Algebraic Specification and Development in Geometric Modeling*

Y. Bertrand, J.-F. Dufourd, J. Françon, P. Lienhardt

Département d'Informatique, Université Louis-Pasteur / CNRS
7, rue René-Descartes, 67084 Strasbourg Cedex, France, tel: 33 88 61 43 00,
e-mail: {bertrand, dufourd, françon, lienhardt} @dpt-info.u-strasbg.fr

Abstract. For several years now, the Geometric Modeling Group of Strasbourg has been working on new formal concepts and tools for describing and manipulating the boundary representation of geometric objects. In a large project of an interactive modeller for volumic objects, the description of which is based on generalized maps, it attempts to cover the whole process from mathematical modeling to efficient implementation, via a complete algebraic specification. Basic concepts and results of this experiment in horizontal and vertical software specification and development are presented along with several illustrations. Advances in algebraic specification methodology are highlighted, specially hierarchical construction of ordered sorts and operations.

1 Introduction

For geometrical applications, algebraic specifications improve modeling and software development processes. We try to show it through the report of a large-size project.

Several methods have been proposed for describing and manipulating geometric objects on computers [41, 42]. One of them, namely boundary representation [2, 38, 39], consists in describing objects by their subdivisions, in vertices, edges, faces and volumes. In this frame, mathematical combinatorial models have been proposed. The oldest one is the combinatorial map [16], well studied in [28, 44, 43, 8]. It has numerous uses in geometric modeling [1, 31, 21, 11]. New combinatorial topological models of n-dimensional objects rely on map extensions [9, 29, 32, 13]. Among them, we have proposed the n-dimensional generalized map, or n-g-map [33, 34, 35]. It is a particularly efficient model to deal with topology of manifolds. When completed by an embedding model, it allows to describe the entire geometry of n-dimensional manifold objects.

Furthermore, for a long time, it has been attempted to improve the computer graphics programming techniques by functional [40], logical [20], or object-oriented [26] approaches. But we think that decisive progress will come above all from formal specification techniques, particularly algebraic ones [17, 3, 46]. Their first outstanding use was in [36, 37] where graphical basic objects and operations were algebraically specified . Other works normalize libraries of graphical interactive primitives, like GKS [10], or describe particular algorithms [30]. Note also the attempt of [25] which specifies elementary geometric constructions with an extended OBJ3. In [12, 13, 14], we give the basis of an algebraic specification of maps and extensions. However using algebraic specifications has never been reported for computer graphics real-size applications.

So, for a full-scale test of both the operationality of the n-g-map model and the efficiency of the algebraic specification techniques, our group has designed and developed original and complex software, namely an interactive volumic modeller, i.e., a program which helps to interactively build and handle 3D geometrical objects, with topological basis [5]. The algebraic specification of this software is the continuation of our preceeding work. It gives us a rigorous hierarchical description of object sorts and operations, with a functional constructive point of view, an horizontal structuring, a vertical development

* This research has been supported by the GDR *Programmation*, CNRS, France.

[18], and even a logical prototyping [3] and a real implementation. As far as we know, this is the first attempt to develop an interactive volumic modeller on a topological basis, and, moreover, with the help of a complete algebraic specification. This kind of product is the kernel of specialized software for industrial applications, e.g., mechanical CAD, 3D meshing, design, architecture.

We present here the n-g-maps and the modeller. However, the paper is especially centered on the specification problem and on the solutions supplied. We insist on the theoretical fundamentals, the methodology of construction of the ordered sorts and the specification of the main operations. We rapidly discuss a few choices we made on realization, interactivity and implementation. The power of the modeller is illustrated by complex geometric objects.

Section 2 describes the embedded n-g-maps, derived notions and properties. Section 3 presents the main functionalities and user operations of the modeller. Section 4 proposes an order-sorted kernel of algebraic specification for the modeller. Section 5 deals with the vertical development. Section 6 treats the horizontal structuring. Section 7 presents technical features of the modeller. Section 8 summarizes this experiment and gives future prospects. An Appendix shows screen pictures of objects built by the modeller.

2 A Geometrical Model

2.1 A Topological Model

The *n-dimensional generalized map, n-g-map* for short, is a combinatorial notion, which is used as a general basis for the topological modeling in dimension n. The mathematical elements we give here are only those used in the following. A complete study as well as justifications for modeling and mathematical proofs of soundness are in [33, 34, 35].

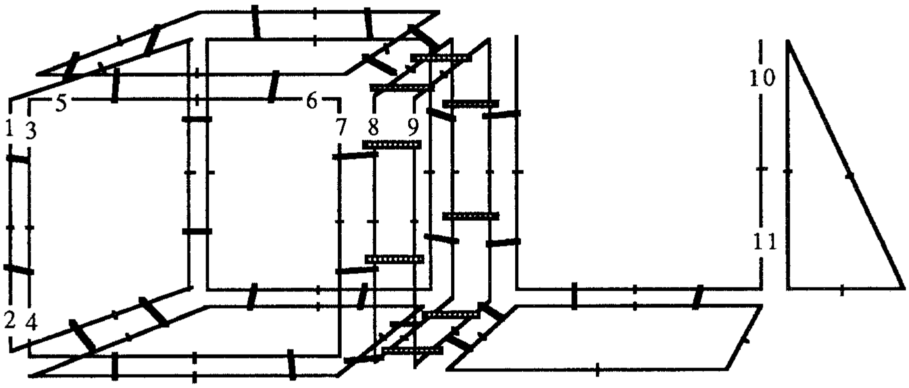


Fig. 1. An example of a 3-g-map with 2 connected components.

Recall that an *involution* α in a set D is a bijection such that $\alpha(\alpha(x)) = x$ for any x in D. Let $n \geq -1$. An *n-g-map* $G = (D, \alpha_0, ..., \alpha_n)$ consists of a finite set D of *darts* with n+1 companion involutions α_k in D, $0 \leq k \leq n$, with the constraints: $\alpha_k \circ \alpha_j$ is an involution, for $k+2 \leq j \leq n$. Darts x and y are said to be *k-sewn* in G if $\alpha_k(x) = y$ (and $\alpha_k(y) = x$).

Fig. 1 (and Picture 1 in Appendix) shows a 3-g-map represented, or *embedded*, in $\mathbf{R}^3$. Darts are natural numbers (only partially written in the drawing for clarity) embedded as half straight line segments in $\mathbf{R}^3$. Involution α_0 is symbolized by thin strokes between half segments, α_1 by their junctions, α_2 by thicker black strokes, and α_3 by hatched strokes. Thus $\alpha_0(1) = 2$, $\alpha_0(2) = 1$, $\alpha_1(3) = 5$, $\alpha_2(1) = 3$, $\alpha_2(8) = 7$, $\alpha_3(8) = 9$, $\alpha_3(9) = 8$, $\alpha_1(10) = 10$, $\alpha_2(9) = 9$, $\alpha_3(1) = 1$, etc.

An n-g-map can be viewed as a *multigraph* in D with n+1 companion symmetric binary relations determined by the α_k, for $0 \leq k \leq n$. The result is a notion of (*strong*) *connected component* for n-g-map as in multigraphs (cf. Fig. 1). The connected component of dart x in G is denoted $<\alpha_0, ..., \alpha_n>(x)$. New simple notions can also be introduced.

For $0 \leq k \leq n$, the *(n-1)-g-map of k-cells* of the n-g-map $G = (D, \alpha_0, ..., \alpha_n)$ is defined by $G_k = (D, \alpha_0, ..., \alpha_{k-1}, \alpha_{k+1}, ..., \alpha_n)$. The *k-cell* of G incident to dart x is the connected component of x in G_k. The 0-cells, 1-cells, 2-cells and 3-cells of G are respectively called the *vertices*, *edges*, *faces* and *volumes* of G (Fig. 2).

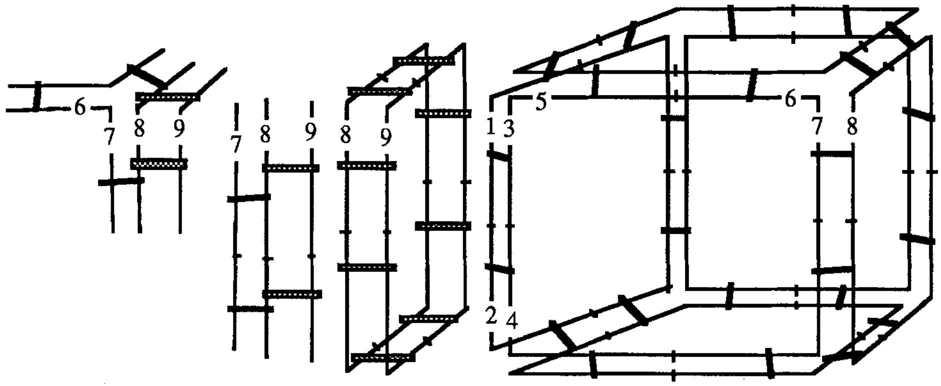


Fig. 2. K-cells incident to dart 8 of the 3-g-map in Fig. 1.

Similarly, for $0 \leq k \leq n$, the *(k-1)-g-map of simple k-cells* of G is defined by $G'_k = (D, \alpha_0, ..., \alpha_{k-1})$. The *simple k-cell* of G incident to dart x is the connected component of x in G'_k (Fig. 3). So, constraints $\alpha_k \circ \alpha_j$ is an involution of the n-g-map definition say that k-sewings are always done in n-g-maps for *all* the darts of *isomorphic* simple k-cells.

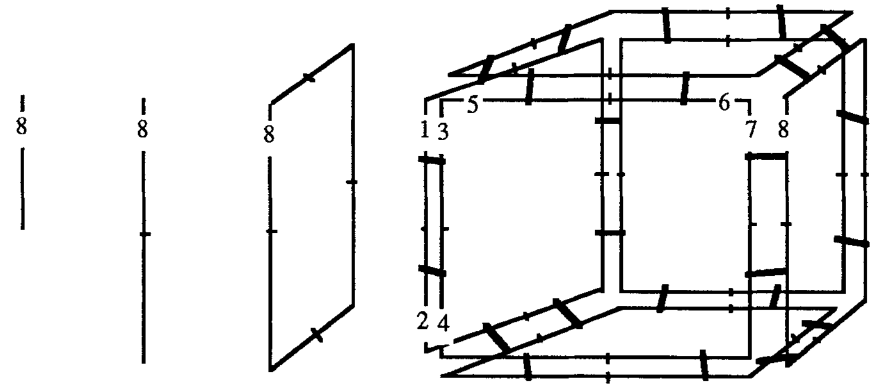


Fig. 3. Simple (or s.) k-cells incident to dart 8 of the 3-g-map in Fig. 1.

2.2 A Geometrical Embedding Model

For any dimension k, $0 \leq k \leq n$, the geometric *k-embedding* of an n-g-map in an n-dimensional Euclidean space R^n is fixed by labels associated with k-cells, in fact by a single dart per k-cell.

More precisely, for $0 \leq k \leq n$, *k*-embedding labels are *manifolds* of dimension k, i.e., in dimensions 0, 1, 2, etc, respectively points, Jordan arcs, disc homeomorphic patches, etc. In Fig. 1, each vertex is 0-embedded in a point of $\mathbf{R}^3$, each edge as a straight line segment, each face as a plane patch, and each volume as a polyhedron. Note that the display often splits the cells, to show darts and sewings, as in Fig. 1 and Picture 1.

2.3 Properties of the Generalized Maps

Many interesting *properties* about the topology of objects can be obtained from a modeling by n-g-maps. The principal ones concern 2-g-maps which modelize surfaces. Indeed, with a given 2-g-map may always be associated the topology of a *subdivision* of a surface, open or closed, orientable or not, with or without boundaries. Conversely, a 2-g-map may always be associated with the topology of any surface subdivision [27, 33].

Three *characteristics* of a 2-g-map are typical of the subdivision of the surface whose topology is represented by the 2-g-map. They are the *number of boundaries*, the *orientability factor* (the value of which is 0 if the surface is orientable, 1 or 2 otherwise), the *genus* (i.e., the number of holes) [27]. For instance, the triplet of characteristics of a disk, a sphere, a torus, a Moëbius strip, a Klein bottle, respectively are (1, 0, 0), (0, 0, 0), (0, 0, 1), (1, 1, 0), (0, 2, 0). These characteristics can be computed from a 2-g-map associated with the surface. Picture 2 shows non orientable open and closed surfaces.

3 Functionalities of the Modeller

The modeller is a program which allows interactively to construct, modify, manipulate, display and record geometric 3D objects, thanks to numerous original facilities.

3.1 Modeled Objects

The end user of the modeller manipulates embedded 3-g-maps with some additional constraints. They modelize objects composed of vertices, edges, faces and volumes, which are *regularized* or *not* [45]. Thus, dangling edges, open faces or volumes, non orientable faces can intentionally be manipulated, efficiently detected and possibly discarded.

More precisely, the following additional *topological constraints* are adopted for the 3-g-maps handled by the user: (1) α_0 is an involution with no fixed point; (2) $\alpha_0 \circ \alpha_1$ has no fixed point; (3) $\alpha_0 \circ \alpha_2$ has no fixed point; (4) $\alpha_1(x) = x$ implies $\alpha_3(x) = x$; (5) $<\alpha_0, \alpha_1>(x)$ and $<\alpha_0, \alpha_1>(\alpha_3(x))$ are disjoint for any dart x.

The intuitive meaning of these constraints is: (1) *dangling* darts are forbidden; (2) *loops* are forbidden; (3) *bent* edges, i.e., simple edges 2-sewn to themselves, are forbidden; (4) only *closed* simple faces can be sewn; (5) *folded* simple faces, i.e., simple faces 3-sewn to themselves, are forbidden. The 3-g-map in Fig. 1 satisfies these properties.

A very simple *embedding* is adopted: each vertex is embedded as a point of $\mathbf{Q}^3$. Thus, an edge is implicitly embedded as the *line segment* joining the two points associated with the vertices that bound the edge. Similarly, a face is implicitly embedded as a *surface patch* joining the line segments associated with the edges which bound the face. If the face is bounded by three edges, the patch is planar. Otherwise its exact form is not essential in our application. Such a patch, which is only defined by its boundary, is easily handled by most basic graphics libraries, for instance to fill it or to render it.

3.2 High-level Operations

About 150 high-level operations can be used through menus. Creation, deletion, duplication, subdivision, sewing and unsewing of k-cells and connected components of embedded 3-g-maps are among the main supplied facilities. Production and sewing of linearized cubic curves, bicubic surfaces, tricubic volumes, revolution and sweeping curves and surfaces are very useful operations offered for building quickly realistic objects.

Motion and deformation with constant topology of designated parts of an object are possible under an interactive control. Immediate computing of the three characteristics of built surfaces allows the user to permanently check their quality. Visualization, coloring, picking, files, undo-redo, press book, etc, complete the functionalities.

Despite the simplicity of the topological and embedding models, it is possible to build complex objects with these operations, as displayed in Pictures 3 to 8 of Appendix. In traditional modellers, such objects sometimes need *Boolean shape operations* [41], which are brute force and time consuming. The clear distinction between topology and embedding in the geometrical model often allows us to avoid these operations. Moreover this distinction is a basic principle of the following formal specifications.

4 Algebraic Specification of an N-g-map Basic Kernel

We formalize a kernel of n-g-maps manipulations. It is more general than what is strictly needed for the modeller in order to make specification, subsequent programmation and future re-using more easy, clear and safe. In this section, the specification is mainly an *horizontal structuring* by *extensions without parameter*, and without explicitly *importing/exporting modules* [18]. The specification is *order-sorted* [23] and *equational*, with the usual *if_then_else_* and == (for comparison) polymorphic functional symbols [17]. We always adopt an *initial semantics* point of view [17, 23]. Specifications of Booleans, non null natural, natural and rational numbers, i.e., ***Boolean***, ***nznat***, ***nat*** and ***rat*** sorts, are built-in, with the *constraint* to be in the *initial* semantics [18].

4.1 Topology

The dart sort is named *dart*. An erroneous dart *err_dart* is introduced with the *dart?* sort as such that *dart* < *dart?*, in the sense of the ordered sorts [23]. We choose atomic n-g-map *generators* to facilitate the building of other operations by composition, particularly sewings and unsewings of k-cells. That leads to sorts which are *greater* than necessary and which are called by the generic name *map*. The specification of the most general sort, *map0*, begins by that of the topological generators, *v*, *i*, and *l*:

```
sorts dart dart? map0
subsort dart < dart?
operations
        err_dart : → dart?
        v : → map0
        i : map0 dart → map0
        l : map0 nat dart dart → map0
```

For simplification's sake, we identify here *dart* sort and ***nznat***, *err_dart* and 0, and *dart?* and ***nat***. This enables us to stay in the *initial semantics* avoiding both parameters and functorial considerations [17]. In fact, darts are actually represented by pointers in the modeller implementation. The symbol of constant *v* corresponds to the empty map, *i(m, x)* inserts a dart *x* in a map *m*, *l(m, k, x, y)* associates *y* to *x* with label (in *dimension*) *k*, in a "semi-sewing". These operations are total. So, we can insert by *i* several times the same dart in a map, and apply *l* anyhow. In initial semantics, the sort *map0* is interpreted as the set of all the closed terms which are generated by these operators.

The following operator *a* formalizes the n-g-map functions α_k (cf. section 2) extended to the maps: *a(m, k, x)* returns the *successor* at dimension *k* of *x* in the map *m*:

```
operation
        a : map0 nat dart → dart?
axioms (m : map0, k j : nat, x y z : dart)
        a(v, k, z) = err_dart
        a(i(m, x), k, z) = if z == x then x else a(m, k, z)
        a(l(m, j, x, y), k, z) = if k == j and z == x then y else a(m, k, z)
```

To retrieve the usual cases, we define a new *map1* sort, with *map1* < *map0*, interpreted as a set of *directed multigraphs* on darts, with edges labelled in **nat**. The invariant *inv_map1* characterizes the maps of *map0* which belong to *map1*. The operation *i(m, x)* is interpreted as the insertion of a graph node *x* in *m*, and *l(m, k, x, y)* as the addition of an edge (*x*, *y*) labelled by *k*. The Boolean operation *ex_dart(m, x)* returns **true** iff *x* is in *m*:

```
sort map1
subsort map1 = map0 [inv_map1]
operations
      ex_dart : map0 dart → Boolean
      inv_map1 : map0 → Boolean
preconditions (m : map1, k : nat, x y : dart)
      prec [map1] i(m, x) ≡ not ex_dart(m, x)
      prec [map1] l(m, k, x, y) ≡ ex_dart(m, x) and ex_dart(m, y)
axioms (m : map0, k n : nat, x y z : dart)
      ex_dart(v, z) = false
      ex_dart(i(m, x), z) = z == x or ex_dart(m, z)
      ex_dart(l(m, k, x, y), z) = ex_dart(m, z)
      inv_map1(v) = true
      inv_map1(i(m, x)) = not ex_dart(m, x) and inv_map1(m)
      inv_map1(l(m, k, x, y)) = ex_dart(m, x) and ex_dart(m, y)
                                and inv_map1(m)
```

Preconditions for *i* and *l* derive from the invariant *inv_map1*: when applied to *map1 m*, *i(m, x)* returns a legal element of *map1* iff *x* does not belong to *m*, and *l(m, k, x, y)* iff *x* and *y* belong to *m*. Each precondition definition is indexed by the target sort of the operator between square brackets, e.g., *[map1]* for *i*. In fact, the above preconditions are the *weakest ones* to satisfy the invariant, which can be proven. The new operations *i* and *l* are restrictions of the old ones: they have exactly the same semantics as previously on the domain defined by the invariant. This satisfies the *monotonicity condition* of [23].

A *hierarchy* of about 40 ordered map sorts with invariants and operations with preconditions has been defined. Constraints get more and more restrictive: sorts with α_k injective, with α_k a permutation, with α_k an involution, with $k \leq 3$, with embedding constraints, etc. For instance, the sort *map3* take into account the *permutativity* of *i* and *l*:

```
sort map3
subsort map3 < map2
axioms (m : map2, k k' n : nat, x y x' y' z : dart)
      i(i(m, x), x') = i(i(m, x'), x)
      l(l(m, k, x, y), k', x', y') = l(l(m, k', x', y'), k, x, y)
      i(l(m, k, x, y), z) = l(i(m, z), k, x, y)
```

The sort hierarchy meets the order-sorted signature constraints: it is *regular* and *coherent*, in the sense of [23]. It is completely described in [4].

4.2 Embedding

We extend the specification with an *embedding generator em*. Thus, *em(m, k, x, q)* embeds the *k*-cell of *m* containing dart *x* on a geometric object *q* of sort *embed* and of dimension *k*:

```
sort embed
operation
      em : map0 nat dart embed → map0
precondition (m : map0, k : nat, x : dart, q : embed)
      prec em(m, k, x, q) ≡ inv_embed(q, k)
```

To meet this constraint, we introduce sorts *embedk* of geometric objects of dimension *k*, with *embedk* < *embed*. We also define an invariant *inv_embed(q, k)*, **true** iff *q* belongs to sort *embedk*, which is the precondition for *em*. In fact, only 0-embeddings are explicit in the modeller. They are 3D points, i.e., triplets of **rat** coordinates, built by the 0-embedding generator *genembed0*. Thus we have in this simple case:

> ***operations***
>   *genembed0 :* ***rat***^{3} *→ embed0*
>   *inv_embed : embed →* ***Boolean***
> ***axiom*** *(x y z x1 :* ***rat****, k :* ***nat****)*
>   *inv_embed(genembed0(x, y, z), k) = k == 0*

The introduction of the map generator *em* enriches the set of *map0* terms. It also forces to complete the previous specifications. For instance, a new axiom is needed for *a*:

> ***axiom*** *(m : map0, k j :* ***nat****, x z : dart, q : embed)*
>   *a(em(m, k, x, q), j, z) = a(m, j, z)*

Such axioms can always be automatically added, and wont be written here anymore. In fact, as there are thousands in the modeller specification, it is impossible to write them by hand. They are called *implicit axioms* in [37]. As previously, only darts present in a map can be embedded. This property restricts the *map0* sort to a new sort *mape1* < *map0* Furthermore, for the modeller, *at most one* dart per *k*-cell is *k*-embedded. Thus, we introduce again a new sort *mape2* < *mape1*, defined by an invariant *inv_mape2*, in fact *free_embed(m, k, x)*, that is **true** iff no dart of the *k*-cell of *x* is *k*-embedded in *m*:

> ***sort*** *mape2*
> ***subsort*** *mape2 = mape1 [inv_mape2]*
> ***axiom*** *(m : mape1, k :* ***nat****, x : dart, q : embed)*
>   *inv_mape2(em(m, k, x, q)) = free_embed(m, k, x)*

We immediately deduce from this invariant the weakest precondition of operator *em* for the *mape2* sort. At this level, the precondition for *l* is more tricky to obtain: *l(m, k, x, y)* is legal w.r.t. the embedding constraints only if, for all $j \neq k$, the *j*-cells of *x* and *y* are still the same ones, or one of them is not *j*-embedded. The auxiliary operation *free_embedl(m, k, x, y)* returns **true** iff this condition holds. Its formal specification is not given here.

4.3 Geometric Operators

We impose to the final objects of the modeller *exactly one* k-embedding per k-cell. A new sort *mape3* < *mape2* derives. To maintain the *mape3* invariant, operator *l* might be called by a new operator *ll* which removes some embeddings to avoid two distinct k-embeddings for a k-cell. Thus, operation *ll(m, k, x, y)* removes, if necessary, the embeddings of the cells which contains *y*, and *k*-sews *x* and *y* by *l(l(m, k, x, y), k, y, x)*. Precisely, before the *k*-sewing, for all $j \neq k$, it calls a function *llk(m, j, x, y)* which removes if necessary the *j*-embedding associated with the *j*-cell containing *y*.

Symmetrically, function *rr(m, k, x)* first unsews in *m* darts *x* and *y = a(m, k, x)*, then duplicates if necessary the *j*-embeddings for all $j \neq k$ if *x* is *j*-embedded in *m*. It calls for all $j \neq k$ a function *rrk(m, k, j, x, y)* which duplicates the *j*-embedding if necessary.

4.4 Sort Hierarchy and Initial Semantics

This leads us to a rather complex map *sort hierarchy*, with the top level *map0*. Several branches go down: one for topology, with ... *map2* < *map1* < *map0*, one for embedding, with ...< *mape2* < *mape1* < *map0*, and other ones for cell *markers* (cf. Section 5). Note that new map generators are always introduced at top level, with automatical repercussions on the lower levels by implicit axioms. The *3-g-map* sort of the embedded 3-g-maps with the modeller constraints of Section 3 is at the bottom of the hierarchy. In an initial semantics, it can be interpreted as the *intersection* of all its upper sorts, in other words, as the set of the *map0* terms which exactly satisfy the invariants of the whole hierarchy [4].

With *order-sorted algebras* [23] it is easy to describe step by step numerous geometric models, starting from a unique model with atomic generators. *Invariants* and *preconditions* help to easily limit ordered geometric sorts, as pointed out by [25]. Intermediate models correspond to intermediate states when building geometric objects [13]. They facilitate defining total operations. They can also be used in modeling as such, when extending the class of handled objects beyond the strict 3-g-maps [4].

However, our specifications might be considered as *over-specified* [6]. It is the case with any specification where generators are chosen, sometimes rather early for practical reasons, because some implementations and algorithms are favoured. In fact, an early decision about the choice of the generators allows us to properly define the sort hierarchy and to make the kernel runable, which is essential for debugging.

Moreover, our aim is that the last level of specification be isomorphic with the implementation. That is an additional reason why, contrary to other approaches, as for instance *loose semantics* [46], we have favoured an *initial semantics* approach during the whole development of the specifications.

The properties of the 3-g-maps, for instance α_k being an involution, for $0 \leq k \leq 3$, α_0 being an involution with no fixed point, $\alpha_0 \circ \alpha_1$ having no fixed point, etc, can be obtained from the specification as *inductive theorems*. That agrees with the initial semantics. These theorems can be proven by a mechanized inductive reasoning [14, 15].

5 Vertical Development

The previous operators are combined to obtain new ones. Their definitions can be given first at a high level of abstraction, appropriate for logical prototyping, but often inefficient for a realistic implementation. The new operators have to be *refined* in a progressive *vertical development*. Our aim is to keep the same specification framework for this refinement. We examine this process through an example.

We specify operator *cc* of cell traversal: *cc(m, x)* extracts the connected component of *m* of sort *map1* containing dart *x*. We start with a short abstract definition *à la Kruskal* using other operators, the specification of which is simple and only given in comments:

```
operation
    cc : map1 dart → map1
precondition (m : map1, x : dart)
    prec cc(m, x) ≡ ex_dart(m, x)
axioms (m : map1, k : nat, x y z : dart)
    cc(i(m, x), z) = if x == z then i(v, z) else cc(m, z)
    cc(l(m, k, x, y), z) = if islink(l(m, k, x, y), z, x)
                           then if islink(m, x, y) then l(cc(m, z), k, x, y)
                                else l(union(cc(m, x), cc(m, y)), k, x, y)
                           else cc(m, z)
/* union(m1, m2) merges m1 et m2;
 islink(m, x, y) is true iff the connected components of x and y in m are the same ones */
```

A straight implementation of *cc* as specified (a logical prototyping) will always have a time complexity exponential w.r.t. *p*, the number of darts in the map (in the best, average and worst cases). In fact, an acceptable response time imposes a complexity of *cc* in *O(p)* in the worst case.

We improve this situation by a refinement of the specification. Thanks to a new *map0 generator of dart marking*, named *mark*, we specify a *depth-first* traversal *mkcc(m, x)*, which marks the connected component of *x* in *m* instead of extracting it. The complexity in maximum time of this new operator is in *O(p)*, when supposing an *O(1)* complexity for the implemented basic operators, which is true in the actual implementation of the modeller. To simplify, we directly pass to a new sort of marked maps, named *mapc*, such that *mapc* < *map1*. The objects of this sort meet the following constraint of *strict connectivity*: existence of a path from a dart *x* to a dart *y* implies existence of another one from *y* to *x*. This property also holds for n-g-maps, so *3-g-map* < *mapc* < *map1*:

```
operation
        mkcc : mapc dart → mapc
precondition (m : mapc, x : dart)
        prec mkcc(m, x) ≡ ex_dart(m, x) and not_mark(m)
axioms (m m1 : mapc, n : nat, x : dart)
        mkcc(m, x) = mkcc_aux(mark(m, x), dim(m), x)
        with mkcc_aux(m, n, x) = if n < 0 then m else mkcc_aux(m1, n - 1, x)
                with m1 = if ismark(m, a(m, n, x)) then m  else mkcc(m, a(m, n, x))
/* mark(m, x) marks the dart x in m;
   ismark(m, x) is true iff dart x is marked in map m;
   not_mark(m) is true iff no dart of m is marked;
   dim(m) gives the dimension of map m */
```

For a final non recursive implementation, we refine once more the specification into another one, which is *tail recursive*, and directly implementable in *O(p)* time. Actually, it is easier to realize this traversal in a *breadth-first* way, the queue of marked darts being explicitly linked in the map itself thanks to another *map generator*, called *markcc*:

```
axioms (m m0 m1 m2 : mapc, n : nat, x  x0 y z  z1 t t0 t1 t2 : dart)
        mkcc(m, x) = mkcc_list(m0, x0, t0)
        with (m0, t0) = mkcc_aux(markcc(m, x, x), dim(m), x, x)
                x0 = succ_markcc(m1, x)
                mkcc_list(m, z, t) = if z == z1 then m else mkcc_list(m1, z1, t1)
                  with z1 = succ_markcc(m, z)
                         (m1, t1) = mkcc_aux(m, n, z1, t)
                         mkcc_aux(m, n, z, t) = if n < 0 then (m, t)
                                                          else mkcc_aux(m2, n - 1, z, t2)
                       with  (m2, t2) = endmarkcc(m, t, a(m, n, z))
/* succ_markcc(m, x) is the  successor of  dart x  in the queue of marked darts;
   endmarkcc(m, t, x) takes dart x after dart t in this queue */
```

Note that the *map0* sort enrichment by new mark generators is only realized with the last version of the generators, i.e., in the example, with *markcc*. In fact, when we go down the sort hierarchy, for instance fixing $n = 3$, or imposing that some involutions be without fixed points, we can write specialized traversal functions still more efficient than the ones above. Thus, an edge traversal in a 3-g-map can be realized by direct compositions of *a*, for dimensions 2 and 3, without explicit depth- or breadth-first traversal. Finally, through factorizing definitions by **with** and using iterators (cf. Section 6), the specification can be directly translated into a procedural language.

6 Horizontal Structuring

6.1 Sewing Operators

At *3-g-map* level, we can write functions to create, remove, sew and unsew k-cells. We distinguish between *topological operators*, with strong topological and embedding preconditions, and *geometrical* (i.e., topological *and* embedding) *operators* of higher level, that only have topological preconditions. Particularly, sewing operators which meet the 3-g-maps constraints of the modeller can be defined this way.

We present three examples: topological sewing, *se*, geometric sewing, *gse*, and insertion, *gie2*, of two simple edges, with only the formal specifications of *se* and *gse*:

```
operations
      se, gse : 3-g-map dart dart → 3-g-map
preconditions (g : 3-g-map, x y : dart)
      prec gse(g, x, y) ≡ ex_dart(g, x) and ex_dart(g, y) and a(g, 2, x) == x
                                      and a(g, 2, y) == y and x ≠ a(g, 0, y) and x ≠ y
      prec se(g, x, y) ≡ prec (gse(g, x, y)) and free_embedl(g, 2, x, y)
                                   and free_embedl(g, 2, a(g, 0, x), a(g, 0, y))
```

axioms *(g : 3-g-map, x y : dart)*
se(g, x, y) = L(L(g, 2, x, y), 2, a(g, 0, x), a(g, 0, y))
with *L(g, k, x, y) = l(l(g, k, x, y), k, y, x)*
gse(g, x, y) = se(llk(llk(llk(llk(g, 1, x, y), 3, x, y), 0, x, y),
0, a(g, 0, x), a(g, 0, y)), x, y)

Operator *gse* is written efficiently by using, instead of the general operator *ll*, operator *llk* at the appropriate dimensions to remove superfluous embeddings (cf. subsection 4.3). Operation *gie2(g, x, y)*, which uses *se*, is rather surprising. It allows us to subdivide one face or to merge two faces, depending on whether the faces of *x* and *y* are the same ones or not (Fig. 4, (a) and (b), and Pictures 9 and 10 in Appendix).

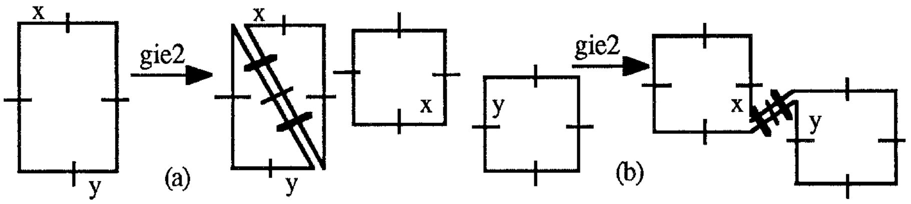


Fig. 4. Insertion of two simple sewn edges.

In the modeller, several useful operators are defined the same way, in particular *Euler operators*, which keep the surface genus unchanged [39].

6.2 Meshes, Second Order and Iterators

We now briefly present the specification of a high-level operator, namely a *mesh contructor* [4]. The interest lies in the way this operator is specified. It clearly distinguishes between topology and embedding, and reduces the required number of operations. Moreover, it makes possible to justify the introduction of second order functions and iterators in specifications. We propose here to create simple meshes, which can be completed by poles afterwards (Fig. 5, Picture 11, and Picture 12 for an extension).

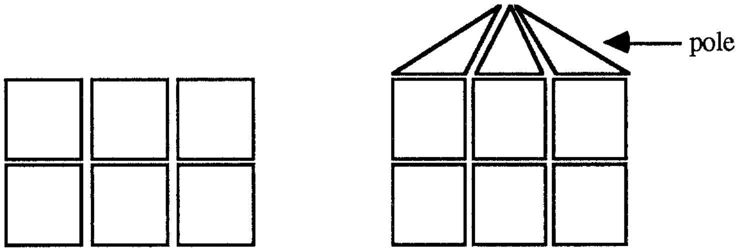


Fig. 5. A simple mesh and a mesh with pole (the 0 and 2-sewings are omitted).

A *simple mesh* is made of simple quadrangles sewn together by operator *se*. The mesh topology is progressively created and embedded thanks to a function *f*: **nat**^{4} → *embed* which 0-embeds a dart per quadrangle. The 4 variables of *f* are *n* and *p*, which give the mesh size, and $i \in [0, n]$ and $j \in [0, p]$, which give the columns and lines of the mesh points.

The mesh function, denoted *grid*, uses auxiliary creations of squares, i.e., *sqrgrid*, *sqrgridl1*, *sqrgridnl* and *sqrgridlp*, whose specification is omitted (cf. [4] for details):

operation
grid : 3-g-map ***(nat***^{4} *→ embed)* ***nat***^{2} *→ 3-g-map*
axioms *(g : 3-g-map, f :* ***nat***^{4} *→ embed, i j k n p :* ***nat****)*
grid(g, f, n, p) = grid2(grid1(sqrgridl1(g, f, n, p), f, 2, n, p), f, 2, 2, n, p
with *grid1(g, f, k, n, p) =*
if *k > n* ***then*** *g* ***else*** *grid1(sqrgridnl(g, f, k, n, p), f, k + 1, n, p)*
grid2(g, f, i, j, n, p) =

```
if i > n then if j ≥ p then g else grid2(sqrgrid1p(g, f, j, n, p), 2, j + 1, n, p)
else grid2(sqrgrid(g, f, i, j, n, p), i + 1, j, n, p)
```

Second order functions, like *grid*, are out of the strict frame of traditional algebraic specifications. They could be avoided by using parameterized modules as in OBJ3 [24]. But it is a heavy solution when the parameters concern only a few functions as it is here the case, where, moreover, the semantics does not create any problems.

The use of auxiliary functions *grid1* and *grid2* to simulate two embedded iterations on the mesh squares by a *tail recursion* makes the specification heavy and reduces its readability. To remove them, we use *iterators*, considered as macro-definitions. For instance, *v = v0 ;* ***while not*** *cond(v)* ***do*** *succ(v)* defines by induction the last value *v* of a sequence of values, where the initial, current and next ones are respectively *v0*, *v* and *succ(v)*, and the stop condition *cond(v)*. It textually replaces the following *iter* function defined by: *v = iter(v0)* ***with*** *iter(v) =* ***if*** *cond(v)* ***then*** *v* ***else*** *iter(succ(v))*.

The following specification which redefines *grid* with two embedded iterators, simplifies the specifications and brings us closer to functional and procedural languages:

```
axioms (g g1 g2 gj : 3-g-map, f : nat⁴ → embed, i j  k n p : nat)
    grid(g, f, n, p) = g2
    with  (i, g2) = (2, g1) ; while i ≤ n do (i+1, gj)
               with  (j, gj) = (2, sqrgrid1p(g2, f, j, n, p)) ;
                            while j ≤ p do (j+1, sqrgrid(gj, f, i, j, n, p))
                   (k, g1) = (2, sqrgrid11(g, f, n, p)) ;
                            while k ≤ n do (k+1, sqrgridn1(g1, f, k, n, p))
```

6.3 Miscellaneous: User Interaction, Errors, Libraries

Classical algebraic specifications do not conveniently deal with *user interactions* [37] which can be seen as a distinct process, concurrent with the modeller process. A formalism suited to the description of concurrency, like LOTOS, could be used. In our case, it is a heavy solution, since the entry of external events is strictly limited to a small specification part. We have prefered to describe user interactions by a *meta-specification*, i.e., a second level of specifications, which manipulate the first level ones by simple axiom modifications [4].

In a modeller, *errors* are linked to user interaction. Operators with preconditions have been implemented in the way they were specified, i.e., without testing the preconditions in their bodies. They have been encapsulated in high-level functions, algebraically specified too, which deal with computer-human interactions, preconditions and errors.

Today, graphical software development makes an intensive use of built-in graphical *libraries*. We have formally specified the library we use (Silicon Graphics GL), by an arbitrary choice of basic generators and by expressing other operators w.r.t. them. Such a task was difficult, given the imperfections of the documentation.

7 Some Technical Features of the Modeller

The modeller includes about 1800 operations: 1400 for geometric modeling and 400 for the environment and the interface. Most of them, except for 100 mathematical or trivial functions, were developed from complete algebraic specifications. The operations of the basic kernel were translated into PROLOG and tested by *logical prototyping* [4]. Only 150 parameterized operations are visible for the user through the interface.

The modeller is currently implemented in C for Silicon Graphics workstations. Thanks to the map concepts, the data structures that are used are very simple. Except for the upper ones, all the map and dart sorts of the hierarchy are really implemented with *the same unified C types*. A unified map structure consists mainly of a linked linear list of similary dart records, the fields of which are more or less constrained, depending on the invariants of the corresponding map sort. Each dart record contains 1 dart pointer to the successor in the list, 4 dart pointers to represent the k-sewings, for $0 \leq k \leq 3$, one pointer to the optional 0-embedding, and several markers for traversal and interactive selections. The program contains about 25 000 lines of C, a rather small number for software of this type. This is mainly due to the conciseness and reusability induced by algebraic techniques.

Response time is comparable with those of commercial products with similar features and may be even better. The main operations, particularly the traversals, have a time complexity in $O(p)$, where p is the number of darts. This is not the case in many commercial modellers which often take $O(p^2)$, because they lack directly accessible topological data. Other classical data structures, such as those derived from the "winged edge" [2], often take comparable memory space for a more restricted modeling area.

8 Conclusion

The n-g-map is a general and efficient model that helps to describe and handle the topology of n-dimensional objects. With embeddings, this model can be implemented in any dimension. For example, we chose to derive 3-g-maps with 0-embedding for a geometric modeller of complex manifold objects embedded in Q^3. Other choices could be made with other map extensions and embeddings. But the handling of *non manifold* objects [45] requires even more general mathematical concepts, cellular complexes [19] for instance.

With the above algebraic specification and its ordered sorts and invariants, we obtain a formal, functional, hierarchized, and homogeneous description of objects and operations. As often [22], genericity has not been necessary. In the future for instance, it may become interesting, to parameterize the modeller by topological or embedding constraints. Finally, we expect to recover a specification language well suited to the needs above [7].

Proofs of *correctness* of our horizontal structurings are possible and will be undertaken, with a mechanical help [24]. But, the proofs of correctness of our vertical developments, which involve techniques of program transformation and synthesis, are more difficult to bring into play. Moreover, work about using algebraic specifications in semi-automatical proofs of geometric properties is in progress [14, 15].

The concrete results of our study is an interactive modeller of 3D objects. Private firms are already interested by its functionalities. However, it must be adapted and extended in order to be used in real applications. Algebraic specifications give a basis which makes those extentions easier. Among the next fundamental ones, there is the cellular complexes [19] and the Boolean shape operators [41].

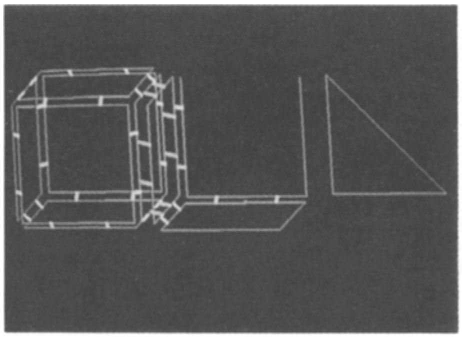
Our experiment illustrates how algebraic specifications help to elaborate and study models in geometry, and to design and develop software in this area. Conversely, the spectacular workbench of geometric modeling is convenient to understand how algebraic specifications improve axiomatization and programming processes in a complex field.

References

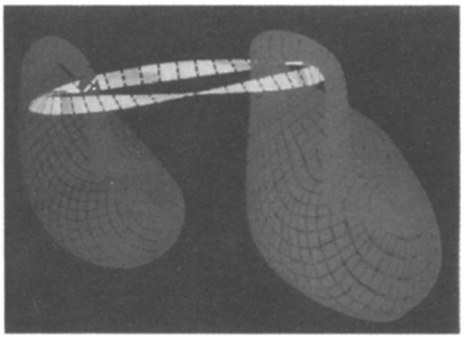
1. Baudelaire, P and Gangnet, M: Planar Maps: An Interaction Paradigm for Graphic Design. *Proc. CHI'89* (1989) 313–318
2. Baumgart, B: A Polyhedron Representation for Computer Vision. *Proc. AFIPS Nat. Conf.* Proc. 44 (1975) 589–596
3. Bergstra, J A, Heering, J and Klint, P: *Algebraic Specification*. ACM, Addison-Wesley (1988)
4. Bertrand, Y: Spécification Algébrique et Réalisation d'un Modeleur Interactif de Subdivisions Tridimensionnelles. *Thèse de Doctorat*, CRI-ULP, Strasbourg (1992)
5. Bertrand, Y, Dufourd, J-F, Françon, J et Lienhardt P: Modélisation Volumique à Base Topologique. *Proc. Micad*, Paris (1992)
6. Bidoit, M: Development of Modular Specification by Stepwise Refinements using the PLUSS Specification Language. *Tech. rep. LIENS* 91-9 (1991),
7. Bidoit, M, Kreowski, H-J, Lescanne, P, Orejas, F and Sannella, D: *Algebraic System Specification and Development*. LNCS n°501, Springer-Verlag (1991)

8. Bryant, R and Singerman, D: Foundations of the Theory of Maps on Surfaces with Boundaries. *Quart. Journal of Math. Oxford* Vol 2 n° 36 (1985) 17–41
9. Cori, R: Un code pour les graphes planaires et ses applications. *Astérisque* n° 27 (1975)
10. Duce, D A , Fielding, E V C and Marshall, L S: Formal Specification of a Small Example based on GKS. *ACM Trans. on Graphics* Vol 7 n° 3 (1988) 180-197
11. Dufourd, J-F, Gross, C and Spehner, J-C: A Digitisation Algorithm for the Entry of Planar Maps. *Proc. Comp. Graphics Int.* Leeds, Springer-Verlag (1989) 649-662
12. Dufourd, J-F: Algebraic Map-Based Topological Kernel for Polyhedron Modelers. *Proc. Eurographics*, Hamburg, Elsevier (1989) 301-312
13. Dufourd, J-F: Formal Specification of Subdivisions using Hypermaps. *Computer Aided Design*, Butterworth-Heinemann Vol 23 n° 2 (1991) 99-116
14. Dufourd, J-F: An OBJ3 Functional Specification for the Boundary Representation. *Proc. ACM-Siggraph Symp. on Solid Modeling Foundations & CAD/CAM Appl.*, Austin (1991) 61-72
15. Dufourd, J-F: Foundations of Boundary Representation Revisited with a New Foremap Axiomatics. *Proc. Eurographics Work. on Formal Spec. in Comp. Graphics*, Marina di Carrara (1991)
16. Edmonds, J: A Combinatorial Representation for Polyhedral Surfaces. *Not. AMS* Vol 7 (1960)
17. Ehrig, H and Mahr, B: *Fundamentals of Algebraic Specifications Vol 1 : Equations and Initial Semantics*. Springer-Verlag (1985)
18. Ehrig, H and Mahr, B: *Fundamentals of Algebraic Specifications Vol 2 : Module Specifications and Constraints*. Springer-Verlag (1990)
19. Elter, H et Lienhardt, P: Extension de la Notion de Carte pour la Représentation de la Topologie d'Objets Géométriques Complexes. *Proc. Journées Gros-Plan*, Lille (1991)
20. Franklin, W R, Wu , P Y F & Samaddar, S: Prolog & Geometry Projects. *IEEE CG & A* Vol 6 (1986)
21. Gangnet, M, Hervé, J-C, Pudet, T and Van Thong, J-M: Incremental Computation of Planar Maps. *ACM Computer Graphics* Vol 23 n° 3 (1989) 345–354
22. Gaudel, M-C: Structuring and Modularizing Algebraic Specifications. *R. 01-92* LRI, Orsay (1992)
23. Goguen, J A & Meseguer, J: Order-Sorted Algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions, and Partial Operations. *Tech. Rep.n° 89-10* SRI-CSL (1989)
24. Goguen, J A and Winkler, T: Introducing OBJ3. *Tech. Rep. n° 88-9*, SRI-CSL Menlo Park (1988)
25. Goguen, J A: Modular Algebraic Specification of Some Basic Geometrical Constructions. *Artificial Intelligence* Vol 37 (1988) 123-153
26. Grant, E, Amburn, P and Whitted, T: Exploiting Classes in Modeling and Display Software. *IEEE CG&A* Vol 6 (1986) 13-20
27. Griffiths, H-B: *Surfaces*, Cambridge Univ. Press, Cambridge (1981)
28. Jacques, A: Constellations et Graphes Topologiques. *Coll. Math. Soc. J. Bolyai* (1970) 657–672
29. James, L: Maps and Hypermaps: Operations and Symmetry. *PhD thesis*, Dep. of Mathematics, Univ. of Southampton (1985)
30. Lakshminarasimhan and A L, Srivas, M: A Framework for Functional Specification and Transformation of Hidden Surface Elimination Algorithms. *CG. Forum* Vol 8 n° 2 (1989) 75-98
31. Lienhardt, P: Free-Form Surfaces Modeling by Evolution Simulation. *Proc. Eurographics*, Nice, Elsevier (1988) 327–341
32. Lienhardt, P: Extension of the Notion of Map and Subdivision of Three Dimensional Space. *Proc. STACS*, Bordeaux, *LNCS* Vol 294, Springer-Verlag (1988) 301-311
33. Lienhardt, P : Subdivisions of Surfaces and Generalized Maps. *Proc. Eurographics*, Hamburg, Elsevier (1989) 439–452
34. Lienhardt, P: Subdivisions of N-Dimensional Spaces and N-Dimensional Generalized Maps. *Proc. 5° ACM Symp. on Comp. Geometry* , Saarbrücken (1989) 228–236
35. Lienhardt, P: Topological Models for Boundary Representation : A Comparison with N-dimensional Generalized Maps. *Comp.-Aided Design* Vol 23 n° 1, Butterworth-H. (1991) 59–82
36. Mallgren, W R: Formal Specification of Graphic Data Types. *ACM TOPLAS* Vol 4 n°4 (1982) 687-710.
37. Mallgren, W R: Formal Specification of Interactive Graphics Programming Languages. *ACM Distinguished Dissertation*, MIT Press (1982)
38. Mäntylä, M and Sulonen, R: GWB : A Solid Modeler with Euler Operators. *IEEE CG & A* Vol 2 n° 7 (1982) 17-31
39. Mäntylä, M: *An Introduction to Solid Modeling*. Computer Science Press, Rockville (1988)
40. Parsons, M S: Image Representations Using Miranda Laws. *Comp. Graphics Forum* Vol 8 n°2, North-Holland (1989) 99-106
41. Requicha, A: Representations for Rigid Solids : Theory, Methods and Systems. *ACM Computing Surveys* Vol 12 n° 4 (1980) 437–464
42. Requicha, A A G and Voelker, H B: Solid Modeling: Current Status and Research Directions. *IEEE CG & A* Vol 3 n° 7 (1983) 25-37
43. Tutte, W: *Graph Theory*. Encyclop. of Mathematics and its Applications, Addison–Wesley (1984)
44. Vince, A: Combinatorial Maps. *J. of Combinatorial Theory* Series B n° 34 (1983) 1–21
45. Weiler, K: The Radial Edge Structure: A Topological Representation for Non-Manifold Geometric Boundary Modeling. in *Geometric Modeling for CAD Applications*, Elsevier (1988) 3-36
46. Wirsing, M: Algebraic Specification. *Handbook of Theoretical Computer Science, Vol 2 : Formal Models and Semantics*, Elsevier (1990) 675-788

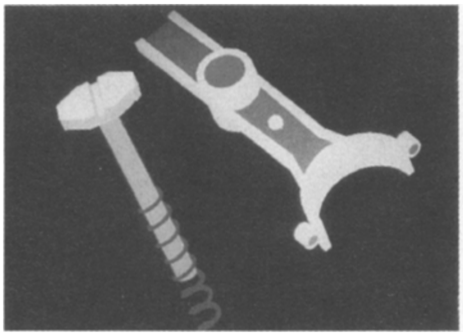
Appendix



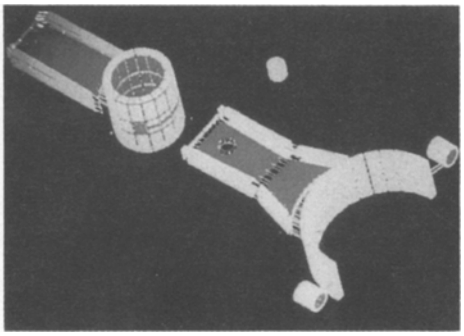
Picture 1: The 3-g-map of Figure 1



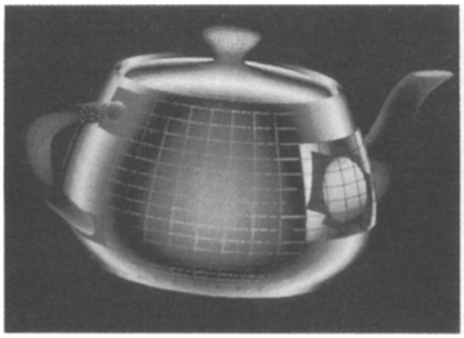
Picture 2: A Moëbius band and 2 Klein bottles



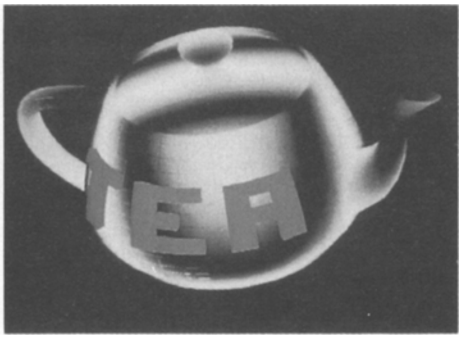
Picture 3: Mechanical parts



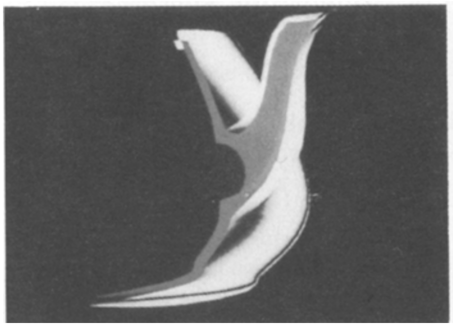
Picture 4: Details of a splitted mechanical part



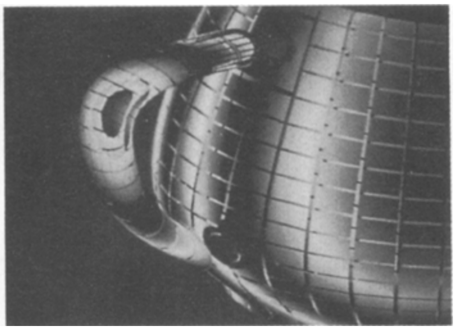
Picture 5: A surfacic teapot



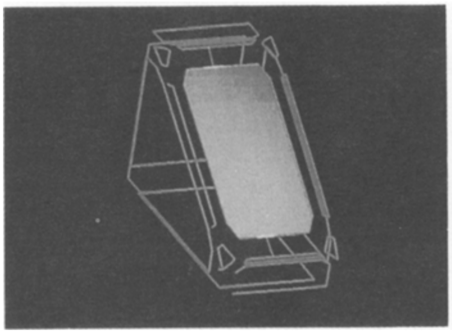
Picture 6: A volumic teapot



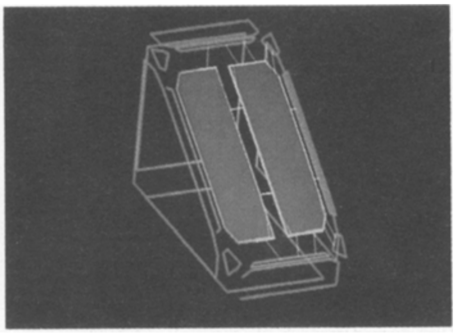
Picture 7: Spout of the volumic teapot



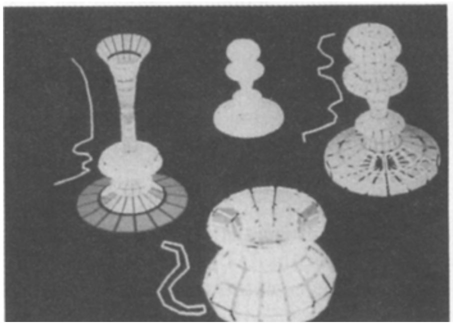
Picture 8: Handle of the volumic teapot



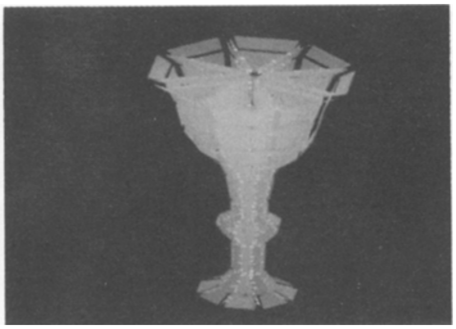
Picture 9: A face to be split



Picture 10: The splitted face



Picture 11: Surfacic meshes



Picture 12: Volumic meshes