•

Oriented combinatorial maps and extensions – Specification, prototyping, applications*

Jean-François Dufourd

Laboratoire des Sciences de l'Image, de l'Informatique
et de la Télédétection (UPRES-A CNRS 7005),
Université Louis-Pasteur de Strasbourg,
Pôle API, rue S. Brant, 67400 Illkirch, France
e-mail: dufourd@dpt-info.u-strasbg.fr
www: http: //lsiit.u-strasbg.fr

Abstract: We propose a formal specification based on a functional prototype of the oriented combinatorial maps embedded in the plane and revisit classical problems in computational geometry. We first recall the basic properties of the maps and their modelling power for closed orientable surfaces. Then, we mainly focus on logical, topological and geometric aspects of algorithms rather than on efficiency and round-off error questions. All the descriptions are written into the Ocaml language and tested. Progressively enriched by topological and geometric features and operations, the prototype is a basis of discussion before doing a real efficient implementation in a computational geometry library. The paper is illustrated by some high-level operations, particularly a Delaunay triangulation. Finally, we present and discuss some extensions in higher dimensions, particularly the generalized maps and their features. We show how, at dimension 2, they model the topology of general surfaces, closed or open, orientable or not.

1 Introduction

We present a formal specification based on a prototype of a basic software module devoted to the definition and the manipulation of oriented combinatorial maps embedded in the Euclidean plane, and their applications to classical problems in computational geometry. Then we present some extensions which offer larger geometric modelling capabilities, particularly the generalized maps.

The algebraic notion of oriented combinatorial map is today widely recognized as a precise, reliable, flexible and economical tool to describe and handle the topology of a subdivision, in vertices, edges and faces, of a closed and orientable surface. With such a surface of any given *genus*, i.e. number of holes, corresponds exactly a unique (up to an isomorphism) oriented combinatorial map, with the same genus, this notion being combinatorially defined. When the genus is null, the map is said *planar*.

Moreover, when the subdivision of a surface with non null genus is embedded into a plane, the corresponding oriented combinatorial map necessarily contains overlaps, i.e. confused vertices, vertices included in edges, or crossing edges. Conversely, when the genus is zero, it is always possible to embed the map into the plane without overlap, by what is called a *planar embedding*. Nevertheless a planar map can also be embedded in a non planar way, i.e. with overlaps.

* This research is supported by the *GDR-PRC Algorithmique, Langages et Programmation*, MENRT, CNRS, France.

Thus, when dealing with oriented combinatorial maps on a computer, it is worth allowing to build and manipulate maps which are either planar or non planar, planarly embedded or not. Moreover, it is essential to find quickly the main characteristic features of a map to be updated so as to pass safely to another kind of map.

In the paper, we first describe maps with no embedding, i.e. in a purely topological, or combinatorial, way. Besides mathematical, pictorial and informal explanations, basic operations are specified algebraicly by using a functional language, namely *Ocaml*, a member of the *ML* family. This way, we avoid any ambiguity, incompleteness or inconsistency. Moreover, the definitions are naturally prototyped and tested to eliminate conceptual errors.

In fact, most operations on oriented combinatorial maps are easier to define on a more general algebraic structure called quasi-map in [DP99]. This notion is the basis of formal proofs on maps by induction, particularly for results on planarity, Euler characteristic and genus [PD00]. Some basic explanations are given in the present paper to understand and use it properly, but a more complete and comprehensive study can be found in [DP00].

In a second step, a real size implementation with efficient concrete data structures will be done to add to a software library of computational geometry, CGAL [FGK+98] for instance. Some indications are given about the implementation, but the paper does not deeply study this question. Therefore, nothing can be said immediately about algorithmic complexity from our specifications. The round-off error problem is another main subject missing in this work. We consider that it is complicated enough to devote to it specific complementary studies later. Moreover, since [Tut63], numerous works have been devoted to the enumeration, the drawing or the random generation of different kinds of maps [CR72, Arq85, AB97, Jac97, JS98, Sch98]. But, all these questions are out the scope of this paper.

Finally, to complete our presentation, we discuss some extensions of the map notion, which has been proposed to enlarge their modelling power for general surfaces or topologic spaces in higher dimensions. We particularly focus on the generalized maps, which, at dimension 2, allow to support and control the topology of subdivisions of surfaces of all kinds, open or closed, orientable or not.

The structure of the paper is the following. Section 2 presents basic mathematical notions. Section 3 defines the algebraic structure of oriented combinatorial map and shows some of its uses in topology. Section 4 develops a basic functional specification of extended maps in *Ocaml*. Section 5 specifies operations to build and manipulate real maps. Section 6 proposes some operations of traversal and equivalence in maps. Section 7 gives extensions to handle map characteristics and planarity. Section 8 goes back to the so-called Euler operators. Section 9 introduces plane embeddings and other attributes. Sections 10 proposes some operations to handle together topological and geometric map notions. Section 11 introduces the question of map duality, with some peculiar problems. Section 12 illustrates our framework by a complete example of Delaunay triangulation. Section 13 deals with other high-level operations. Section 14 gives some indications about a subsequent efficient implementation of maps and operations. Section 15 present extensions at dimension d: hypergraphs, maps and above all generalized maps and their properties. Section 16 focuses on the modelling power of the generalized maps at dimension 2 for the surfaces of all kinds and Section 17 concludes. A basic knowledge of functional programming and of a representative of the *ML* languages family makes the understanding easier, but it is not absolutely necessary.

2 Permutations

Defining maps requires some elementary mathematical notions.

Notation Let α and β be applications and z an element. In this paper, application of α to z is written in prefix notation, αz or $\alpha(z)$, and composition of α and β is denoted $\alpha\beta$: $\alpha\beta z = \alpha\beta(z) = \alpha(\beta(z))$. In the whole section, D is a finite set.

Definition 1 Let α: $D \to D$ be an application. The couple (D, α) has the algebraic structure of a direct functional graph, whose D is the set of nodes and α the set of arcs. An element z is a fixpoint for α if $\alpha z = z$. Finally, α is called an *injection* in D if each element of D has at most one predecessor by α.

Example In Figure 1, $D = [1..9]$ and $\alpha = (1\ 5)(5\ 2)(2\ 4)(4\ 1)(7\ 3)(3\ 8)(8\ 7)(6\ 6)$, written in the classical notation for applications defined in extension, is an injection in D. Note that 6 is a fixpoint for α.

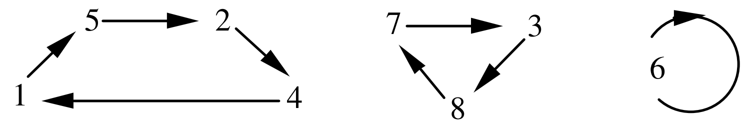


Figure 1: The direct functional graph of injection α

Definition 2 An application α: $D \to D$ is a *surjection* in D if each element of D has at least one predecessor, and it is a *bijection* in D if it is both an injection and a surjection in D. In fact, since D is finite, a surjection in D is necessarily an injection, and conversely, an injection in D is always a surjection, therefore it is a bijection in both cases. An application α: $D \to D$ is *involutive* if $\alpha\alpha z = z$ for each z in D. Such an application is necessarily a permutation which is called an *involution*.

Proposition 1 Let α be a permutation in D. Each connected component of the directed graph (D, α) is an elementary simple circuit, i.e. a path not trespassing twice by the same arc or node, except for its origin and extremity which are confused.

Example In Figure 1, (D, α) has three connected components, or circuits, denoted by a circular list of their elements following the arrows of α: (1 5 2 4), (7 3 8) and (6).

Definition 3 Let α be a permutation in D. The *inverse* α^{-1} of α is a permutation in D defined for each z by: $\alpha^{-1}z$ = predecessor of z by α

Example The inverse of α in Figure 1 is $\alpha^{-1} = (5\ 1)(2\ 5)(4\ 2)(1\ 4)(7\ 8)(8\ 3)(3\ 7)(6\ 6)$ in Figure 2, with inversed arrows.

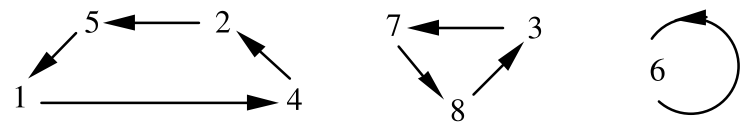


Figure 2: Graph of permutation α^{-1}, inverse of α in Figure 1

Finally, the composition $\alpha\beta$ of two permutations α, β: $D \to D$ is a permutation in D as well.

3 Oriented combinatorial maps

The first time it was thought to use an algebraic structure to describe the topology of polyhedra seems to be in [Edm60]. Moreover, since [Tut63], numerous mathematical studies have been devoted to enumerate *topological maps* of different kinds, particularly planar maps, for many surfaces. Then, the notion of *constellation*, as a pair of permutations independent from an embedding, has been introduced in [Jac70]. It has been often promoted under the appellation of *combinatorial map* and generalized under different forms, for instance in [Cor75, Vin83a, Vin83b, BS85, Spe90, Lie91]. In [Tut84], the constellation is studied under the name of *oriented combinatorial map*, which we take up. In fact, it has exactly the same geometric modelling power as the "winged edge" [Bau70], but is more precisely and concisely defined in an algebraic way.

Definition 4 A (*2-dimensional*) *oriented combinatorial map* is a 3-tuple $M = (D, \alpha_0, \alpha_1)$, where D is a finite set of elements which are called *darts*, α_0 an involution in D, and α_1 a permutation in D. Let k be 0 or 1 and $z \in D$. The darts $\alpha_k z$ and $\alpha_k^{-1} z$ are respectively the *k-successor* and *k-predecessor* of z. They are said to be *k-sewn* to z by *k-sewings*.

To make short, we often write only *map* for *2-dimensional oriented combinatorial map*.

Example Figure 3 shows a map M with $D = [1..18]$, $\alpha_0 = (1\ 2)(3\ 4)(5\ 6)(7\ 8)(9\ 10)(11\ 12)(13)(14\ 15)(16)(17\ 18)$ and $\alpha_1 = (1\ 6\ 3)(2\ 4\ 12)(5\ 9\ 7)(8\ 11\ 10)(13)(14\ 17\ 16)(15)(18)$.

It is embedded in the plane with the following conventions. Each dart labels a half-segment of curve delimited by a black point and a small dash. A dash between two half-segments represents a 0-sewing between their labels. The half-segments are ordered counter-clockwise around the black points depending on their succession by α_1. Darts 13 and 16 are fixpoints for α_0, and 13, 15 and 18 for α_1.

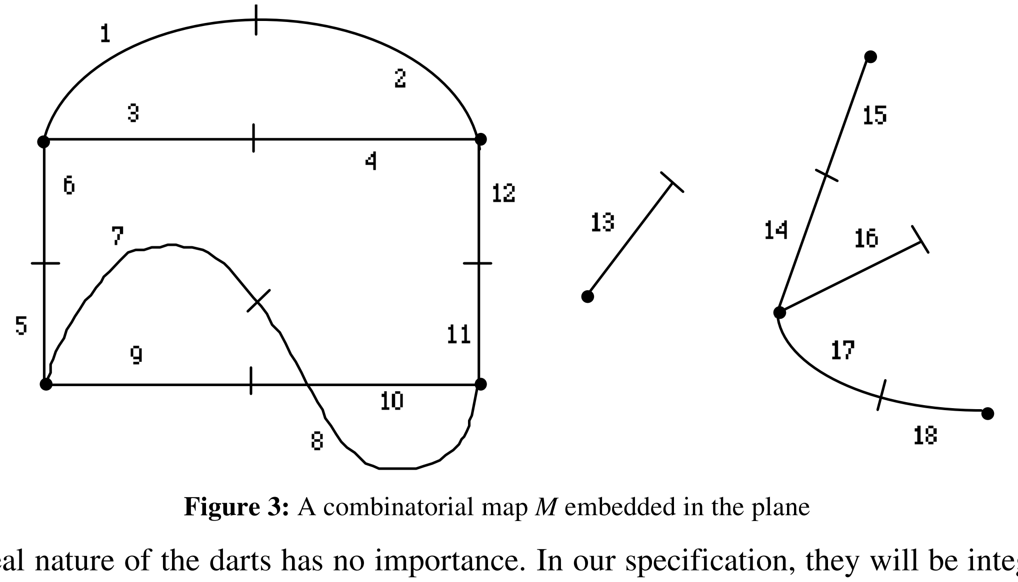


Figure 3: A combinatorial map M embedded in the plane

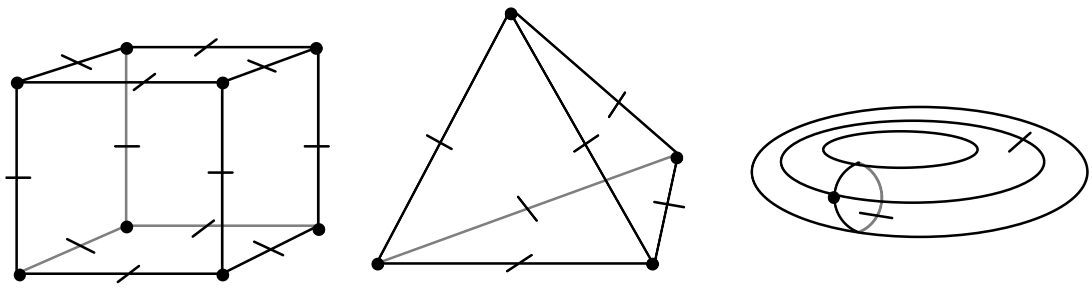
The real nature of the darts has no importance. In our specification, they will be integers, but in an efficient implementation, they may be *pointers* on data records. Note that, for d • 3, d-dimensional combinatorial maps can be defined the same way with d-1 involutions or permutations [Vin83a, Lie91, Duf91a]. In all the following, $d = 2$ implicitly. Moreover, in our map definition, fixpoints are allowed for α_0 or α_1, as in the original proposition of [Jac70], but those for α_0 are often prohibited. Thus, oriented combinatorial maps with no fixpoints for α_0 allow us to model the topology of subdivisions of orientable and closed surfaces, i.e. the boundary of usual solids [Edm60, Tut84, Lie91].

Example Figure 4(a) shows a unique non connected map embedded in the 3D space. Each connected component determines a subdivision of an orientable and closed surface limiting a volume: a cube, a tetrahedron and a torus. In Figure 4(b), this map is embedded in the plane, without vertex or edge overlaps for cube and tetrahedron, but necessarily with an overlap for the torus. This is related with the genus of a surface, as we will see.

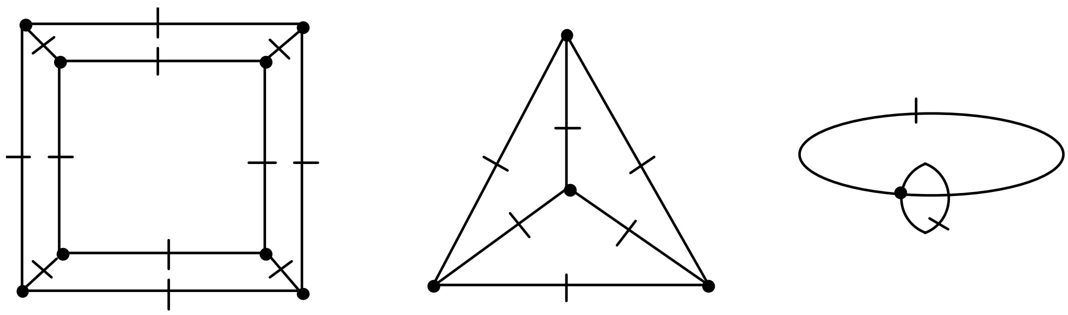
Thanks to the notion of *orbit*, maps make easy to express definitions of and access to the usual cells of subdivisions:

Definition 5 Let D be a set and Π a set of applications in D. Connected components of the functional multigraph (D, Π) are called *orbits* for Π. The orbit for Π that contains the element z of D is denoted $<\Pi>(z)$. Let $M = (D, \alpha_0, \alpha_1)$ be a map. Orbits for $<\alpha_0>$ are the 0-*orbits* or *edges* of M, those for $<\alpha_1>$ its 1-*orbits* or *vertices*, those for $<\alpha_1^{-1}\alpha_0>$ its *direct faces*, those for $<\alpha_1\alpha_0>$ its *backward faces*, and those for $<\alpha_0, \alpha_1>$ its *connected components*. The *face* of z is the ordered pair $(<\alpha_1^{-1}\alpha_0>(z), <\alpha_1\alpha_0>(\alpha_0 z))$ composed of the direct face of z and the backward face of $\alpha_0 z$.

Example In the map of Figure 3, when reducing an orbit to its support set: $<\alpha_0>(1) = \{1, 2\}$, $<\alpha_0>(6) = \{5, 6\}$ and $<\alpha_0>(16) = \{16\}$ are edges; $<\alpha_1>(8) = \{8, 11, 10\}$, $<\alpha_1>(17) = \{17, 16, 14\}$ are vertices; $<\alpha_1^{-1}\alpha_0>(3) = \{3, 2\}$, $<\alpha_1^{-1}\alpha_0>(5) = \{5, 1, 12, 8, 9, 11, 4, 6, 7, 10\}$ are direct faces; $<\alpha_1\alpha_0>(4) = \{4, 1\}$, $<\alpha_1\alpha_0>(5) = \{5, 3, 12, 10, 7, 11, 2, 6, 9, 8\}$ are backward faces; $<\alpha_0, \alpha_1>(17) = \{17, 18, 15, 16, 14\}$ is a connected component.



(a) A non connected map embedded in the 3D space



(b) The same map embedded in the plane

Figure 4: Map and embeddings

As a consequence of Proposition 1, vertices, edges, direct faces and backward faces form circuits in their respective graphs (D, α_0), (D, α_1), (D, $\alpha_1^{-1}\alpha_0$) and (D, $\alpha_1\alpha_0$). For instance, vertex $<\alpha_1>(8)$ can be written (8 11 10) with parentheses. Note that any cell, or orbit, of a map may be uniformly designated by the corresponding permutations and overall, by a unique dart.

4 Basic functional specification of maps

Several propositions for an algebraic specification of maps or extensions appear in [Duf89, 91a, 91b, BDF+93, BD94, BD98, Duf97, DP00]. In the present paper, we directly use a functional language, namely *Ocaml* [Ler98], to build a rapid and executable prototype able to support experimentations easily.

The basic functional map specification is presented in Table 1a. The symbols `dart` and `dim` are defined, after the keyword `type`, as the types for darts and dimensions. In the simple prototype presented, they are just synonymous with `int`, the built-in type of integers. Besides, a map is an object of type `map` defined using one of three *induction templates* headed by the *constructors* `V`, `I`, `L`, and separated by `|`. Let `q` be a map, `x,y` two darts and `k` a dimension. Then, `V` is the void, or empty, map (with no darts or sewings), `I(q,x)` the map built by inserting in `q` the new dart `x`, and `L(q,k,x,y)` the map obtained from `q` by `k`-linking `x` to `y`, in such a way that `y` becomes the `k`-successor of `x`, and `x` becomes the `k`-predecessor of `y`. In fact, freely applying these constructions leads to objects which can be more general than what is desirable for maps. The constructors must be used only when they respect preconditions given later. Note that integer `0` will play the role of a void dart.

Example The two small connected components of the map in Figure 3 also form a map `q` which can be built from a map term `q0`:

```
let q0 = L(L(L(L(I(I(L(I(I(L(L(I(I(V,14),15),0,14,15),0,15,14),13),16)
   ,1,13,13),17),18),1,14,17),1,16,14),0,18,17),1,15,15);;
let q = L(L(L(L(q0,0,13,13),1,17,16),0,17,18),1,18,18),0,16,16);;
```

A lot of map *observers* are defined on the `map` structure after `let`, the recursion being signalled by `rec`. Let `q` be a map, `z` a dart and `j` a dimension. Then, `empty q` tests if `q` is the `V` map by returning a Boolean, `e q z` tests if `z` exists in `q`, `fk q j z` (resp. `fk_1 q j z`) tests if `z` is free of explicit `j`-successor (resp. `j`-predecessor) by `L` in `q`. Note the principle of *structural induction* in the definition of these operations with a pattern-matching `match ... with ...` of `q` successively with the three `map` templates separated by `|`, the result being after `->`. Finally, `||` and `&&` denote Boolean disjunction and conjunction and _ an anonymous variable.

Moreover, `a q j z` (resp. `a_1 q j z`) returns `0` when `z` has no `j`-successor (resp. `j`-predecessor) explicitly constructed by `L`, otherwise returns the `j`-successor (resp. `j`-predecessor) of `z` in `q`, i.e. $\alpha_j z$ (resp. $\alpha_j^{-1} z$) in Definition 4, and `phi q z` returns, when there is one, the successor of `z` in its directed face of `q`, otherwise `0`.

Table 1a: Basic map specification (begin)

```
type dart = int;;
type dim = int;;
type map = V  | I of map * dart | L of map * dim * dart * dart;;

let empty q =
   match q with
         V -> true
       | I(m,x) -> false
       | L(m,_,_,_) -> false;;

let rec e q z =
   match q with
         V -> false
       | I(m,x) -> z = x || e m z
       | L(m,_,_,_) -> e m z;;
```

```
let rec fk q j z =
   match q with
        V -> true
      | I(m,x) -> z = x || fk m j z
      | L(m,k,x,y) -> (k <> j || z <> x) && fk m j z;;

let rec fk_1 q j z = ... similar ...

let rec a q j z =
   match q with
        V -> 0
      | I(m,x) -> if z = x then 0 else a m j z
      | L(m,k,x,y) -> if j = k && z = x then y else a m j z;;

let rec a_1 q j z = ... similar ... ;;

let phi q z = let z0 = a q 0 z in if z0 = 0 then 0 else a_1 q 1 z0;;
```

Example For the previous `q0` and `q` maps, `e q0 13 = true`, `e q 19 = false`, `fk q0 1 18 = true`, `fk q 1 17 = false`, `a q0 0 16 = 0`, `a q 0 16 = 16`, `a q0 1 17 = 0`, `a q 1 17 = 16` and `a_1 q 1 13 = 13`.

Note that `fk q j z` always returns `false`, which is not the case for `fk q0 j z`, because `q` is *complete*, in the meaning that all the circuits for its edges and vertices are explicitly closed by sequences of `L`, whereas `q0` is not complete. The latter is called a *quasi-map* in [PD00, DP00] which do an extensive study of this structure.

Table 1b: Basic map specification (continuation)

```
let rec d q z =
   match q with
        V -> V
      | I(m,x) -> if z = x then m else I(d m z,x)
      | L(m,k,x,y) -> L(d m z,k,x,y);;

let rec r q j z =
   match q with
        V -> V
      | I(m,x) -> I(r m j z,x)
      | L(m,k,x,y) -> if j = k && z = x then m else L(r m j z,k,x,y);;
```

In Table 1b, two *destructors*, `d` and `r`, inverses of `I` and `L`, allow the dismantling of a map `q`. Thus, `d q z` deletes `z` from `q` and `r q j z` rips the `j`-sewing from dart `z` in `q`.

Example Complete deletion of edge `{17,18}`, with all incident sewings, from previous `q`, is realized by:

```
d (d (r (r (r (r q 1 17) 1 18) 0 17) 0 18) 17) 18
```

In order to build sound extended maps, i.e. according to Definition 4, and to observe them correctly, *preconditions* in the use of operations are given in Table 1c. They are written in our prototype only as comments between `(*` and `*)`. For instance, `prec L(m,k,x,y)` states that `k`-sewing `x` to `y` in `m` is allowed if and only if `x` and `y` both exist in `m`, `x` has no `k`-successor, `y` no `k`-predecessor, `k = 1`, or `k = 0` and either `x` has no `k`-predecessor and `y` no `k`-successor or `y` has `x` as `k`-successor (and `x` has `y` as `k`-predecessor).

Table 1c: Preconditions

```
(*
   prec I(m,x) = not e m x
   prec L(m,k,x,y) =
      e m x && e m y && fk m k x && fk_1 m k y && (k = 1 ||
      k = 0 && (fk_1 m k x && fk m k y || a m k y = x))
   prec a q j z = prec a_1 q j z = prec fk q j z =
      prec fk_1 q j z = (j = 0 || j = 1) && e q z
   prec d q z = fk q 0 z && fk q 1 z && fk_1 q 1 z
   prec r q j z = (j = 0 || j = 1) && e q z
*)
```

The top level of *Ocaml* allows one to run operations through questions (before `;;`) and answers (after `- :`). With `q0` as defined above, the following conversation can occur:

```
e q0 17;;
   - : bool = true
fk q0 1 17;;
   - : bool = true
a q 0 17;;
   - : dart = 18
d (r (r q0 0 18) 1 16) 16;;
   - : map = L(L(I(I(L(I(I(L(L(I(I(V,14),15),0,14,15)
             ,0,15,14),13),1,13,13),17),18),1,14,17),1,15,15);;
```

In the interpretation of a function defined by induction, the three `map` templates headed by `V`, `I` and `L` are sequentially examined until a *matching* succeeds on the term to reduce.

5 Specification of oriented combinatorial maps

In the full oriented combinatorial maps we specify now, all vertices and edges are closed by a sufficient number of `L` applications. Recall that fixpoints at dimension 0 are allowed, but could be prohibited. In Table 2a, three basic operations, roughly corresponding to `V`, `I` and `L`, help to construct oriented combinatorial maps: `v` is the void map, `i q x` inserts in map `q` a new dart `x` 0- and 1-sewed to itself (Figure 5(a)), and `l q j x y` `j`-sews `x` to `y` while restoring completely and consistently the `j`-permutation (Figure 5(b,c1,c2)). Note that, if `x` and `y` are in two different `j`-orbit, this operation merges the two `j`-orbits into a unique one (Figure 5(c1)), otherwise it splits the orbit in two new ones (Figure 5(c2)).

Table 2a: Oriented map specification

```
let v = V;;

let i q x = L(L(I(q,x),0,x,x),1,x,x);;

let l q j x y =
   if x = y && a q j x = x then q
   else let xj = a q j x and y_j = a_1 q j y
        in let q1 = r (r q j x) j y_j
        in L(L(q1,j,x,y),j,y_j,xj);;

let dm q z = d (r (r q 0 z) 1 z) z;;

let rm q j z = l q j z z;;
```

Conversely, `dm q z` deletes the isolated dart `z` from map `q` with all its sewings to itself, and `rm q j z` extracts `z` from its `j`-permutation, i.e. is just a particular case of `l`. All these functions are defined from the previous section specification, which plays the role of an *abstract implementation*.

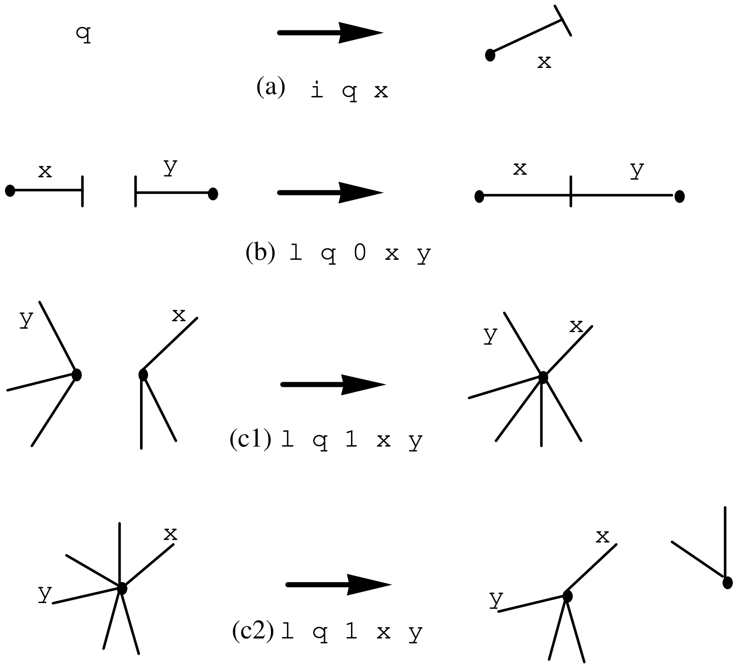


Figure 5: Basic operations on oriented combinatorial maps

A lot of secondary operations can be defined to simplify the user's work. Thus, Table 2b specifies `iil q x y`, `iib q x y` and `ile q x' y' x y` to insert in map `q` a new edge composed of two new darts `x` and `y`, i.e. a loop, a bar and a new edge 1-sewed to the ancient darts `x'` and `y'`, respectively (Figure 6(a,b,c))

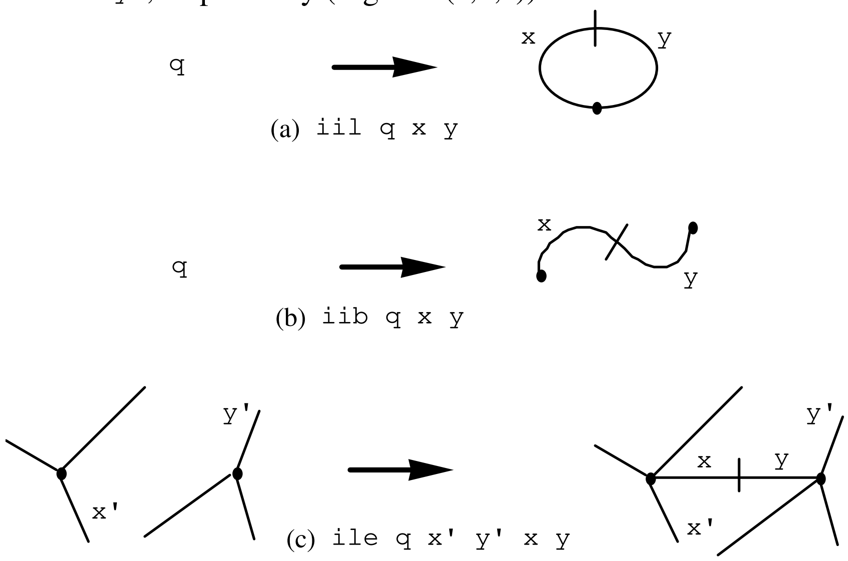


Figure 6: Secondary operations on oriented combinatorial maps (forms of the edges are not important)

Conversely, `de q z` deletes from `q` the edge `z` belongs to whatever its nature. Note that this operation could be specialized in accordance with its different hidded cases.

Table 2b: Map secondary operations

```
let iil q x y = L(L(L(L(I(I(q,x),y),0,x,y),0,y,x),1,x,y),1,y,x);;

let iib q x y = L(L(L(L(I(I(q,x),y),0,x,y),0,y,x),1,x,x),1,y,y);;

let ile q x' y' x y = l (l (iib q x y) 1 x' x) 1 y' y;;

let de q z =
   let z0 = a q 0 z
   in dm (dm (rm (rm (rm q 1 z) 1 z0) 0 z) z) z0;;
```

Preconditions are in Table 2c. Respecting them guarantees the map integrity, particularly the existence of an involution and a permutation at dimensions 0 and 1.

Table 2c: Preconditions for maps

```
(*
   prec i q x = not e q x
   prec l q j x y = (j = 0 || j = 1) && e q x && e q y
   prec iil q x y = prec iib q x y = not e q x && not e q y && x <> y
   prec ile q x' y' x y = not e q x && not e q y && x <> y
       && e q x' && e q y' && x' <> y'
   prec de q z = e q z
*)
```

This map axiomatics is *complete*, in the sense that all cases have been considered in the specification of the functions. That is due to a systematic use of the inductive definition method from the three map templates. It is also *consistent*, in the sense that no redundancy can occur in the interpretation of these definitions. In other words, there is no risk of *junk* or *confusion* of values in the types `bool`, `dart` or `dim` induced by the new definitions [EM85]. Finally, every observer or destructor is obviously finitely *terminating*, because, in each inductive specification template, a recursive call always acts on a shorter term. These good properties, which are preserved in what follows, allow to prove:

Proposition 2 Any oriented combinatorial map can be described by a term obtained only by composition of `v`, `i` and `l`, subject to the satisfaction of their preconditions. Conversely, such a term always describes a map.

Note however that a map can be described by several terms, which behave similarly [DP00]. Moreover, a function that tests whether an extended map is a real map can be easily defined by induction. Such a test can serve as a precondition of some subsequent operations, but, to simplify, it is not used in what follows.

6 Traversals and equivalences in maps

In maps, traversals and equivalences of darts can be specified. Thus, in Table 3a, `eq q j z t` (resp. `eqv q z t`, `eqe q z t`, `eqf q z t`, `eqc q z t`) is `true` iff `z` and `t` are in the same `j`-orbit (resp. vertex, edge, direct face, connected component) in map `q`. The definitions of `eq` and `eqf` use the auxiliary functions `eq1` and `eqf1`, which traverse an orbit, whereas `eqv` and `eqe` immediately derive from `eq` at dimensions `j = 1` and `j = 0`. The function `eqc q z t` is defined *à la* Kruskal as in a directed graph.

Moreover `epv q z t` (resp. `epe q z t`, `epdf q z t`) is `true` iff there is a path from `z` to `t` in a vertex (resp. edge, direct face) of extended map `q`, i.e. in the traversal from `z` with the succession function α_1 (resp. α_0, $\alpha_1^{-1}\alpha_0$) not defined everywhere in `q`. Finally, `chdart q` chooses a dart in the non void map `q`. All these functions are defined inductively in Table 3a, with an exception mechanism imposed by *Ocaml* for the latter. A function `epbf` similar to `epdf` could be defined for backward faces.

Table 3a: Traversals and equivalences

```
let rec eq1 q j z zlim t =
   if z = t then true
   else if z = zlim then false else eq1 q j (a q j z) zlim t;;

let eq q j z t = z = t || eq1 q j (a q j z) z t;;

let eqv q z t = eq q 1 z t;;

let eqe q z t = eq q 0 z t;;

let rec eqf1 q z zlim t =
   if z = t then true
   else if z = zlim then false else eqf1 q (phi q z) zlim t;;

let eqf q z t =  if z = t then true else eqf1 q (phi q z) z t;;

let rec eqc q z t =
   match q with
      V -> false
   |  I(m,x) -> z = x && t = x || eqc m z t
   |  L(m,_,x,y) -> eqc m z t ||
            eqc m z x && eqc m t y || eqc m z y && eqc m t x;;

let rec epv1 q z t =
   match q with
      V -> false
   |  I(m,x) -> z <> x && t <> x && epv1 m z t
   |  L(m,0,x,y) -> epv1 m z t
   |  L(m,_,x,y) -> epv1 m z t || epv m z x && epv m y t
and epv q z t = z = t || epv1 q z t;;

let epe q z t = z = t || a q 0 z = t;;

let rec epdf1 q z t =
   match q with
      V -> false
   |  I(m,x) -> z <> x && t <> x && epdf1 m z t
   |  L(m,0,x,y) -> epdf1 m z t ||
      let y_1 = a_1 m 1 y in (y_1 <> 0 && epdf m z x && epdf m y_1 t)
   |  L(m,_,x,y) -> epdf1 m z t ||
      let y_0 = a_1 m 0 y in (y_0 <> 0 && epdf m z y_0 && epdf m x t)
and epdf q z t = z = t || epdf1 q z t;;

exception Emptymap;;

let rec chdart q =  match q with
      V -> raise Emptymap
   |  I(m, x, _) -> x
   |  L(m, _, _, _) -> chdart m;;
```

Example In map `q` (small components of Figure 3), `eqv q 14 16 = eqe q 14 15 = eqc q 15 18 = true, eqc q 13 14 = false, eqdf q 18 16 = epv q 14 16 = epdf q 14 17 = true`, and in `q0, epdf q0 18 16 = true` but `epv q0 14 16 = epdf q0 14 17 = false`.

In fact, all these operations are classical in directed multigraphs and can be very efficiently programmed depending on the concrete data structure chosen for the implementation. Once again, preconditions are given in Table 3c, only for the main functions.

Table 3c: Preconditions for map traversals and equivalences

```
(*
   prec eqv q z t = prec eqe q z t = prec eqf q z t = prec eqc q z t =
     prec epv q z t = prec epe q z t = prec epdf q z t = e q z && e q t
   prec chdart q = not (empty q)
*)
```

7 Map characteristics and planarity

Map characteristics [Jac70, Tut84] are essential to further refine the control of the behaviour of modelled objects by topological considerations [Män88]:

Definition 6 *Euler characteristic ec* and *genus g* of an oriented combinatorial map are:

$$ec = nv - (nd - ne) + nf \qquad \text{and} \qquad g = nc - ec / 2,$$

where *nd*, *ne*, *nv*, *nf* and *nc*, are the numbers of its darts, edges, vertices, direct faces and connected components, respectively. A map with null genus is said to be *planar*.

Note that the number of direct faces is also the number of backward faces or the number of faces. Fortunately, the Euler characteristic is always an even integer, the limit of which depends on the theorem of the genus.

Theorem 3 (*of the genus*) The genus *g* of a map is a natural number.

Example The map in Figure 3 is non planar: *nd* = 18, *ne* = 10, *nv* = 8, *nf* = 4, *nc* = 3, *ec* = 4, and *g* = 1.

Let `q` be a `map`. In Table 3a, `nd q`, `ne q`, `nv q`, `nf q`, and `nc q` are the numbers of darts, edges, vertices, direct faces, and connected components of `q`. They can be entirely defined by induction on the extended map structure, without any preconditions. Finally, `ec q` and `g q` are the Euler characteristic and genus of `q`.

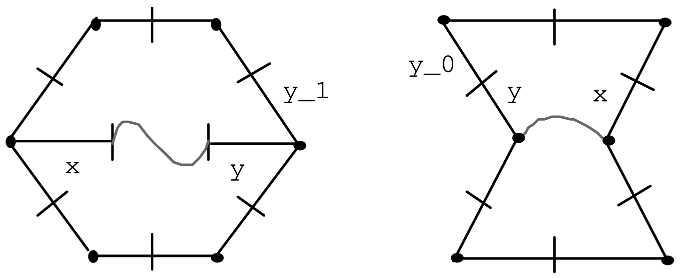
Table 4a: Specification of characteristics

```
let rec nd q =
   match q with
      V -> 0
    | I(m,x) -> nd m + 1
    | L(m,k,x,y) -> nd m;;

let rec ne q =
   match q with
      V -> 0
```

```
      | I(m,x) -> ne m + 1
      | L(m,0,x,y) -> if a m 0 y <> 0 && x <> y then ne m - 1 else ne m
      | L(m,_,x,y) -> ne m;;

let rec nv q =
   match q with
      V -> 0
      | I(m,x) -> nv m + 1
      | L(m,0,x,y) -> nv m
      | L(m,_,x,y) -> if epv m y x then nv m else nv m - 1;;

let rec nc q =
   match q with
      V -> 0
      | I(m,x) -> nc m + 1
      | L(m,k,x,y) -> if eqc m x y then nc m else nc m - 1;;

let rec nf q =
   match q with
      V -> 0
      | I(m,x) -> nf m + 1
      | L(m,0,x,y) -> let y_1 = a_1 m 1 y
           in if y_1 = 0 || epdf m y_1 x then nf m else nf m - 1
      | L(m,_,x,y) -> let y_0 = a_1 m 0 y
           in if y_0 = 0 || epdf m x y_0 then nf m else nf m - 1;;

let ec q = nv q - nd q + ne q + nf q;;

let g q = nc q - ec q / 2;;
```

Once again, proofs involving these operations could be made easier by induction. Map *planarity* is an important feature, often imposed for object correctness [GHP+89]. In order to preserve it during map manipulations, it is enough, for instance, to constrain operation `L` more strictly by a rather complex precondition given and completely justified in [PD98, 00], but out the scope of the present paper. Thus, `L` becomes a new operation called `L'`.

To preserve planarity as well, operations of full map construction are affected by this precondition. For instance, the planar version of `l` is named `l'`. Table 4b presents the new precondition of `l' q j x y`, i.e. the map sewing, imposing for planarity that `x` and `y_j = a_1 q (1 - j) y` belong to the same direct face, when `x` and `y` are connected. Intuitively, the meaning is that map planarity is guaranteed by sewings operating systematically *in* an oriented face. Figure 7(a,b) illustrates `l' q j x y` for `j = 0`, `1`.



(a) `j = 0` (b) `j = 1`

Figure 7: Planar j-sewings : `l' q j x y` (new sewings are in bent strokes)

Similarly, the planar version of `ile` is `ile'`. Table 4b also contains the new precondition for planarity of `ile' q x' y' x y`, the edge insertion and sewing, imposing `x'` and `y'` in the same direct face when they are connected. Obviously, `v` and `i` are planar operations, when they are equipped by their usual preconditions. Thus, we have the following result.

Proposition 4 An oriented combinatorial map is planar iff it can be described by a term built only by composition of `v`, `i` and `l'`, subject to the satisfaction of their preconditions.

Table 4b: Precondition to preserve planarity

```
(*
  prec l' q j x y = prec l q j x y
         && eqc q x y implies eqf q x (ac_1 q (1 - j) y));;
  prec ile' q x' y' x y = prec ile q x' y' x y
         && (eqc q x' y' implies eqf q x' y');;
*)
```

This important planarity criterion, which is very difficult to establish rigorously in a purely combinatorial way, has been recently illustrated and formally proved in the *Calculus of inductive constructions* with the aid of the *Coq* system [PD98, 00]. It is directly linked to a discrete Jordan theorem in the planar oriented combinatorial maps.

8 Euler operators

The so-called *Euler operators* have been described in [Män82, 88] to manipulate the topology of surfaces. In [Män82], ten Euler operators plus their inverses are presented. Most of them can be built by composition of our primitive constructors `V`, `I` and `L`. Thus, it is easy to verify formally that the primitives within these Euler operators are always used in cases where the preconditions we have fixed are satisfied. Other, like the so-called `kfmrh` and `mfkrh`, can not be defined in the same way because they involve faces which can have several disconnected borders. Thus, the geometric model underlying the Euler operators cannot be described by pure combinatorial oriented maps, but rather using face labels or maps extensions, like those in Sections 15-16.

As an example which works, the first five topological Euler operators of [Män82] are in Table 5. The first, `mvfs`, is exactly our primitive `i`. The second, `mev`, consists in adding a new dart `x` and to 0-sew it to an ancient dart `x'`. The third, `mef1`, is the building and sewing of an ordinary edge `{x,y}`. The fourth, `mef2`, is the building of a loop `{x,y}` whose vertex `x` is merged with another one `x'`, thus creating a new face. The fifth, `mekr`, is once again the building and insertion of an ordinary edge `{x,y}`, like `mef1`, but in the case where a *ring*, i.e. a loop, is killed, or in other words absorbed by another component.

Table 5: Euler operators

```
(* make vertex shell face *)
let mvfs q x = i q x;;

(* make edge vertex *)
let mev q x' x = l (i q x) 0 x x';;

(* make edge face 1 *)
let mef1 q x' y' x y = ile q x' y' x y;;

(* make edge face 2 *)
let mef2 q x' x y = l (iil q x y) x' x;;
```

```
(* make edge kill ring *)
let mekr q x' y' x y = ile q x' y' x y;;
```

In fact, the difference between `mef1` and `mekr` does not lie in their primary result on the map, but in their secondary effects on the sets of vertices, edges, faces and connected components, which is predicted from special preconditions. From our point of view, our three constructors `V`, `I` and `L`, or their completed versions `v`, `i` and `l`, with their inverses, are quite sufficient to cover all the needed updating. They do not require a strict control of the case where they are applied, while leading to the desired effects, which can be noticed *a posteriori* by the battery of observers. That is a comfortable loose position.

Thus, with our three operators only, we can totally control the building and updating of non planar topologies, for 3D objects with holes, or we can absolutely forbid them.

9 Embedding features

When dealing with *geometry*, a 2-dimensional map has to be *embedded* in a n-dimensional space, with $n = 2$ or not. For instance, in Section 3 (Figure 4), a 2-dimensional map is embedded in a 3D space. Form attributes can be associated with elements of any topological dimension k, 0 • k • d, in a k-embedding [BDF+93]. In the case where $n = 3$, vertices are 0-embedded on 3D points, edges are 1-embedded on 3D curves or Jordan arcs, and faces are 2-embedded on surface patches.

At these dimensions, the link to the topology is the following one. The genus of an oriented combinatorial map is also the genus, or the number of holes, of the surface modelled by the map. More precisely, we have the following result linking embedded maps to surfaces.

Theorem 5 (*Poincaré*) The maximum number of closed Jordan curves which can be drawn on a closed orientable compact surface of genus g without disconnecting it is $2g$.

Example On a sphere, of genus 0, a unique curve is disconnecting. On a torus, of genus 1, two curves can be drawn without disconnection (i.e. one going around the hole and the other in the hole), but three are always disconnecting.

In fact, the embedding of a connected oriented map in the plane can always be obtained by a *stereographic* projection from an embedding without overlap on a convenient surface. Thus, a planar map embedded on the sphere can be projected without overlap on the plane. A map of genus 1 embedded without overlap on a torus is projected in the plane necessarily with overlaps. That is illustrated in Figure 4.

In this paper, we only consider geometric embeddings for oriented combinatorial maps in an Euclidean plane. Then, it is possible to associate a point and an arc with each dart irrespective of the other embeddings. Conversely, with (all the darts of) a vertex can be associated a unique point and with (all the darts of) an edge a Jordan arc whose endpoints are the embeddings of the two darts of the edge. In the first case, the embedding is said *discontinuous*, and in the second *continuous*. We will only develop the second case, which is the more common. *Photogrammetric* attributes could also be introduced in the same way at any dimension: colours of vertices, edges, faces, thickness of lines, texture of faces, etc. Similarly, an attribute could be associated with an entire cell, but labelling is out the scope of the paper.

Moreover, we consider that the embedding of an edge is always the line segment which joins the two points 0-embedding the two darts of the edge. In this case of *linear* plane embeddings, the 1-embedding can remain implicit and we only need to embed vertices on points. In Table 6a, `point` is defined as the type for points in the plane, in fact the Cartesian product of `float` by itself, with `P` as a constructor. Thus a point always matches the template `P(p1,p2)`, where `p1,p2` are its abscissa and ordinate in a fixed Euclidean coordinate system, `origin` being the name given to `P(0.,0.)`. Note that the decimal writing of floating numbers in *Ocaml* has to contain a dot, therefore `0.` denotes zero. Functions `abs` and `ord` return coordinates `p1` and `p2` of a point `p` unified with `P(p1,p2)` and `pi` is an approximation of π.

A lot of classical geometric function could be defined at this level, however specifications involving geometric embeddings suppose exact computations on floating-point data, an hypothesis which is realistic only under some conditions whose pertinence is out the scope of the paper. Moreover, to solve the problem of sharing an embedding between darts, vertices or even maps, we introduce *point indexes* through type `ipoint`, which is here an alias of `int`. Then, the embedding of a vertex is realized by the 0-embedding on a point index of all its darts. The correspondence between indexes and points is realized thanks a global table, or point environment, of type `envp`.

Table 6a: Management of points

```
type point = P of float * float;;
let origin = P(0.,0.);;
let abs p =
   match p with P(p1, p2) -> p1;;
let ord p =
   match p with P(p1, p2) -> p2;;
let pi = 3.14159;;
...

type ipoint = int;;
type envp = Vp | Ip of env * ipoint * point;;
let rec ap e i =
   match e with
      Vp -> origin
   |  Ip(e1, j, p) -> if i = j then p else ap e1 i;;
let ep = ref Vp;;
let ip i p = ep := let t = !ep in Ip(t,i,p);;
let ae i = let t = !ep in ap t i;;

let apoint_n = ref 0;;
let genapoint () = apoint_n := !apoint_n + 1; !apoint_n;;
```

An object of type `envp` is just a bag built from the empty environment `Vp` by incremental insertions of pairs (index, point) using `Ip` constructor. Then `ap e i` returns the point corresponding to index `i` in environment `e`. A global reference `ep` to an environment is declared with managing functions. Thus, `ip i p` adds by *side effect* a new pair (index `i`, point `p`) to the environment `!ep` associated with `ep`, and `ae i` returns the point associated with index `i`. Finally, `genapoint ()` returns a new point index thanks to the updating of an integer referred by `apoint_n` with initial value `0`.

Of course, we are not any more in a *pure* functional programming. A specification of completely 0-embedded maps is given in Table 6b, with a new constructor `I` introducing at the same time a dart and its corresponding point index. The index of dart `z` is got back from map `q` by `aig q z`. Then, using implicitly the global point environment `ep`, `ag q j z` returns the point which j-embeds `z`, only `j = 0` being operational for the moment.

Table 6b: Embedded maps

```
type map = V
    | I of map * dart * ipoint
    | L of map * dim * dart * dart;;

let rec aig q j z =
   match q with
     V -> 0
    | I(m, x, p) -> if z = x then p else aig m j z
    | L(m, k, x, y) -> aig m j z;;

let ag q j z = ae (aig q j z);;

let rec ang q z  = ...
```

With this map definition, the previous specifications of functions by induction on the map structure have to be updated to take into account the new parameter of `I`. Finally, operation `ang q z` returns the oriented angle in [0, 2π[with the *x*-axis of the arc 1-embedding `z`, the arc being considered oriented with `z` as origin.

Preconditions presented in Table 6c constrain these operations. Owing to the choice of point association, embedding has an influence on topology. One of the main constraints introduced in an embedded map is of course that all the darts of each vertex are embedded on the same point index. For `k = 1`, the precondition of `L(m,k,x,y)` is completed with this aim by `aig m 0 x = aig m 0 y` and add nothing else for `k = 0`. Thus, it does not avoid possible loops, null edges, overlaps, self-intersections, angle inconsistencies or non planarity. These drawbacks can be removed at a second level, and, once again, `L'` can be used and constraint to keep the planarity (Section 7). Finally, by its precondition, `ang` has to be always computable, i.e. applied to a dart which is neither a 0-fixpoint nor an endpoint of an edge with null 1-embedding. Note however that the precondition of `L` does not guarantee against self-intersections and non planarity. Sewing constraints are detailed and discussed in the general case of hypermaps and general embeddings in [BD94].

Table 6c: Preconditions

```
(*
  prec L(m,k,x,y) = ancient prec
    && (k = 1 implies aig m x = aig m y)
  prec aig q z = prec ag q z = e q z
  prec ang q z = e q z && a m 0 z /= z && ag q O z /= ag q 0 (a q 0 z)
*)
```

Operations of higher level can be defined to handle and access map cells, i.e. edges, vertices, faces. The first group, we have already seen, contains *purely topological* operators, that do not manage embeddings, and the second one, we will present, *mixed* (*topological-embedding*) operators, which allow to act both on topology and embedding. These operations are specified by composition of low-level ones, with strong topological and embedding preconditions.

10 Topological-geometric map operations

Many operations can be designed for real maps linearly embedded in the plane, i.e. each vertex of which is 0-embedded on a 2D point index. A sample is presented in Table 7a and illustrated in Figure 8. First, `iidg q ip` inserts into `q` a new isolated dart 0-embedded on a point index `ip` and `iibg q ip ir` inserts into `q` a new isolated linear bar composed of two darts 0-embedded on `ip` and `ir` (Figure 8(a,b)). Both return the new map and generated darts.

Next, `msvg q x y` merges the vertices of `x` and `y` in `q` and restores the permutation of their darts in accordance with their angle order, provided that `x` and `y` have the same 0-embedding and are in two different vertices (Figure 8(c1)). When they are in the same vertex, the latter is split into a vertex containing the darts from `x` to `y` excluded in counter-clockwise order and into a vertex containing the other darts, the permutations and 0-embedding being correctly restored (Figure 8(c2)). In fact, in the first case, this operation can be profitably extended when `x` and `y` have different 0-embeddings [BDF+93, BD94]. Moreover, `ieg q x y` inserts in `q` a new linear edge, the darts `x'` and `y'` of which become the new 1-successors of `x` and `y`, provided no inconsistency arises in the angle ordering of the vertices (Figure 8(d)).

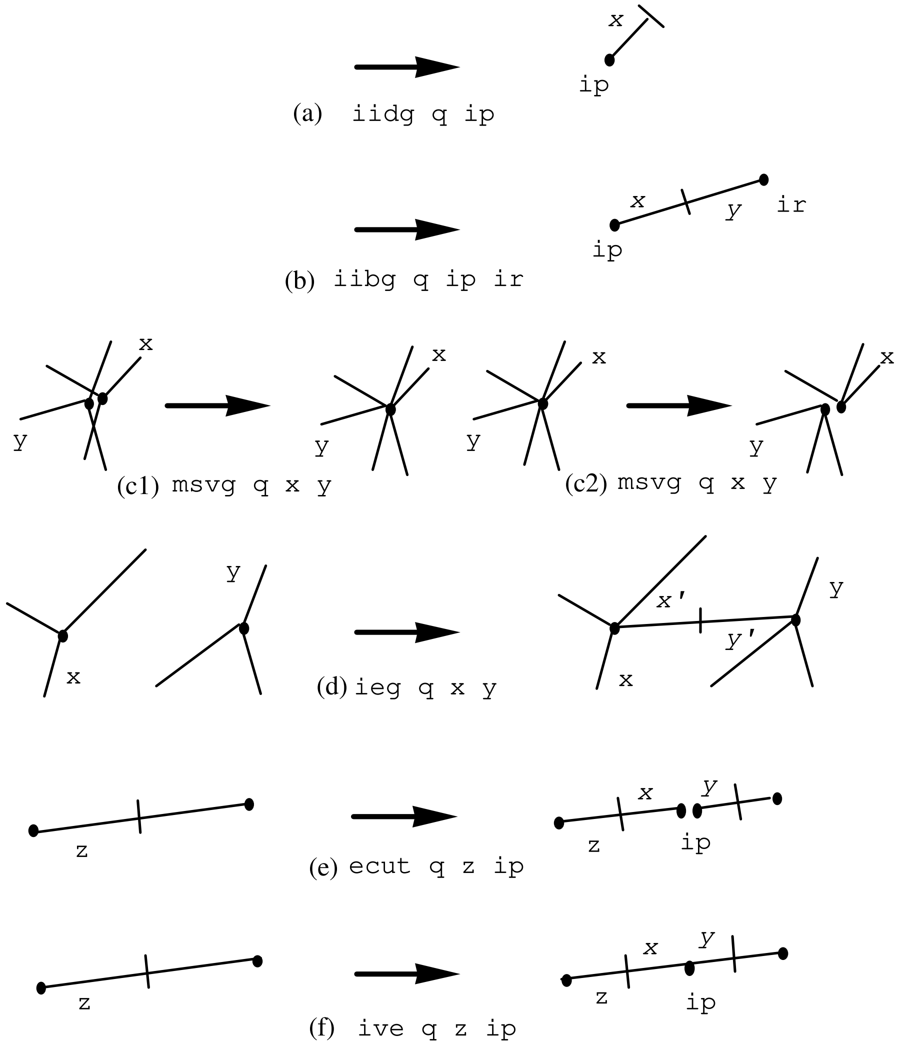


Figure 8: Topological-geometric operations

Note that, even if loops and null edges are avoided, these operations possibly enable *self-intersections*. Finally, `ecut q z ip` cuts an embedded edge `z` at a point index `ip` (Figure 6(e)), `ive q z ip` inserts a point index `ip` onto an embedded edge `z` (Figure 6(f)) and `deg q z` deletes the embedded edge `z`. In fact, the latter is the same as `de q z`.

Table 7a: Topological-geometric operations on embedded maps

```
let iidg q ip =
   let x = gendart ()
   in let q1 = i q x ip in (q1, x);;

let iibg q ip ir =
   let (x, y) = gendart2 ()
   in let q1 = l (i (i q x ip) y ir) 0 x y in (q1, x, y);;

let ieg q x y =
   let ip = aig q 0 x and ir = aig q 0 y
   in let (q1, x', y') = iibg q ip ir
      in let q2 = l (l q1 1 x x') 1 y y'
         in (q2, x', y');;

let msvg q x y = ... ;;

let ieg q x y c = ... ;;

let ecut q z p = ... ;;

let ive q z p = ... ;;

let deg q z = de q z;;

let dart_n = ref 0;;

let gendart () = dart_n := !dart_n + 1; !dart_n;;

let gendart2 () = (gendart (), gendart ());;
```

Most of these operations have to insert and sew unknown darts. Then, function `gendart()` returns a new dart, thanks to the management of a counter referred by `dart_n` with initial value `0`. Operation `gendart2()` do the same to generate a pair of new darts (Table 7a). Once again, Table 7b indicates some preconditions to avoid 0-embedding inconsistencies, loops and null edges. Predicate `condang q x y` is `true` iff the angle ordering is preserved around the vertices of `x` and `y` after inserting the new edge.

Table 7b: Preconditions

```
(*
  prec iibg q ip ir = ae ip • ae ir
  prec msvg q x y = prec msv q x y && a q 0 x /= y
                    && aig q 0 x = aig q 0 y
  prec ieg q x y = prec ie q x y && ag q 0 x /= ag q 0 y
                    && condang q x y
*)
```

11 Duality

Dual maps are useful to set and solve by transformation some problems about subdivisions. For instance, it is well known that the dual of a Voronoï plane diagram is a Delaunay plane diagram, and conversely. In the (projective) plane, both are subdivisions, i.e. embedded planar maps. The duality can be considered first at the topological level, secondly at the geometric level.

Definition 7 Let $M = (D, \alpha_0, \alpha_1)$ be an oriented combinatorial map. Then $M\sim\ = (D, \alpha_0, \alpha_1\alpha_0)$ is also an oriented combinatorial map, which is called the *dual* of the *primal* M.

Since $(\alpha_1\alpha_0)\alpha_0 = \alpha_1(\alpha_0\alpha_0) = \alpha_1$, we easily obtain the following result.

Proposition 6 The dual map $(M\sim)\sim$ of a dual map $M\sim$ is the primal map M, i.e. $(M\sim)\sim = M$. The vertices, edges and backward faces of $M\sim$ are the backward faces, edges and vertices of M, respectively. Finally, $M\sim$ and M have the same connectivity, Euler characteristic and genus.

Example Figure 9 shows in the dual map $M\sim\ = (D, \alpha_0\sim, \alpha_1\sim)$ of map M in Figure 3, with M = [1..18], $\alpha_0\sim\ = \alpha_0$ and $\alpha_1\sim$ = (1 4)(5 3 12 10 7 11 2 6 9 8)(13)(14 15 17 18 16). $M\sim$ has 8 direct faces, (2 4 5), (3 1 11), (9 12 7), (8 10 6), (13), (14), (15 16 18) and (17), and 8 backfaces, (2 4 12), (3 1 6), (5 9 7), (8 11 10), (13), (14 17 16), (15) and (18), which are the vertex of M. Thus the number of darts, edges, vertices, direct faces, components, and the Euler characteristic and genus of $M\sim$ are: $nd\sim\ = 18$, $ne\sim\ = 10$, $nv\sim\ = 4$, $nf\sim\ = 8$, $nc\sim\ = 3$, $ec\sim\ = 4$, and $g\sim\ = 1$.

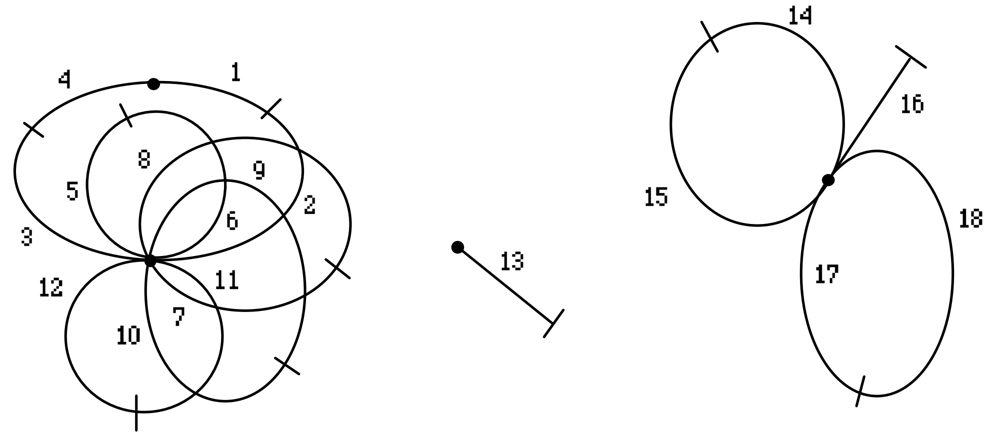


Figure 9: Dual $M\sim$ of map M in Figure 3 (with the usual drawing conventions)

In Table 8a, `dual q` returns the topological dual of embedded map `q`. It is specified inductively with respect to the usual `V`, `I` and `L` constructors.

Table 8a: Topological dual map

```
let rec dual q =
   match q with
      V -> V
   |  I(m,x,i) -> I(dual m,x,i)
   |  L(m,0,x,y) ->
         let m' = L(dual m,0,x,y)
         in if fk m 1 y then m'
            else let y1 = a m 1 y in L(m',1,x,y1)
   |  L(m,k,x,y) ->
```

```
    let m' = dual m
    in if fk_1 m 0 x then m'
       else let x_0 = a_1 m 0 x in L(m',1,x_0,y);;
```

For the geometric dual map, we consider the particular framework of the Voronoï and Delaunay diagrams for a set of points in the Euclidean plane (Figure 10). In fact, [GS85] underlines that ”the hardest part of constructing a Voronoï diagram is the determination of its topological structure, that is, the incidence relation between vertices, edges and faces”.

Most of the building processes begin with the Delaunay diagram and deduce the Voronoï diagram by duality. For Delaunay step, the reader can refer to the classical paper [GS85], where the problem is solved with the help of topological operators rigorously described by an “edge algebra“, a bit more general than our combinatorial maps. In Section 12, we transpose in our framework the incremental algorithm for Delaunay triangulations proposed in this paper. Thus, we suppose in this section that a Delaunay diagram is already built as a linearly embedded combinatorial planar map `q`, and we specify the transformation `edual q` into the Voronoi dual diagram, which is also a planar map linearly embedded in the *projective plane*, i.e. the Euclidean plane completed by *points at infinity* in all the directions. The main useful property is the following.

Property 7 Each vertex of a Voronoï diagram is 0-embedded at the center of the circumcircle of the dual direct face in the corresponding Delaunay diagram.

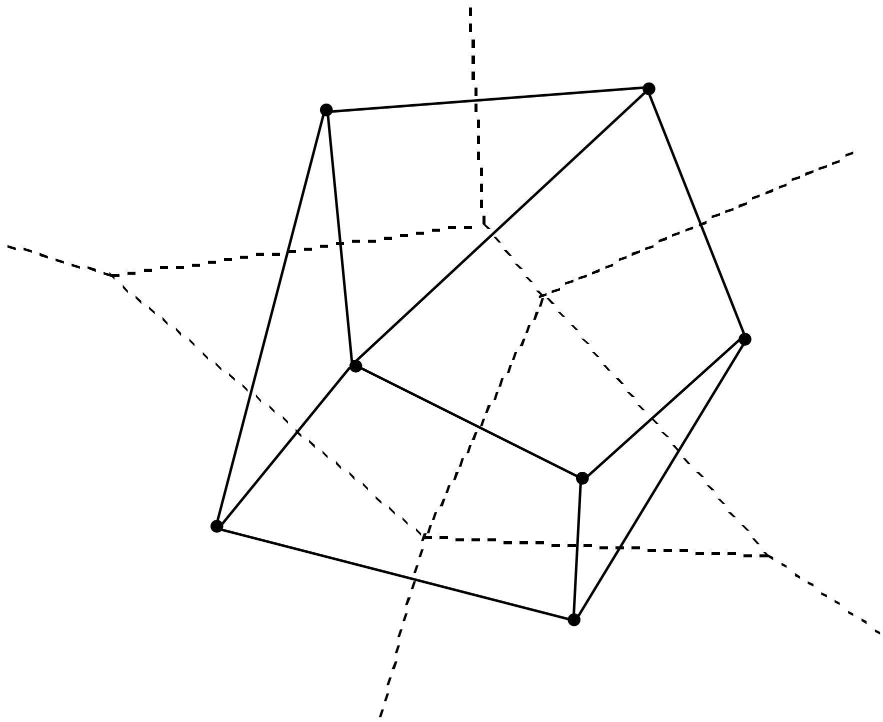


Figure 10: Delaunay diagram (in full lines) and its Voronoï dual (in dash lines)

When the vertices are collinearly embedded, the circumcircle is degenerated into a line and the center of the circle is a point at infinity. Two methods can be used to obtain the Voronoï diagram. First, during the vertex merging or splitting of the operation which returns the dual, the new vertex 0-embeddings must be determined by computing centres of circumcircles of direct faces in the primal map. Secondly, when the topological map is entirely computed, the embedding is realized in one move. Table 8b precises this last process in the specification of `edual q`, where auxiliary `ccc q z` returns an index for the center of the circumcircle of the direct face of `q` containing `z`.

Table 8b: Voronoï diagram

```
let rec edual1 q qr =
   match q with
      V -> dual qr
   |  I(m,x,i) -> I((edual1 m qr),x,(ccc qr x))
   |  L(m,k,x,y) -> edual1 m qr

let edual q = edual1 q q;;
```

Operation `edual q` uses auxiliary `edual1 q qr,` where `q` is dismantled whereas `qr` is a reference map to compute the circle centres. When `q` is empty, the topological dual `dual qr` is returned. Note that the darts which are 0-embedded in the Voronoï diagram are exactly those which were 0-embedded in the Delaunay diagram.

However, this specification is partially wrong, because the Voronoï vertex which is the dual of the external, or unbounded, Delaunay backward face must be rejected to the infinity in the directions that are perpendicular to the bording edges, so that the infinite Voronoï edges can be correctly embedded. On the screen or on a paper sheet, the edges are in fact necessarily bounded. This particular question may be treated afterwards, depending on the exact drawing conditions, preferably in the projective plane with homogeneous coordinates. Finally, the reader may verify that the representation of the dart permutation around the non infinite vertices in the Voronoï diagram is the inverse of the one in the Delaunay diagram. In other words, the permutation turns clockwise in the Voronoï diagram if the usual counter-clockwise convention is used in the Delaunay diagram.

12 Delaunay triangulation

To illustrate the power of the tools we have recommended, namely combinatorial embedded maps and functional specification, we develop a significant example of map construction, for a Delaunay triangulation. Basically, we paraphrase the incremental method presented in [GS85], which works by insertion of sites, one after the other, under the following hypotheses.

The sites are supposed to be strictly inside some larger polygon, whose vertices are among the given sites. The Delaunay diagram of the sites inserted so far is thus a triangulation of the interior of this bounding polygon. The problem solved here consists in adding a new site. Collinearity of three sites or cocircularity of four sites are allowed and considered, but redundant sites are removed.

As technical particularities in our framework, the map corresponding to the triangulation is completely 0-embedded with a linear implicit 1-embedding. Solving the problems needs some auxiliary operations defined in Table 9a. Thus, `det2`, `det3` and `det4` return the determinant of a matrix 2 * 2, 3 * 3 and 4 * 4, respectively. The two fundamental predicates in the Euclidean plane are `circ pa pb pc`, which is `true` iff the points `pa`, `pb` and `pc` are traversed is that order counter-clockwise in the plane, and incircle pa pb pc pd, which is true iff point pd is strictly inside triangle pa pb pc circumcircle when circ pa pb pc, otherwise strictly outside. An extensive axiomatic study of their properties lies in [Knu93], as well as an impressive amount of theoretical, practical and methodological results around convex hull and Delaunay-Voronoï problems and algorithms.

Predicate rightof q pa z is true iff point pa is strictly at the right of the oriented segment which embeds z in map q, and function mod2 p returns the square distance from point p to the origin. Finally onedge q p z is true iff point p is strictly inside the segment which 1-embeds z in map q.

Table 9a: Delaunay auxiliaries

```
let det2 a11 a12 a21 a22 = a11 *. a22 -. a12 *. a21;;

let det3 a11 a12 a13 a21 a22 a23 a31 a32 a33 = ...;;

let det4 a11 a12 a13 a14 a21 a22 a23 a24 a31 a32 a33 a34
          a41 a42 a43 a44 = ...;;

let circ pa pb pc =
   det3 (abs pa) (ord pa) 1. (abs pb) (ord pb) 1.
        (abs pc) (ord pc) 1. > 0.;;

let rightof q pa z = let pb = ag q 0 (a q 0 z) and pc = ag q 0 z
   in circ pa pb pc;;

let mod2 p = let x = abs p and y = ord p in x *. x +. y *. y;;

let incircle pa pb pc pd =
      det4 (abs pa) (ord pa) (mod2 pa) 1.
                    (abs pb) (ord pb) (mod2 pb) 1.
                    (abs pc) (ord pc) (mod2 pc) 1.
                    (abs pd) (ord pd) (mod2 pd) 1. > 0.;;

let onedge q p z =
   let pa = ag q 0 z and pb = ag q 0 (a q 0 z)
            and x = abs p and y = ord p
   in let xa = abs pa and ya = ord pa and xb = abs pb and yb = ord pb
      in det2 (x -. xa) (x -. xb) (y -. ya) (y -. yb) = 0.
         && (let l = if xb = xa then (y -. ya) /. (yb -. ya)
                       else  (x -. xa) /. (xb -. xa)
             in 0. < l && l < 1.);;
```

In Table 9b delaunay q ip solves the problem of inserting a new site referred by point index ip in a triangulation represented by map q. We only follow the algorithm to show our tools at work, without recall of geometric justifications, because they are clearly exposed in [GS85].

Example Figure 11 illustrates the construction steps, the built elements being numbered in parentheses. In this drawing, the initial polygon (number 0) is the triangle abc.

The process begins by locating the point corresponding to ip in triangulation q, that is determining a dart of the oriented triangular face the point is included in. That is the role of locate q p (chdart q), which needs starting from some initial dart, given here by chdart q.

Example In Figure 11, the first added site, numbered (1), is `p1` which is located strictly inside the initial triangle. When the new site is redundant with a previous one, nothing is done, and when it falls on an edge, the latter is deleted, thus creating a quadrangular face.

Then, by `istar q1 ap e`, a "star" of edges around a new vertex embedded on `ip` is built in the current map `q1`, joining this vertex to the vertices of the triangular or quadrangular face. Starting from a dart called `e`, like in [GS85], the process consists in creating the first edge of the star by `iibg`, then traversing the whole face by a recursive call of `istar1`. The latter creates and sews one by one the other star edges by `ieg`, until coming back to `e`.

Example In Figure 11, a star is created around `p1`, with edges `ap1`, `bp1`, and `cp1` numbered by (1). The second site is `p2`, which falls on edge `cp1`. Thus, the latter is deleted and a quadrangle `ap1bc` appears. The corresponding star contains edges `ap2`, `bp2`, `p1p2` and `cp2` numbered by (2).

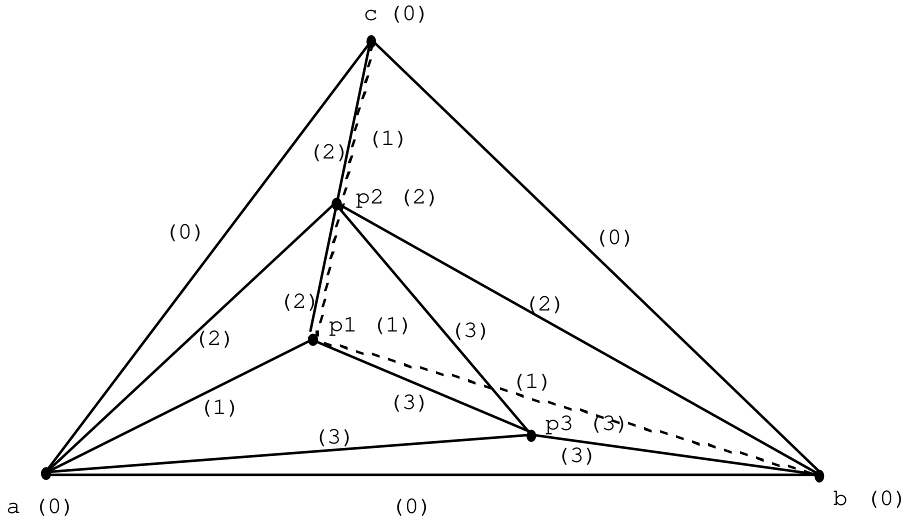


Figure 11: Delaunay triangulation
(edges created then removed are in dash lines)

After creating the star, each edge opposed to `ip` in a triangle, which is called *suspect* in [GS85], is examined to determine if it has to be "swapped" with the diagonal of the quadrangle whose the first edge is the other diagonal. That is necessary if `ip` is inside the circumcircle of the triangle formed by the other three vertices of the quadrangle, what is tested by `incircle`. The complete swapping process is achieved by a `swapping` call, which is parameterized by a first star edge, called `base` like in [GS85], adjacent to `e`. If necessary, `swapping` does the first swap and the convenient sewings by `swap q e`, then releases the traversal by calling the recursive function `swapping1` on the successor of `base` in the star. This successor can be the diagonal which has just been added. The call of `swapping1` does recursively the other edge exchanges and sewings until the initial `base` is found anew. Note that when `ip` is not opposed to a triangle beyond `e`, because the latter borders the convex hull, the process always works since `incircle` always returns `false`.

Example In Figure 11, no swap is necessary for the first two sites `p1` and `p2`. For the third one, `p3`, which is inside triangle `abp1`, the star is `ap3`, `p1p3` and `bp3`. However, `p3` lies inside the triangle `p1bp2` circumcircle. Thus the diagonals `p1b` and `p3p2` of quadrangle `p1p3bp2`, must be swapped. Then `p1b` is deleted and the new edge `p3p2` is added and sewn in the star.

Table 9b: Delaunay triangulation

```
let rec istar1 q ip el e yp =
   if e = el then q
   else let ir = aig q 0 e
        in let (q1, x, y) = iibg q ir ip
           in let q2 = l (l q1 1 e x) 1 yp y
              in istar1 q2 ip el (phi q2 e) y;;

let istar q ip e =
   let ir = aig q 0 e
   in let (q1, x, y) = iibg q ir ip
      in let q2 = l q1 1 e x
         in istar1 q2 ip x (phi q2 e) y;;

let swap q e =
   let q1 = deg q e and x = (a q 0 (a q 1 e))
       and y = (phi q (a_1 q 1 e))
   in let (q2, x', y') = ieg q1 x y in q2;;

let rec swapping1 q p basel base =
   if base = basel then q
   else let e = a_1 q 1 base
        in let pe = ag q 0 e and pe0 = ag q 0 (a q 0 e)
                    and pe2 = ag q 0 (a q 0 (a_1 q 1 e))
           in if incircle pe pe2 pe0 p
              then swapping1 (swap q e) p basel base
              else swapping1 q p basel (phi q (a_1 q 1 base));;

let swapping q p base =
   let e = a_1 q 1 base
   in let pe = ag q 0 e and pe0 = ag q 0 (a q 0 e)
          and pe2 = ag q 0 (a q 0 (a_1 q 1 e))
      in if incircle pe pe2 pe0 p
         then swapping (swap q e) p base
         else swapping1 q p base (phi q (a_1 q 1 base));;

let delaunay q ip =
   let p = (ae ip)
   in let z = locate q p (chdart q)
      in let pz = ag q 0 z and pz0 = ag q 0 (a q 0 z)
         in if p = pz || p = pz0 then q
            else let (e, q1) = if onedge q p z then (phi q z, deg q z)
                                else (z, q)
                 in let q2 = istar q1 ap e in swapping q2 p (a q2 1 e);;
```

Table 9c specifies the point location by `locate q p z` of point `p` in triangulation `q` from initial dart `z` known to be in the convex hull of `q`. The result is a dart of the face containing `p`, the latter being possibly in the closed segment 1-embedding the dart.

Table 9c: Point location

```
let rec locate q p z =
 let pz = ag q 0 z and pz0 = ag q 0 (a q 0 z)
 in if p = pz || p = pz0 then z
    else if rightof q p z then locate q p (a q 0 z)
         else let z1 = (a q 1 z)
              in if not (rightof q p z1) then locate q p z1
                 else let z2 = a q 1 (a q 0 z1)
                      in if not (rightof q p z2) then locate q p z2
                         else z;;
```

The process mainly consists in examining by `rightof` the place of `p` with respect to the oriented edges of the current face. If `p` is in the face of `z` or on its edge, then the result is `z`, otherwise the process is recursively carried on with an adjacent face, `p` is at the left or on an incident edge of which. This algorithm was first described in [GS77].

13 Other high-level operations

In this section, we introduce two other examples of higher level operations solving concrete topological and geometric problems in embedded maps. To be short, we only give informal statements. More complete and formal developments can be found in [DP00], which shows how operations are inductively defined on our map templates.

First, we take a map `q` linearly embedded in the plane with vertex and edge overlaps. Then, through edge-cutting and vertices-merging, `refine q` solves the problem of restoring a good embedding, i.e. without any overlap, and returns a planar map planarly embedded. This operation of *map refinement* is completely explained in [CD97, CD99, DP00]. It can be illustrated by the superposition of the two plane subdivisions of Figure 12 into a unique map with overlaps in Figure 13(a).

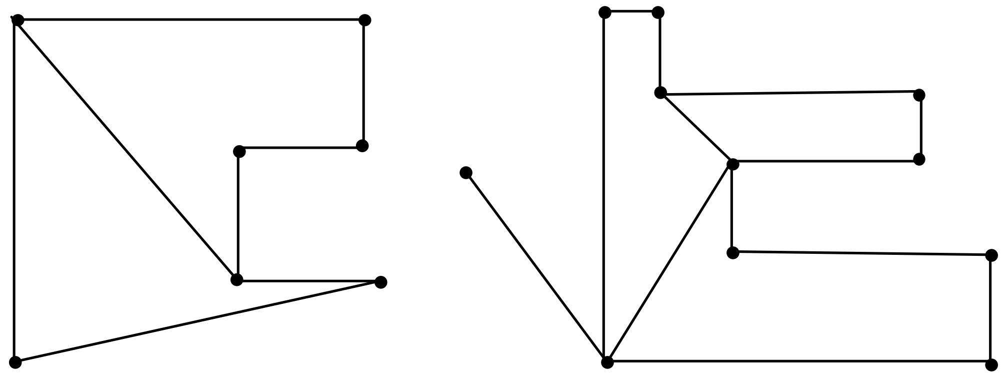
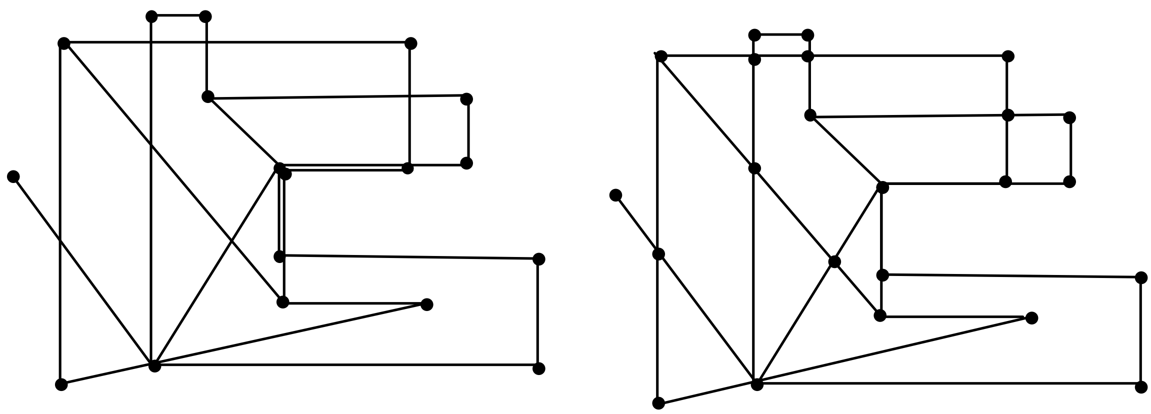


Figure 12: Two plane subdivisions

The result of the refinement of this map in a planarly embedded map representing a “good“ subdivision with no overlaps is then presented in Figure 13(b).

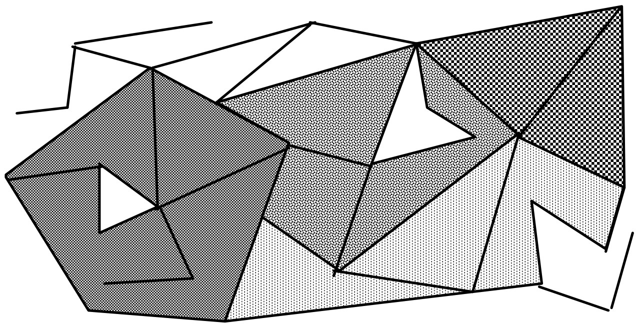


(a) Superposition of the subdivisions (b) Final result of the refinement

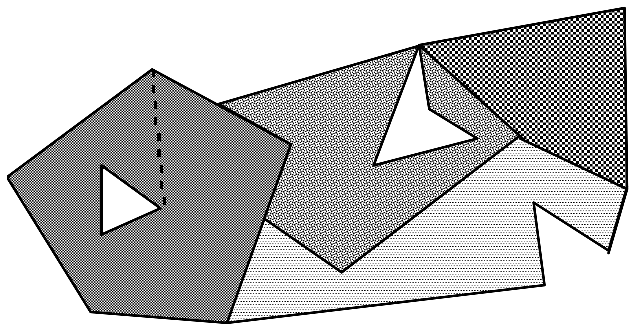
Figure 13: Refinement of the subdivisions

Secondly, when all the direct faces of a planar map are labelled, `segmente q` merges the adjacent direct faces with the same label, by deletion of their common walls. When the labels are colors or grey levels, this operation can be seen as a very simple case of image *segmentation*. Thus, Figure 14(a) presents a subdivision with coloured direct faces and Figure 14(b) the result of the segmentation of this subdivision.

The result may be disconnecting or not, depending on the preservation of some isthmus during the segmentation. Such an isthmus is drawn in Figure 14(b) as a dashed segment, which avoids a disconnected direct face included in another one and is, or not, a part of the result. Details about this operation and variants are also in [DP00]. Note that it is a sheer operation of *computational topology*, in accordance with [Män82, DEG98].



(a) Original subdivision (textures correspond to face labels)



(b) Final subdivision after segmentation

Figure 14: Segmentation of a map

14 A proposal of implementation

In an efficient classical implementation in main memory, for instance directly with C language, an embedded map is a pointer to a map structure and, instead of a natural as in the prototype, a dart is a pointer to a dart structure. The map structure mainly contains the current map characteristics and a pointer to the first element of a one-way linked list of all the dart structures of its darts. Note that we do not divide the list according to the connected components of the map, which is harmful as shown in [BD94]. Moreover, for a connected completely embedded map, memorizing and updating the address of an external dart, i.e. a dart incident to the unbounded face, is useful in numerous problems, like the construction of a Delaunay diagram.

Each dart structure mainly contains a pointer to the successor in the list, and the dart images in the map by the permutations α_0, α_1 and α_1^{-1}. It also possesses pointers to the point, arc, attributes which embed the corresponding dart, and some markers, Boolean or darts, which help to memorize tracks during traversals of the map. From this point of view, operation `gendart` simply becomes an allocation of a new dart structure which returns a pointer to this structure, and `gendart2` two allocations. This way, the void dart is represented by the `null` address and the unicity of the darts is automatically guaranteed.

It is pleasant to manipulate dart attributes also through pointers, which reify our indexes. The index generation and environment management mechanisms we have described in our functional language are easily translated into efficient data structure allocation with pointers and table handling. Moreover, to quickly recover the map entities from attributes, each attribute object may contain a pointer towards a representative dart of the corresponding entity. In fact, the mode of attribute sharing is of prime importance.

With such an implementation, time complexity can be considered for the operations, with respect to the number n of darts. Recall that the numbers of vertices, edges and faces are always bounded by n. Thus, constructors, `V`, `I` and `L`, are in O(1), as well as most of the basic operations, `empty`, `a`, `fk`, `phi`, `r`, `i`, `l`, `l'`, `iil`, `iib`, etc. Others, like `e`, `d` or `nd`, are in O(n) in the worse case, as well as traversals, equivalences and computing from scratch the map characteristics. However, the current characteristics can be stored in the map structure and incrementally updated. In short, computing in maps is quite similar as computing is graphs, but rather in favourable cases.

Concretely, a lot of operations to manage maps in an usual way have to be added. Thus, it is necessary to duplicate maps in a main memory, with or without dart changing, or to destroy them physically. It is essential to store them on auxiliary memories or to load them from files. If some of these functions ask questions in a framework of formal specification [BHK88], all of them can be easily written in a functional language.

With an object-oriented language, the vision can be more modular. In this way, several hierarchical classes are to be defined. At the top of the hierarchy, the class `qmap` contains all the features of our extended map at the first level, with the three constructors `V`, `I`, `L`, constrained by their preconditions. Just below, the class `map`, subclass of `qmap`, exactly describes the oriented combinatorial maps, with the constructors `v`, `i`, `l`, which are built from the previous ones and have their own preconditions in order to preserve the map integrity. In this class, `V`, `I`, `L` are hidden to the users. Finally, classes `eqmap`, subclass of `qmap`, and `emap`, subclass of `map` and `eqmap`, contain all the features and operations of the embedded quasi-maps and maps. In fact, a lot of other classes can be defined depending on the features exactly required.

15 Extensions and *d*-dimensional case

The above notions extend to the proposals of d-dimensional hypermaps, generalized maps and maps, used in the literature and in modellers to describe d-dimensional space subdivisions. *Hypermaps* were introduced in [Cor75] and used in [Duf91a] to generalize and formalize the handling of d-dimensional subdivisions on computers:

Definition 8 Let d be a natural number. A *d-dimensional hypermap*, or *d-h-map*, is a d+2-tuple $H = (D, \alpha_0, \alpha_1, ..., \alpha_d)$, where D is a finite set of darts and $\alpha_0, \alpha_1, ..., \alpha_d$ are permutations in D.

A notion of *d-dimensional map* [Lie88, Lie89, Duf91a], able to describe orientable and closed subdivisions of d-dimensional spaces, derive.

Definition 9 Let d be a non zero natural number. A *d-dimensional map*, or *d-map*, is a (d-1)-h-map $M = (D, \alpha_0, \alpha_1, ..., \alpha_{d-1})$, where $\alpha_0, \alpha_1, ..., \alpha_{d-2}$ are involutions and $\alpha_j\alpha_i$ is an involution for all $i, j \in [0, d\text{-}1]$ such that $i+2 \bullet j$.

The last conditions ensure the closure and the consistency of the sewing at any dimension. Thus, our combinatorial oriented maps are just 2-maps. Variants of 3-maps have been proposed and studied under the name *pavings* [AK89, Spe90, 91]. Moreover, the notion of *d-dimensional generalized map* was introduced in [Lie88, Lie89] to describe subdivisions of d-dimensional spaces, orientable or non orientable, open or closed (Figures 15-17). It is particularly simple, efficient and attractive, because it involve only involutions:

Definition 10 Let d be a natural number. A *d-dimensional generalized map*, or *d-g-map*, is a d-h-map $G = (D, \alpha_0, \alpha_1, ..., \alpha_d)$, where $\alpha_0, \alpha_1, ..., \alpha_{d-1}$ are involutions without fixpoint, α_d is an involution, and $\alpha_j\alpha_i$ is an involution for all $i, j \in [0, d]$ such that $i+2 \bullet j$. A d-g-map is *open* if it has fixpoint relatively to α_d, otherwise it is *closed*.

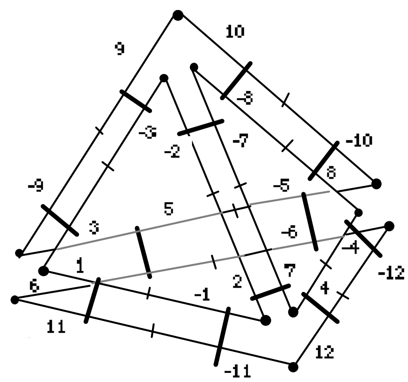


Figure 15: A connected closed 2-g-map that models a tetrahedron
(Small dashes correspond to α_0, medium thick bars to α_2, and points to α_1)

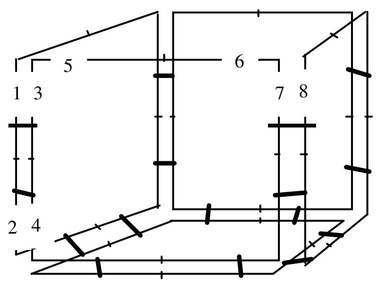


Figure 16: A connected open 2-g-map that models an open cube
(5 and 6 are two fixpoints relatively to α_2, thus on the border)

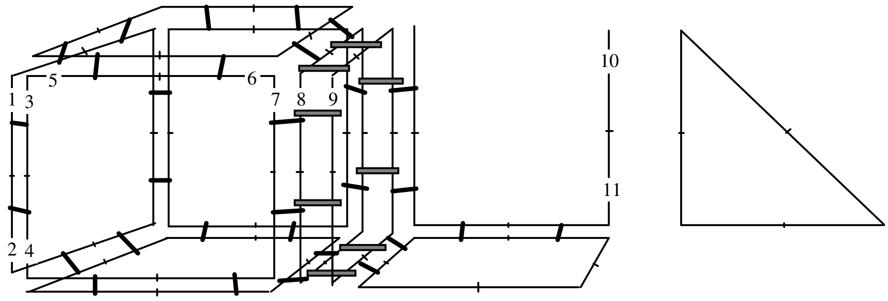


Figure 17: A non-connected 3-g-map *under construction* to model adjacent solids
(Very thick gray bars correspond to α_3)

Once more, the last conditions ensure the consistency of the described objects. But they can be alleviated during a construction phase, as illustrated in Figure 17. Like for maps, cells of dimensions $k \in [0, d]$ can be extracted from a d-g-map G:

Definition 11 The *cells* of dimension k of G, or *k-cells*, are the connected components of the (d-1)-g-map $G_k = (D, \alpha_0, \ldots, \alpha_{k-1}, \alpha_{k+1}, \ldots, \alpha_d)$. The *vertices*, *edges*, *faces* and *volumes* of G are the k-cells for k = 0, 1, 2 and 3, respectively.

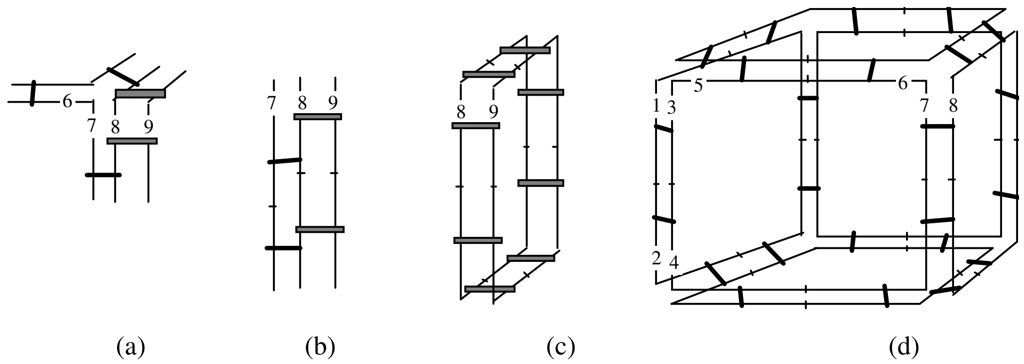


Figure 18: Cells of the 3-g-map in Figure 17 : vertex (a) , edge (b), face (c), volume (d)

Anew, cells can be defined for non completely consistent g-maps, as in Figure 18. One can see g-maps components from a more elementary point of view (Figure 19):

Definition 12 The *simple cells* of dimension k of G, or *k-s-cells*, are the connected components of the (k-1)-g-map $G'_k = (D, \alpha_0, \ldots, \alpha_{k-1})$. The *simple vertices*, *simple edges*, *simple faces* and *simple volumes* of G are the k-s-cells for k = 0, 1, 2 and 3, respectively.

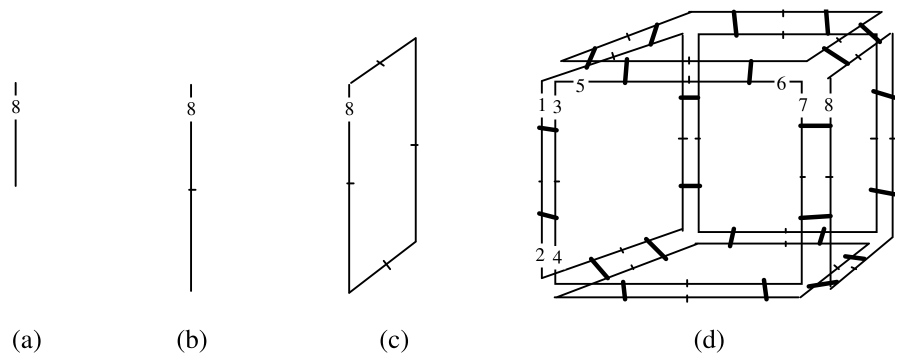


Figure 19: Simple cells of the 3-g-map in Figure 16 : simple vertex (a) , edge (b), face (c), volume (d)

In fact, the conditions of Definition 10 express that the sewings of a g-map must involve closed simple cells at any dimension. A notion of border is also obtained throughout graph considerations [Lie88, 89] (Figure 16):

Definition 13 The *borders* of G are the connected components of the closed 1-g-map $G' = (D', \alpha'_0, \alpha'_1)$, where D' is the subset of D formed by the fixpoints relatively to α_d, α'_0 is the restriction of α_0 to D', and α'_1 is defined for any x in D' by: $\alpha'_1(x)$ = the only dart distinct from y in the connected component of x in the 1-g-map $(D, \alpha_{d-1}, \alpha_d)$.

It is clear that the map axiomatics is directly extensible to d dimensions, in the spirit of [Duf91a]. Operations to create d-maps, d-g-maps, k-cells or k-s-cells, to sew/unsew darts, k-cells or k-s-cells for any k, to count darts or k-cells, to traverse k-cells or k-s-cells, are easily specified and prototyped in the way we have followed for oriented maps. Embeddings can also be defined for all the *k*-cells, for instance in the 3D space, points for the vertices, Jordan arcs for the edges, disc homeomorphic patches for faces, and ball homeomorphic manifolds for volumes. An original 3D cellular interactive modeller named *Topofil* was completely specified and developed in real-size from these considerations [BDF+93, BD94, Ber98, BD98].

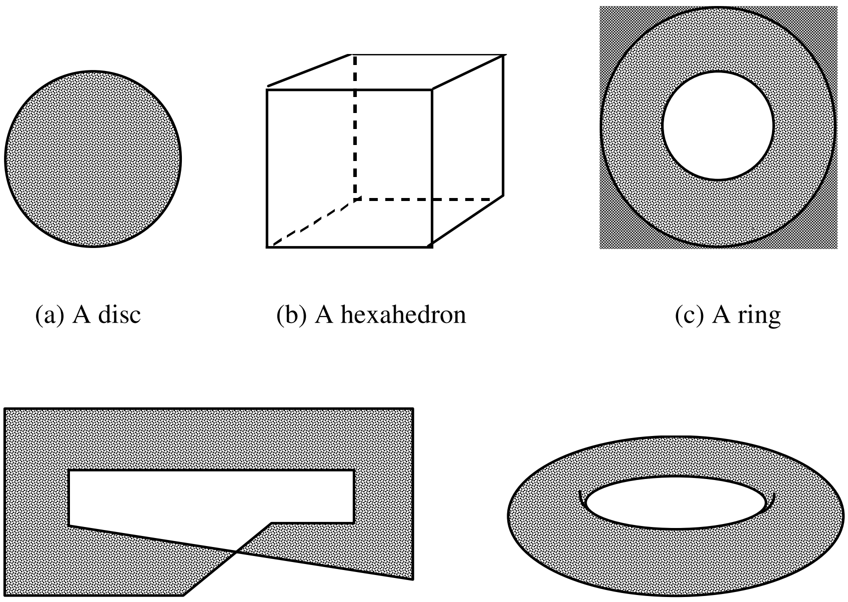
In fact, the above map notions has been significantly enlarged, as combinatorial oriented maps into quasi-maps, for reasons of formal specification and computation discussed in [Duf89, Duf91a, BD94, BD98]. In the future, we expect all these sorts of data, with their invariants and operations, to be hierarchically organized in a *semantic network* with *ordered types*, *inheritance* and *polymorphism*, also including embedding features.

Finally, starting from maps or generalized maps, a lot of propositions can be made to describe *non manifold* topological spaces of any dimension. For instance, the *chains* of generalized maps [EL92] have a modelling power greater than the well known *radial edge* [Wei85] to describe non manifold objects of dimension 3. However, the next section focuses of the topological modelling of varieties of dimension 2, i.e. *surfaces*, which is always a topical subject in geometric modelling.

16 Topology of surfaces

The definition of a surface is often given as follows in mathematics textbooks:

Definition 14 A *surface* is a Hausdorff space, each point of which having a neighborhood homeomorphic to the plane or to the half-plane y • 0.



(a) A disc (b) A hexahedron (c) A ring

(d) A Möbius band (e) A torus

Figure 20: Surfaces

Examples of different kinds are given in Figure 20. Topologies of (connected) surfaces are characterized, or *classified*, by three naturals: b, their number of borders, q, their coefficient of *orientability* (0 for orientable, 1 or 2 for non orientable) and g, their *genus*, that is their number of holes (See for instance [Gri81] for a comprehensive presentation).

Example For the surfaces in Figure 20, the triples (b, q, g) are respectively: (a): (1, 0, 0), (b): (0, 0, 0), (c): (2, 0, 0), (d): (1, 1, 0), (e): (0, 0, 1). It is (0, 2, 0) for a Klein Bottle.

Instead of such a descriptive approach, it is often better in computer science to work constructively. For instance, it is convenient for interactive manipulations on a computer to consider that, topologically speaking, a surface is the result of successive collages of *lamellas*, i.e. patches homeomorphic to discs, along their borders [Gri81]. That naturally forms a subdivision of the surface which can be described by a 2-g-map. More precisely, we have:

Proposition 8 There is a one-to-one correspondence between topologies of surface subdivisions and classes of isomorphic 2-g-maps.

It follows that the characteristics of surface topologies can be combinatorially obtained:

Proposition 9 The topological characteristics (b, q, g) of a surface S are computed from a corresponding 2-g-maps G in the following way:

b is the number of borders of G
$\chi = nv + ne + nf - nd$
$g = 1 - (b + q + \chi)/2$
$q =$ ***if*** G is orientable ***then*** 0
else if $b + \chi$ is *odd* ***then*** 1 ***else*** 2

where nv, ne, nf and nd are the numbers of vertices, edges, faces and darts of G, respectively, and χ is exactly the map *Euler characteristic* of Section 7.

Note that, once again, g can be proven to be a natural number. Most of the above numbers can be obtained by simple traversals of the 2-g-map graph structure of G. The only difficulty is testing the orientability of G. It is obtained from the following considerations.

Definition 15 The *oriented volumes* of G are the connected components of the combinatorial oriented map $V = (D, \alpha_2\alpha_0, \alpha_2\alpha_1)$.

Example The oriented volumes of the tetrahedron 2-g-map in Figure 13 are in Figure 19.

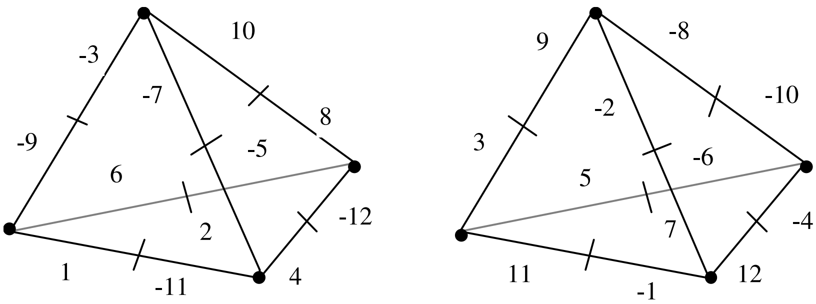


Figure 21: Oriented volumes of the tetrahedron 2-g-map

The map of the oriented volumes is computed by an efficient graph-like traversal of the 2-g-map. Finally, the orientability is simply defined as follows:

Definition 16 Let G be a connected closed 2-g-map. It is said *orientable* if it has two distinct oriented volumes.

When the 2-g-map is open, it can be closed by putting a *cap* on each border, which is a trivial operation. If it is not connected, the previous considerations apply to each of its connected components. Finally, note that all the Euler operators [Män82, 88] can be described in the framework of the 2-g-maps.

17 Conclusion

When dealing with the combinatorial topology of d-dimensional geometric objects, the notion of map and all its derivatives can be used to build any model, thanks to involutions and permutations which are more or less constrained [Lie91, Duf91a, EL92, BD94]. The original concept of oriented combinatorial map allows to model and manipulate the topology of two-dimensional manifolds, i.e. surfaces, of any genus, but always closed and orientable. But to deal with real geometric problems, maps must be embedded in a n-dimensional space, i.e. associated with geometric entities. Thus recall that, when embedded in the plane, a combinatorial map can present self-intersections.

The map formalism has been presented by means of *functional specification* techniques. They allow to formalize, abstract, build, prove and prototype. In this framework, preconditions for hierarchized operations can be defined, which leads to sound geometric objects satisfying integrity constraints. Topological constraints have been emphasized, but complex embedding constraints can also be handled [Män88]. Such a specification is a sound basis of discussion between software designers, developpers and final users.

Moreover, the clear distinction between topology and embedding on the one hand, and the high level of abstraction of functional writings on the other hand, should help to study fundamental approximation or complexity problems, where the role of the different basic numerical operations precisely appears, before doing a real implementation [BP97].

Among all the languages for functional specifications, *Ocaml* appears as well adapted: it allows types, classes, modularity, genericity, inheritance, polymorphism, and, of course, execution. Nevertheless, the coexistence of types inductively defined and hierarchized classes is impossible for the present being. Owing to their clarity and proof facilities [DP00], we have exclusively opted for the way of inductive types.

Efficient *implementations* from algebraic or functional specifications is a real difficulty [BHK88] that we have only pragmatically approached through large-size experiments [BDF+93, BD94, BD98]. This transition must be deeply studied, in connection with the wide development of some programming technologies, like object-oriented ones.

However, given the difficulty and the wide range of computational geometric problems, it seems difficult to design and build today an universal topological and geometric high-level *toolbox*. In the way open by [Knu92], the most profitable is probably to deeply study and implement basic types, or classes, easy to extend and specialize, and atomic operations easy to compose, depending on the applications. To help exploring this long way, maps and functional languages seem good candidates.

Acknowledgements

The ideas of the "Axiomatic Methods for Geometric Modelling" Group of Strasbourg have greatly inspired this work. Thus, I am particularly grateful to Y. Bertrand, D. Cazier, M. Decker, C. Dehlinger, C. Essert, L. Fuchs, S. Luther, P. Mathis, F. Puitg and P. Schreck, who have studied with me formal specifications and construction techniques in geometry and computer graphics, from theoretical and practical points of view.

References

[AB97] Arquès, D., Béraud, J.F.: Enumération des cartes pointées sur la bouteille de Klein. *RAIRO Inf. Théor. et Applic.*, Vol 31, No 4, 1997, 385-409.

[AK89] Arquès, D., Koch, P.: Modélisation de solides par pavages. Proc. *PIXIM Conf.*, Hermès, 1989, 47-61.

[Arq85] Arquès, D.: Une relation nouvelle sur les cartes planaires pointées. *Journal of Comb. Theory - Serie B*, Vol 39, No 1, 1985, 27-42.

[Bau70] Baumgart, B.G.: A polyhedron representation for computer vision. Proc. *Nat. Comp. Conf.*, AFIPS Press, 1970, 589-596.

[BD94] Bertrand, Y., Dufourd, J.F.: Algebraic Specification of a 3D Modeller based on Hypermaps. *Computer Vision-GMIP*, Vol 56, No 1, 1994, 29-60.

[BD98] Bertrand, Y., Dufourd, J.F.: Du Modèle Géométrique à la Programmation par les Spécifications Algébriques. *TSI*, Vol 17, No 4, Hermès, 1998, 485-516.

[BDF+93] Bertrand, Y., Dufourd, J.F., Françon, J., Lienhardt, P.: Algebraic Specification and Development in Geometrical Modeling. Proc. *EATCS TAPSOFT*, Orsay, LNCS 668, Springer-Verlag, 1993, 75-89.

[Ber98] Bertrand, Y.: Topofil: un Modeleur d'Objets Géométriques Tridimensionnels. *TSI*, Vol 17, No 4, Hermès, 1998, 443-484.

[BHK88] Bergstra, J.A., Herig, J., Klint, P.: *Algebraic Specification*. ACM Press, Addison-Wesley, 1988.

[Bri89] Brisson, E.: Representing Geometric Structures in d Dimensions: Topology and Order. Proc. *5th ACM Symp. on Computational Geometry*, Saarbrücken, Germany, 1989, 218-227.

[BS82] Bryant, R.P., Singerman, D.: Foundations of the theory of maps on surfaces with boundary. *Quart. J. Math. Oxford* (2), 36, 1982, 17-41.

[CD97] Cazier, D., Dufourd, J.F.: Term Rewrite Systems to Derive set Boolean Operations on 2D Objects. Proc. *4th Formal Methods Europe*, Graz, Austria, LNCS 1313, Springer-Verlag, 1997, 605-623.

[CD99] Cazier, D., Dufourd, J.F.: A formal specification of geometric refinements. *The Visual Computer Journal*, Vol 15, 1999, 279-301.

[Cor75] Cori, R.: Un Code pour les Graphes Planaires et ses Applications. *Astérisque*, Vol 27, Soc. Math. de France, Paris, 1975.

[CR72] Cori, R., Richard J.: Enumeration des graphes planaires à l'aide des séries formelles en variables non commutatives. *Discrete Mathematics*, Vol 2, 1972, 115-162.

[DEG98] Dey, T.K., Edelsbrunner, H., Guha, S.: Computational Topology. *Research report*, 1998.

[Duf89] Dufourd, J.F.: Algebraic Map-Based Topological Kernel for Polyhedron Modelers. Proc. *Eurographics*, Hamburg, FRG, Elsevier, 1989, 301-312.

[Duf91a] Dufourd, J.F.: Formal Specification of Subdivisions using Hypermaps. *Computer Aided Design*, Butterworth-Heinemann, Vol 23, No 2, 1991, 99-116.

[Duf91b] Dufourd, J.F.: An OBJ3 Functional Specification for the Boundary Representation. Proc. *1st ACM-Siggraph Symp. on Solid Modeling Found. and CAD/CAM Appl.*, Austin, Texas, ACM Press, 1991, 61-72.

[Duf97] Dufourd, J.F.: Algebras and formal Specifications in Geometric Modelling. *The Visual Computer Journal*, Vol 13, Springer-Verlag, 1997, 131-154.

[DP00] Dufourd, J.F., Puitg, F.: Functional specification and prototyping with combinatorial maps. *Computational Geometry – Theory and Applications*, Elsevier, Vol 16, 2000, 129-156.

[Edm60] Edmonds, J.: A Combinatorial Representation for Polyhedral Surfaces. *Notices of AMS*, Vol 7, 1960.

[EL92] Elter, H., Lienhardt, P.: Extension of the notion of map for the representation of the topology of cellular complexes. *4th Canadian Conf. on Computational Geometry*. St Johns, Newfoundland, 1992.

[EM85] Ehrig, H., Mahr, B.: *Fundamentals of Algebraic Specifications - Equations and Initial Semantics*. Springer-Verlag, 1985.

[FGK+98] Fabri, A., Giezeman, G.J., Kettner, L., Schirra, S., Schönherr, S.: On the design of CGAL, the Computational Geometry Algorithms Library. *Research report* 3407, INRIA, France, 1998.

[GS77] Green, P.J., Sibson, R.: Computing Dirchlet tesselation in the plane. *Computing J.*, Vol 21, No 2, 1977, 168-173.

[Gri81] Griffiths, H.B.: *Surfaces*, Cambridge University Press, 2nd edition, 1981.

[GS85] Guibas, L., Stolfi, J.: Primitives for the manipulationof general subdivisions and the computation of Voronoï diagram. *ACM Trans. on Graphics*, Vol 4, 1985, 74-123.

[Jac70] Jacques, A.: Constellations et Graphes Topologiques. *In Combinatorial Theory and Applications*, Budapest, Hungary, 1970, 657-673.

[Jac97] Jacquard, B.: *Cartes et arbres : énumération, génération et dessins*, PhD thesis, LIX, Ecole polytechnique, 1997.

[JS98] Jacquard, B., Schaeffer, G.: A bijective census of nonseparable planar maps. *Journal of Comb. Theory - Serie A*, Vol 83, No 1, 1998, 1-20.

[Knu92] Knuth, D.E.: *Axioms and Hulls*. LNCS 606, Springer-Verlag, 1992.

[Ler98] Leroy, X.: The Objective Caml system, documentation and user's guide. *INRIA*, 1998.

[Lie91] Lienhardt, P.: Topological Models for Boundary Representation: a Comparison with N-dimensional Generalized Maps. *Computer Aided Design*, Vol 23, No 1, Butterworth-Heinemann, 1991, 59-81.

[Män82] Mäntylä, M., Sulonen, R.: GWB : A Solid Modeler with Euler Operators. *IEEE CG&A*, Vol 2, No 7, 1982, 17-31.

[Män88] Mäntylä, M.: *An Introduction to Solid Modeling*. Computer Press, 1988.

[PD98] Puitg, F., Dufourd, J.F.: Formal Specifications and Theorem Proving Breakthroughs in Geometric Modelling, Proc. *Theorem Proving in HOL*, Canberra, Australia, LNCS 1419, 1998, 401-422.

[PD00] Puitg, F., Dufourd, J.F.: Formalizing mathematics in higher-logic: A case study in geometric modelling. *Theoretical Computer Science*, Vol 234, Elsevier, 2000, 1-57.

[Sch98] Schaeffer, G.: *Conjugaison d'arbres et cartes combinatoires aléatoires*. PhD thesis, LABRI, Université de Bordeaux, 1998.

[Spe90] Spehner, J.C.: *Cartes et pavages*. *In* C.R. Acad. Sciences, t. 311, Série I, Paris, 1990, 143-146.

[Spe91] Spehner, J.C.: Merging in maps and pavings. *Theoretical Comp. Sc.*, Vol 86, Elsevier, 205-232.

[Tut63] Tutte, W.T.: A census of planar maps. *Canadian Journal of Mathematics*, Vol 15, 1963, 249-271.

[Tut84] Tutte, W.T.: *Graph Theory*. *In* Encyclopedia of Mathematics and its Appl. Addison-Wesley, 1984.

[Vin83a] Vince, A.: Combinatorial maps. *J. of Combinatorial Theory*, Serie B 34, 1983, 1-21.

[Vin83b] Vince, A.: Regular combinatorial maps. *J. of Combinatorial Theory*, Serie B 35, 1983, 256-277.

[Wei85] Weiler, K.: Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments. *IEEE CG&A*, Vol 5, No 1, 1985, 21-40.