{S}^{\mathsf{var}}}(\mathsf{V}^{\mathsf{var}})$. *A* collection expression *is a collection of* $C_{\mathsf{S}^{\mathsf{var}} \cup \mathsf{S}}(\Sigma)$. *A* rule *is a couple* $\alpha \rightarrow \beta$ *where* α *is a pattern and* β *is a collection expression.*

Let $\alpha = a_1.x_1 + ... + a_p.x_p$ *be a pattern and* $\beta = e_1.x_1 + ... + e_q.x_q + e'_1.\sigma_1 + ... + e'_{p'}.\sigma_{p'}$, *with* $q \leq p$, *be a collection expression. A couple of collections* (l, r) *is an* occurrence of the rule $\alpha \rightarrow \beta$ *iff it exists* $\rho_{\mathsf{V}} \in \Gamma_{\mathsf{V}}$ *and* $\rho_{\mathsf{S}} \in \Gamma_{\mathsf{S}}$ *such that:*

$$l = \rho_{\mathsf{V}}(a_1).\rho_{\mathsf{S}}(x_1) + ... + \rho_{\mathsf{V}}(a_p).\rho_{\mathsf{S}}(x_p)$$

and

$$r = \xi(\rho_{\mathsf{V}}, e_1).\rho_{\mathsf{S}}(x_1) + ... + \xi(\rho_{\mathsf{V}}, e_q).\rho_{\mathsf{S}}(x_q) \ + \ \xi(\rho_{\mathsf{V}}, e'_1).\sigma_1 + ... + \xi(\rho_{\mathsf{V}}, e'_{p'}).\sigma_{p'}$$

The set of all occurrences of $\alpha \rightarrow \beta$ *is written* $\mathrm{Occ}_{\alpha \rightarrow \beta}$.

We are now able to define the concept of transformation.

Definition 10 (Transformation). *Let* T *be a set of rules* $\alpha_i \rightarrow \beta_i$. *The* transformation *associated with* T *is the relation* $|\triangleright_{R_T}$ *where* $R_T = \bigcup_i \mathrm{Occ}_{\alpha_i \rightarrow \beta_i}$.

3 The MGS Language

The previous section gives a formal description of topological collections and transformations without any explicit definition of expressions. In this section, we propose an implementation of this formalism through the definition of an experimental functional language called MGS. In particular, we focus on the definition of topological collections and on the specification of a transformation.

3.1 MGS Topological Collections

The MGS language provides the means to specify topological collections. In particular, the programmer is allowed to create new cellular complexes and to label them.

The very basic function `new_cell` creates fresh topological cells. It takes three parameters providing the dimension of the cell to be created, the sequence of its faces and the sequence of its cofaces. Associated with a recursive `letcell...in` construction (similar to a `let rec...in` in OCaml), this simple function allows to create basic complexes. Finally, the specification of a topological collection consists in associating values with cells relying on the formal sum notation given in Definition 3.

Thus, the example of Figure 1 is evaluated by the following MGS program:

```
letcell v1 = new_cell 0 ()         (e1,e3)
and     v2 = new_cell 0 ()         (e1,e2)
and     v3 = new_cell 0 ()         (e2,e3)
and     e1 = new_cell 1 (v1,v2)    (f)
and     e2 = new_cell 1 (v2,v3)    (f)
and     e3 = new_cell 1 (v1,v3)    (f)
and     f  = new_cell 2 (e1,e2,e3) ()      in
  (0,4)*v1 + (3,0)*v2 + (-3,0)*v3 + + 5*e1 + 6*e2 + 5*e3 + 12*f
```

The reader is invited to pay attention that topological collections are heterogeneous, *i.e.*, any type of data can be associated with the cells within a collection (here couples of integers and integers).

3.2 MGS Patch Transformations

In Section 2, patterns and collection expressions are defined as topological collections. The difficulty of their descriptions is hidden by the implicit partial orders that they come with. As an implementation of this formalism, MGS has to make these objects as explicit as possible with a special syntax for the programmer. The objectives of the syntax is to simplify as much as possible the specification of the rewriting rules. Different rule languages have been developed. In this article, we focus on *patch transformations* (called *patches* from now on), a kind of transformation specially designed for the specification of topological surgeries.

A patch is specified as follows:

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

The language *Pattern* has been introduced to specify the left hand side (l.h.s.) while the right hand side (r.h.s.) are basic expressions of the language where MGS operators (*e.g.*, `new_cell`) can be used to specify new collections.

A patch is a function having a topological collection as argument. More specifically, the expression $P(c)$ where c is an MGS collection, evaluates into a new collection c' such that $c|\triangleright_{R_P} c'$ as defined in Definition 10. Obviously, the relation R_P (which may be an infinite set) is not computed within this evaluation. In fact, a pattern matching mechanism has been developed to compute one of the possible collections c'. The interested reader should refer to [13] for an elaboration.

In the following, we detail the patch transformation language. In order to illustrate our comments, we consider the topological modification described on Figure 2: an edge named `e`, whose faces are called `v1` and `v2`, is matched and replaced by two new edges `e1` and `e2` together with a new vertex `v`; the edge `e1` is bound by vertices `v1` and `v`, and `e2` by `v2` and `v`. This rule specifies the insertion of a vertex on an edge.

Fig. 2. Insertion of a vertex on an edge

Patch Patterns. Rather than specifying patterns of rules as explicit topological collections, patch patterns describe them implicitly with a list of constraints (on dimension and incidence). The topological cells of sub-collections that are matched by these patterns, have to respect the constraints.

The specification of a pattern relies on the following grammar:

Pattern

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

Op

$o ::= \varepsilon \mid \texttt{<} \mid \texttt{>}$

Clause

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

where ε represents the empty word. A pattern (*Pattern*) is a finite list of *clauses* separated by some operators (*Op*). A clause (*Clause*) corresponds to a topological cell to be matched. Each clause is characterized by some optional information that will guide the pattern matching process:

- `x` is a pattern variable that allows to refer to the cell matched by the clause anywhere else in the rule. A variable can be used before its definition. Nevertheless, a name refers to one and only one element. If two clauses share the same identifier, they will match the same cell (the predicates of both clauses have to hold together). Since a clause refers to an element of a collection, that is a couple (value v, cell σ), the use of the variable `x` in the rest of the rule might be ambiguous. We avoid such a problem by setting that `x` binds for the value v and the expression `^x` binds for the cell σ.
- The expression exp_d associated with the field `dim` returns an integer constraining the dimension of the cell matched by the clause,
- Expressions exp_f and exp_{cf} respectively associated with fields `faces` and `cofaces` are evaluated in sequences of topological cells. These lists constrain the incidence relationships of the matched cell. They do not have to be exhaustive; a cell can get more (co)faces than those referred in the pattern. In these expressions, pattern variables or direct references to topological cells may be used,
- The last expression exp_b can be used to specify some arbitrary predicate the cell has to satisfy (it can be used to constrain the value associated with the cell for example),
- Two kinds of clause can be specified depending on the presence of the unary operator `~`. When it is present, the clause is considered as a context for the pattern matching and the associated cell in the sub-collection is not considered as consumed by the pattern matching. If no operator `~` is specified, the topological cell is matched and consumed by the rule: no other sub-collection can refer to a not-tilded element.
  The consumption of a cell during the pattern matching ensures sub-collections to be disjoint as required by Definition 7. Operator `~` allows a same cell to be referred into distinguished sub-collections. Using notations of Definition 7, such cell belongs to $\text{Shape}(l) \cap \text{Shape}(c)$.

Operators *Op* correspond to syntactic sugar. The infix binary operator `<` (resp. `>`) constraints the element matched by the left operand to be a face (resp. coface) of the element matched by the right operand. Therefore, any patterns

of type `a:[...] < b:[...]` can be rewritten `a:[..., cofaces=b] b:[..., faces=a]`. When no operator separates two clauses, there is no constraint between the matched elements.

The pattern corresponding to the l.h.s. of the rule pictures on Figure 2 is specified by:

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

Here, `v1` and `v2` are not consumed as they are used as a context for the pattern matching of `e`. They are referred to in the r.h.s. but they are not rewritten.

Right Hand Side of a Rule. The r.h.s. of a rule is an MGS expression that has to evaluate to a collection. In other words, the r.h.s. can be any MGS program. It makes the transformation very flexible and allows program factorization. Therefore, using the MGS syntax for defining new collections given Section 3.1, the graphical rule of Figure 2 is specified in MGS by:

```
~v1 < e:[ dim = 1 ] > ~v2 =>
  letcell v  = new_cell 0 ()      (e1,e2)
  and     e1 = new_cell 1 (^v1,v) (cofaces ^e)
  and     e2 = new_cell 1 (^v2,v) (cofaces ^e) in
    (some expression)*v
```

In the r.h.s., the three new cells are created and then a collection is defined. The cofaces of the new defined edges `e1` and `e2` are the cofaces of the edge matched by `e`, so that if `e` belongs to the boundary of a 2-cell, `e1` and `e2` belong to it after the application of the patch. The new sub-collection consists of the association of an arbitrary value with the new vertex `v` while the collection is left undefined on `e1` and `e2`.

4 Mesh Refinement Algorithms

In this section, we propose to use the rule-based programming of patches to specify a classical but not trivial familly of algorithms in CAD: the *refinement of meshes*.

4.1 Loop Subdivision

Subdivision algorithms generate to the limit smooth surfaces by iterating subdivisions of polygonal mesh. In [14], the descriptions of the usual subdivision processes dedicated to the geometrical modeling and to the animation of solids can be found. These algorithms are locally specified by *masks*. A mask is a cellular complex describing a part of a mesh centered on an element to be refined (*e.g.*, an edge, a triangle, etc.). The refinement consists in inserting a new vertex whose coordinates are determined by an affine combination of the other vertices belonging to the mask. The properties of smoothness of the surface obtained to the limit and therefore the quality of the subdivision depends on the mask. The

intended properties correspond to surfaces as smooth as possible (C^1-continuity or C^2-continuity everywhere for example) and are difficult to obtain on arbitrary meshes that exhibit irregularities on singular points. The mask is then chosen depending on the kind of the mesh (triangular, quadrangular, etc.) and the property to hold.

We are here interested in the Loop subdivision [15]. This algorithm:

- is based on *vertex insertions*: a vertex is created on each edge of the mesh within each application of the algorithm; old nodes are conserved and new nodes are linked to them by new edges;
- works on and generates *triangular meshes*;
- is *approximating*: old nodes positions are changed in the new mesh and new nodes positions are computed as a function of the old nodes positions in the old mesh (approximating subdivisions are opposed to interpolating ones for which old nodes positions are not affected).

The algorithm describes on the one hand the topological modifications and on the other hand the geometrical modifications.

Topological Modification. The mesh refinement is based on the polyhedral subdivision shown on Figure 3. A vertex is inserted on each edge and triangles are refined in 4 smaller triangles.

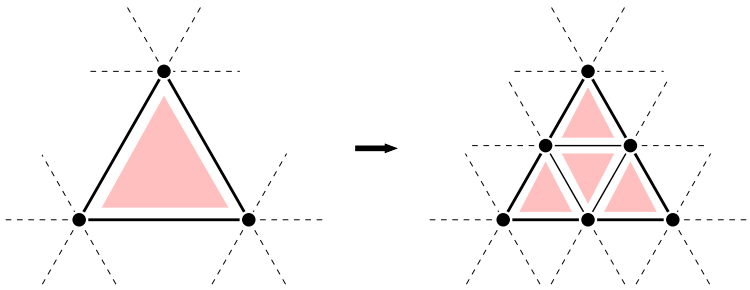

Fig. 3. Loop Algorithm: Topological Modification

Geometrical Modification. Loop's masks provide the information required to compute the positions of new vertices (as a function of the coordinates of the old ones) and the new positions of the old vertices (see Figure 4). On the left, coordinates of a new vertex v inserted on the edge whose boundary is composed of vertices v_1 and v_2 are given by:

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

On the right, new coordinates of an old vertex v is computed as a function of its 1-neighbors positions:

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

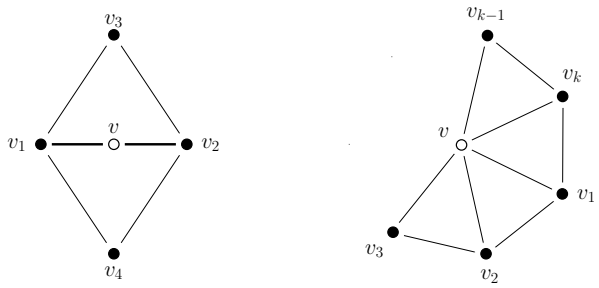

Fig. 4. Loop Algorithm: Geometrical Modification

4.2 Representation of Geometrical Objects in MGS

Meshes can be easily represented by MGS topological collections. Since they are surfaces composed of polygons, their structures are represented by a cellular complex of dimension 2. In the topological collections, only the geometrical positions of the 0-cells matter. So, we associate the constant value `'Triangle` with triangles, the constant value `'Edge` with edges, and coordinates with vertices. The coordinates of a vertex are represented by an MGS record value (that is equivalent to a C `struct`). The type of a coordinate is defined as follows:

```
record coordinate = {
    x:float, y:float, z:float, old:(coordinate|'Nil)   }
```

whose fields `x`, `y` and `z` encode the coordinates in the 3D space. The last field `old` is used to distinguish new and old vertices. For new vertices, `old` is set to the value `'Nil` (an MGS constant). For old vertices, `old` is set to the coordinates before the Loop algorithm iteration.

To simplify the description of the MGS program, we assume the existence of a function `addCoord` that sums two values of type `coordinate`, and the global constant `O` corresponding to the 3D space origin.

4.3 Loop Subdivision in MGS

Three steps are used to implement the Loop algorithm.

Update of the old vertices. In this step, we save in the field `old` the current coordinates of each vertex and we update the coordinates with respect to the Loop's mask (see Figure 4 on the right).

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

The single rule of this patch does not change the topology: one element of dimension 0 named `v` is matched and an elementary collection on `v` is computed in the r.h.s. Variables `s` and `b` correspond respectively to the sum of the coordinates of the 1-neighbors (see Definition 2) of `^v` and to the β coefficient of the Loop's mask (see Equation (2)). The MGS primitive `ccellsfold(`f`,`z`,`σ`,`i`)` computes the sequence $(v_1, \ldots, v_n)$ of the values associated with the i-neighbors of σ and computes the value $f(v_1, (\ldots f(v_n, z)\ldots))$. Old coordinates referred by the variable `v` are saved in the field `old`.

Insertion of new vertices. The following patch is quite similar to the patch given as an example in Section 3.2.

```
patch odd_vertex = {
  ~v1 < e:[dim=1] > ~v2
  ~v1 < ~e13 > ~v3 < ~e23 > ~v2
  ~v1 < ~e14 > ~v4 < ~e24 > ~v2
  =>
    letcell v  = new_cell 0 ()      (e1,e2)
    and     e1 = new_cell 1 (^v1,v) (cofaces ^e)
    and     e2 = new_cell 1 (^v2,v) (cofaces ^e) in
      { old = 'Nil, x = v_x, ... }*v + 'Edge*e1 + 'Edge*e2
}
```

The pattern is extended to take into account the vertices `v3` et `v4` required by the Loop's mask (see Figure 4 on the right). Only the edge `e` is consumed to be removed and replaced. All the other clauses are used as pattern matching context. Coordinates (v_x, v_y, v_z) of the new created vertex `v` are computed using the field `old` of vertices `v1`, `v2`, `v3` and `v4` w.r.t. Equation (1).

Creation of the refined triangles. The previous patch leads to the transformation of the mesh triangles into hexagons. These hexagons are then removed and replaced by new smaller triangles.

```
patch subdivideFace = {
  f:[ dim = 2, faces = (^e1,^e2,^e3,^e4,^e5,^e6) ]
  ~v1 < ~e1 > ~v2:[ v2.old == 'Nil ] < ~e2 >
  ~v3 < ~e3 > ~v4:[ v4.old == 'Nil ] < ~e4 >
  ~v5 < ~e5 > ~v6:[ v6.old == 'Nil ] < ~e6 > ~v1
  =>
    letcell a1 = new_cell 1 (^v2,^v4) (f1,f4)
    and     a2 = new_cell 1 (^v4,^v6) (f2,f4)
    and     a3 = new_cell 1 (^v6,^v2) (f3,f4)
    and     f1 = new_cell 2 (a1,^e2,^e3) ()
    and     f2 = new_cell 2 (a2,^e4,^e5) ()
    and     f3 = new_cell 2 (a3,^e6,^e1) ()
    and     f4 = new_cell 2 (a1,a2,a3) () in
      'Edge*a1 + ... + 'Triangle*f4 }
```

This patch is composed of a single rule sketched in Figure 5. Note the presence of tests `v`i`.old == 'Nil` to distinguish old and new vertices.

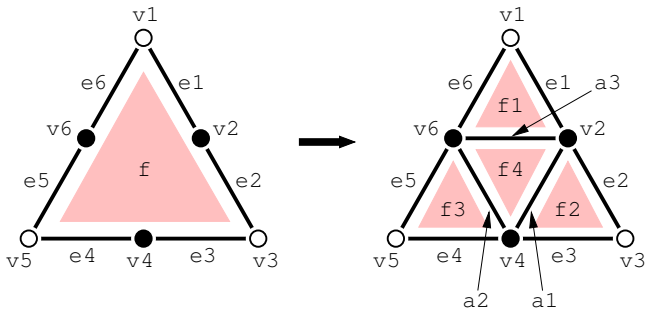

Fig. 5. Construction of the 4 refined triangles in the Loop's algorithm

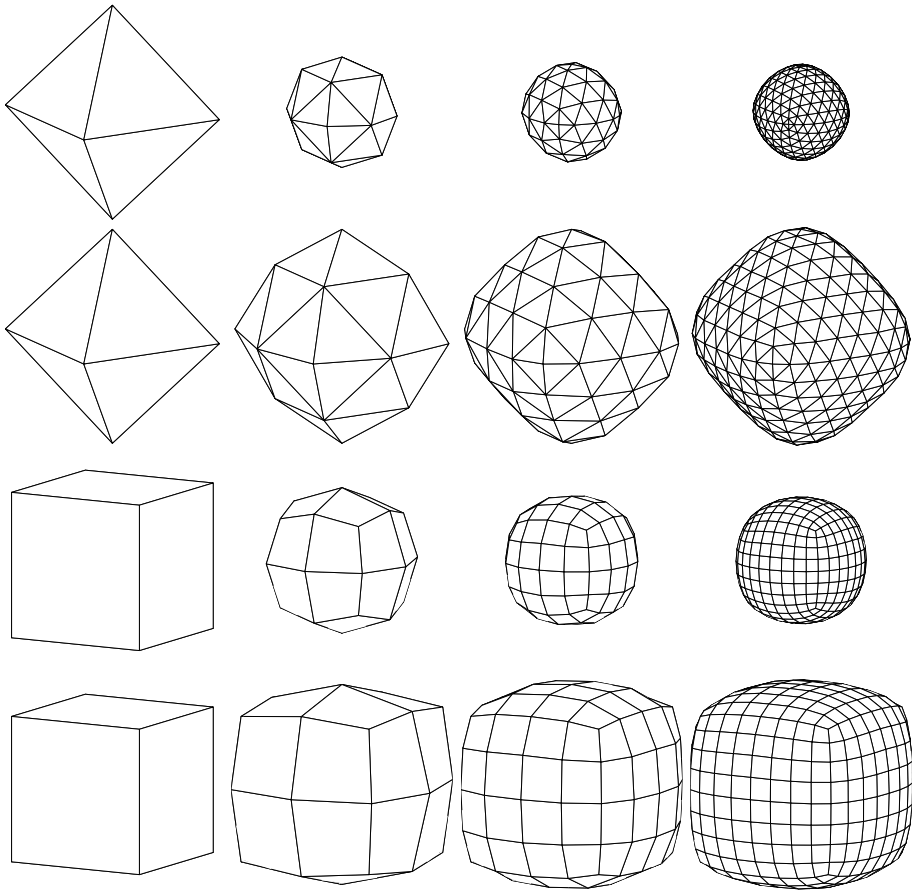

Fig. 6. Results of the application of subdivision algorithms. From top to bottom, the Loop's algorithm, the Butterfly algorithm, the Catmull-Clark's algorithm and the Kobbelt's algorithm. From the left to the right, the initial state then 3 iteration steps. These pictures have been generated by the current MGS prototype.

Figure 6 shows an example of Loop's algorithm iterations generated by the current prototype of MGS. The figure also shows outputs of three other algorithms [14] that have been implemented in the same way in MGS.

5 Conclusion

In this article, we have presented the MGS language as a positive answer to the question of a framework that allows a declarative, intensional and coordinate-free specification of mesh algorithms.

The MGS main notions, topological collections and transformations, rely on a formal definition of a topological rewriting relation. This computational model is based on the substitution of labeled cellular complexes that are an extension of graphs to higher dimensions. The expressiveness of topological collections and their transformations allows the programming of complex algorithms (in a large range of domains) in a very concise way. It has been exemplified in this paper with the specification of non trivial mesh subdivision algorithms.

Topology has already been introduced in graph transformation in different kinds of contexts [16,17,18]. Work presented in [17] is far from our purpose since the concept of topology is used to ensure structural constraints on graphs being transformed. Nevertheless, MGS and its current implementation can be used as a programming language to express and simulate the proposed model transformation rules. In [16], GRiT (Graph Rewriting in Topology), a kind of hyper-graph rewriting, is developed to take into account topological properties (from homology and homotopy) in models of parallel computing based on rewriting theory [19]. This approach has been applied in the modeling of chemical reactions, DNA computing, membrane computing, etc. MGS has also been extensively used in this context [4]. Since we focus in this paper on an application in topological modeling, our work can be compared to [18] where graph transformations have been applied to specify topological operations on *G-maps*. G-maps are a data structure used to encode cellular complexes corresponding to a specific class of topological objects called *quasi-manifolds* [20]. Roughly speaking, G-maps are graphs where vertices are named *darts* and where edges correspond to *involutions* defined between darts. Because G-maps are a specific kind of cellular complexes, they can be handled by the approach presented here. Indeed, they have been implemented in MGS (see [10] for an elaboration) as well as many of the applications proposed in [18] (especially in biology). The formal approach developped in [18] is very specific and focuses on the correct handling of involutions in G-maps construction and deletion operations.

Our contribution differs from the works mentionned above on two main points: the object to be transformed and the addressed mathematical framework. These works are all based on (hyper-)graph transformation. We propose to transform cellular complexes since the extension from graphs to complexes seems intuitive and relevant in a lot of application area [21]. Our mathematical description is not based on the usual graph morphisms and pushouts (like in [12,17,18]). Our objective was here to relate the notion of transformation to the very definition

of topological collections in the context of a programming language. It requires to give explicit details on the construction of collections and how the rewriting is effectively done. Our mathematical formalization of complex transformation is inspired by [22] where graph rewriting based on a (multi-)set point of view is developed. The proposed model is close to term rewriting modulo associativity and commutativity (where the l.h.s. of a rule is removed and the r.h.s. is added) and can be applied on the formal sum notation of topological collections. This kind of approach also allows to extend results from term rewriting to topological rewriting (as we did for termination in [23]).

References

1. Chaikin, G.: An algorithm for high speed curve generation. Computer Graphics and Image Processing 3, 346–349 (1974)
2. Prusinkiewicz, P., Samavati, F.F., Smith, C., Karwowski, R.: L-system description of subdivision curves. International Journal of Shape Modeling 9(1), 41–59 (2003)
3. Smith, C., Prusinkiewicz, P., Samavati, F.: Local specification of surface subdivision algorithms. In: Pfaltz, J.L., Nagl, M., Böhlen, B. (eds.) AGTIVE 2003. LNCS, vol. 3062, pp. 313–327. Springer, Heidelberg (2004)
4. Giavitto, J.L., Michel, O.: The topological structures of membrane computing. Fundamenta Informaticae 49, 107–129 (2002)
5. Giavitto, J.L.: Invited talk: Topological collections, transformations and their application to the modeling and the simulation of dynamical systems. In: Nieuwenhuis, R. (ed.) RTA 2003. LNCS, vol. 2706, pp. 208–233. Springer, Heidelberg (2003)
6. Spicher, A., Michel, O.: Declarative modeling of a neurulation-like process. BioSystems 87, 281–288 (2006)
7. de Reuille, P.B., 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 103(5), 1627–1632 (2006)
8. Spicher, A., Michel, O., Giavitto, J.L.: Algorithmic self-assembly by accretion and by carving in mgs. In: Talbi, E.-G., Liardet, P., Collet, P., Lutton, E., Schoenauer, M. (eds.) EA 2005. LNCS, vol. 3871, pp. 189–200. Springer, Heidelberg (2006)
9. Valencia, E., Giavitto, J.L.: Algebraic topology for knowledge representation in analogy solving. In: ECCAI, A. (ed.) European Conference on Artificial Intelligence (ECAI 1998), Brighton, UK, Christian Rauscher, August 23-28, pp. 88–92 (1998)
10. Spicher, A.: Transformation de collections topologiques de dimension arbitraire. Application à la modélisation de systèmes dynamiques. PhD thesis, Université d'Évry (2006)
11. Munkres, J.: Elements of Algebraic Topology. Addison-Wesley, Reading (1984)
12. Ehrig, H., Pfender, M., Schneider, H.J.: Graph grammars: An algebraic approach. In: IEEE Symposium on Foundations of Computer Science, FOCS (1973)
13. Giavitto, J.L., Michel, O.: Pattern-matching and rewriting rules for group indexed data structures. In: ACM Sigplan Workshop RULE 2002, Pittsburgh, pp. 55–66. ACM, New York (2002)
14. Zorin, D.: Subdivision zoo. In: Subdivision for modeling and animation, Schröder, Peter and Zorin, Denis, pp. 65–104 (2000)
15. Loop, T.L.: Smooth subdivision surfaces based on triangle. Master's thesis, University of Utah (August 1987)

16. Liu, J.Q., Shimohara, K.: Graph rewriting in topology. i. operators and the grammar. SIG-FAI 45, 21–26 (2001)
17. Levendovszky, T., Lengyel, L., Charaf, H.: Extending the dpo approach for topological validation of metamodel-level graph rewriting rules. In: SEPADS 2005: Proceedings of the 4th WSEAS International Conference on Software Engineering, Parallel & Distributed Systems, Stevens Point, Wisconsin, USA, pp. 1–6. World Scientific and Engineering Academy and Society (WSEAS) (2005)
18. Poudret, M., Arnould, A., Comet, J.P., Gall, P.L.: Graph transformation for topology modelling. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) ICGT 2008. LNCS, vol. 5214, pp. 147–161. Springer, Heidelberg (2008)
19. Liu, J.Q., Shimohara, K.: Graph rewriting in topology iv: Rewriting based on algebraic operators (algorithms in algebraic systems and computation theory). RIMS Kokyuroku 1268, 64–72 (2002)
20. Lienhardt, P.: Topological models for boundary representation: a comparison with n-dimensional generalized maps. Computer-Aided Design 23(1), 59–82 (1991)
21. Tonti, E.: On the mathematical structure of a large class of physicial theories. Rendidiconti della Academia Nazionale dei Lincei 52(fasc. 1), 48–56 (1972); Scienze fisiche, matematiche et naturali, Serie VIII
22. Raoult, J.C., Voisin, F.: Set-theoretic graph rewriting. Technical Report RR-1665, INRIA (April 1992)
23. Giavitto, J.L., Michel, O., Spicher, A.: Spatial organization of the chemical paradigm and the specification of autonomic systems. In: Software-Intensive Systems (2008)