A formal specification of geometric refinements *

David Cazier and Jean-François Dufourd
Laboratoire des Sciences de l'Image, de l'Informatique
et de la Télédétection (LSIIT, UPRES-A CNRS 7005)
Université L. Pasteur, 7, rue Descartes, 67084 Strasbourg
cazier@dpt-info.u-strasbg.fr

Keywords: computational geometry, boolean operations, geometric refinement, algebraic specifications, rewrite systems.

Abstract: *The geometric refinements of 2D and 3D subdivisions are basic operations in geometric programming. They consist in partitioning their cells, i.e. their volumes, faces and edges, until no intersection exists between them and in achieving their topological restructuring. These crucial operations deserve formal and precise definitions. We present, in this paper, new methods to formalize their design. Starting from a mathematical definition of the topological models and refinements, we present an algebraic specification of the operators needed to handle them. Then, we give a high-level and complete description of the refinement processes thanks to the use of rewrite systems. This approach allows us to exhibit integrity constraints for the handled objects and to focus on the conceptual and logical aspects of the refinement, avoiding tedious details of implementation. Finally, we show how the systems are enhanced to reflect choices of implementations and algorithmic improvements.*

Introduction

The *refinement* of geometric objects and, more generally, of plane or space subdivisions is the basis of numerous treatments in geometric modeling. Roughly speaking, this operation corresponds to the merging of plane or space tesselations. It is frequently involved in the evaluation of boolean operations in modelers using the boundary representation. It can also be used to merge objects built separately and to construct 2D or 3D meshes from curves or pieces of surfaces that are more easily handled than complex faces or volumes.

In a practical way, the refinement of two subdivisions, i.e. two partitions of the 2D or 3D space in distinct *cells* (vertices, edges, faces and volumes for the 3D case), consists in producing a new subdivision that contains the cells of the starting ones that have been cut out along their intersections. The incidence and adjacency relationships between these transformed cells and the inserted ones are maintained and completed during the whole process.

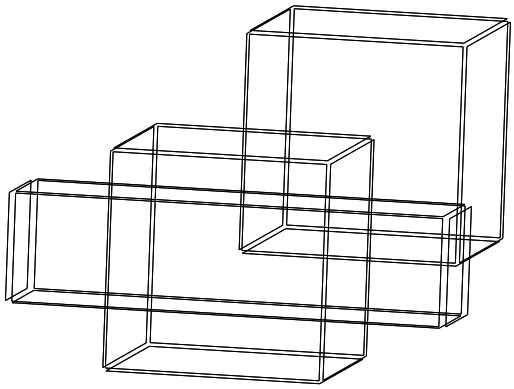


Figure 1: *Three parallelepipeds grouped in a set of cells containing intersections. (The volumes are exploded to make the faces visible.)*

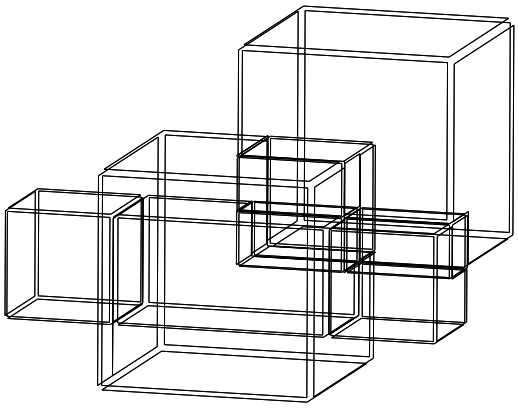


Figure 2: *The result of the refinement is the subdivision in volumes of the 3D space.*

The refinement is carried out for any number of subdivisions by grouping them in a set of cells that may contain intersections or overlaps (see Fig. 1 for a 3D example). The refinement of this set aims to transform it into a subdivision that correctly mod-

*This research is supported by the *PRC-GDR de Programmation* and the *PRC-GDR Algorithme, modles et infographie* (MENRT and CNRS).

els a *partition* of the *embedding* space (see Fig. 2). The result of a boolean operation between the starting objects is obtained, from the refinement, by a selection or extraction of the required parts.

For the 2D case, the refinement is a generalization of the line arrangements problems [1, 2] or of the building of the trapezoidal map of a set of segments [3]. The 3D refinement has mainly been studied in the scope of classical 3D boolean operations [4, 5, 6, 7, 8, 9], but also for general nD booleans [10] or specific models like the SGC one [11].

The refinements have been studied many times, but problems remain with regard to their precise definitions and reliabilities. The robustness problems are mainly rooted in approximations and round-off errors due to floating-point arithmetic. As pointed out in [12], the refinement problem is a prototype of the difficulties encountered with the approximation issues.

Numerous solutions have been proposed, including the limitation of the redundancy of numerical data and the addition of symbolic reasoning systems insuring against geometrical decisions that are in contradiction with previous ones [6].

Other purely numerical approaches aim to control and limit the round-off errors by the use of floating-point intervals [13], rationales [14], multi-precision integer [15] or lazy rational arithmetic [9].

If these studies seem to successfully answer the numerical questions, they still leave some problems unsolved. The difficulties mainly come from the need of a precise and rigorous definition of the refinement whose intrinsic complexity is increased by two constraints: to keep the description close enough to an implementation and to take into account the efficiency of the algorithms that is essential in practice.

In most approaches, the implementation considerations are emphasized. In these cases, the refinements are defined for objects whose topological structure is not explicitly given (for instance objects only regarded as lists of faces or edges). Thus, the topology is often introduced on an ad hoc basis and only for complexity reasons.

Indeed, the data structures are often extended to optimize the computations. That notably includes redundant pointers added to speed up the traversals of cells, bounding boxes used to avoid extra intersection tests and, often, links to the priority queues and dictionary that are needed for the plane-sweep algorithms or for a multidimensional searching of the couples of secant bounding boxes in 3D.

The consequences are multiple. Firstly, it is impossible to express precise integrity constraints for the objects handled and thus serious doubts remain about the exact domain of application of the proposed operations. Secondly, these informal definitions may lead to confusions between the data structures that implement the topological model and those that are added for efficiency reasons.

Thirdly, in those approaches, the corresponding algorithms are often described informally, in terms of concrete data structures, and with a procedural point of view. All the mentioned improvements are usually described for a general position case that excludes the cases that need specific treatments. Therefore, if the results are proved for the general case, they cannot be assumed for all particular cases and for the implemented extended structures. In the same way, such descriptions prevent any formal proof of the correctness of the algorithms.

On the other hand, there are works that put forward a precise mathematical definition of the objects and a formal description of the refinement process. If they answer the previously mentioned problems, the transition from those formal descriptions to an efficient implementation of the corresponding algorithms is still a task often more arduous than simple algorithm design.

To bridge the gap between a desirable formalism and the practical requirements, we present a new methodology for the refinement design. We combine an algebraic model of subdivisions with the use of formal methods that are subjects of increasing interest in software design. Precisely, we use *algebraic specifications* [16, 17, 18] allied to *rewriting* [19, 20].

The principles are as follow: we start from a mathematical definition of the topological model and its refinement. We give a high-level description of the geometric objects through an algebraic specification of the model and of the operators we need to handle it. The refinement process is then defined by a *rewrite system* that describes a set of elementary and independent transformations. Those *rewrite rules* are applied successively to the starting objects until the expected result is obtained.

At this level, we are able to prove some logical properties of the process and to check that the defined operations verify their mathematical definition and the integrity constraints. The next step consists in enhancing the rewrite system with *control structures* and new rules in order to reflect choices of implementation and algorithmic improvements.

Although any other equivalent model could be a possible candidate, we choose to use the model of *embedded combinatorial maps* [21] to describe and handle the *topology* both in the 2D and 3D cases. The main reason is that this model is simple, well defined and naturally supports the *embedding*. The topology defines the object cells and their adjacency relationships, while the embedding defines the positions and shapes of the cells. This model provides

a precise and concise description of subdivisions and is algebraically defined by a mathematical description of its properties and integrity constraints. Therefore, maps are defined regardless of all implementation and provide a good starting point for our purposes.

We start with a detailed study of the 2D case that will permits us to present the formalisms used. This first part includes the definition of the 2-maps and 2D refinement, their specifications and an outline of the proof of the *convergence* of the corresponding rewrite system. The 2D case ends with the enhancements that lead to a high-level description of a plane-sweep strategy.

In the second part, we tackle the 3D refinement. We discuss the different cases of face/face intersection and show how they lead to a rewrite system for the 3D refinement. Then, we explain how classical strategies can be taken into account with our methods. Finally, we present examples of refinement and discuss the way our approach makes easier the prototyping and further implementations of the objects and processes described.

The 2D refinement

Definition and properties of 2-maps

Let us recall some basic notions on maps. A *combinatorial 2-map* is a 3-tuple (B, α_0, α_1), where B is a finite set whose elements are called *darts*, α_0 is an involution on B without fixed points, i.e. a permutation such that $\alpha_0(\alpha_0(x)) = x$ and $\alpha_0(x) \neq x$, for all x, and α_1 is a permutation on B.

For any permutation σ on B, the orbit $\langle\sigma\rangle(x)$ of x with respect to σ is the set $\{x, \sigma(x), \ldots, \sigma^k(x)\}$, where k is the smallest positive integer such that $\sigma^{k+1}(x) = x$. Clearly, all elements of $\langle\sigma\rangle(x)$ have the same orbit. The darts of a same orbit with respect to α_i are said to be i-linked.

Since α_0 is an involution without fixed points, for any dart x, $\langle\alpha_0\rangle(x) = \{x, \alpha_0(x)\}$. Such an orbit, always formed by exactly two 0-linked darts, is called a *topological edge*. The same way, an orbit with respect to α_1 is called a *topological vertex*. Intuitively, a vertex is a sequence of 1-linked darts.

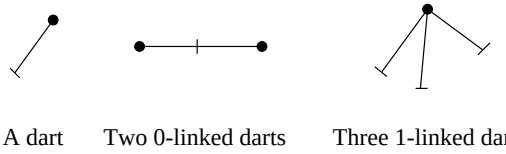


Figure 3: *Representations of darts and links.*

In the plane, the darts are usually interpreted as half-edges. Fig. 3 presents the drawing conventions with half-segments associated to darts. Two 0-linked darts are drawn as segments. The 1-links are not explicitly represented, but the darts of a vertex share the extremities of the half-segments used to depict them.

Maps provide a simple way to traverse faces. If we consider a face as a sequence of darts, then, the successor of x in its face is $\varphi(x) = \alpha_1(\alpha_0(x))$. The orbit $\langle\varphi\rangle(x)$ is called the *oriented face* of x.

Example 1 *Figure 4 shows a 2-map with* $B = \{1, \ldots, 7, -1, \ldots, -7\}$, $\alpha_0 = (-1, 1)\ (-2, 2)\ (-3, 3)\ (-4, 4)\ (-5, 5)\ (-6, 6)\ (-7, 7)$, $\alpha_1 = (1, 2)\ (-2, 3)\ (4, -6)\ (-3, -4, 7)\ (-7, 6, 5, -1)(-5)$.

The used cyclic notation $\sigma = \ldots(x_1, \ldots, x_n)\ldots$ *means that for all* i *in* $[1, n-1]$, $\sigma(x_i) = x_{i+1}$ *and that* $\sigma(x_n) = x_1$.

Thus $\alpha_0(1) = -1$, $\alpha_0(-1) = 1$, *and the orbit* $\langle\alpha_0\rangle(1) = \{1, -1\}$ *defines an edge. Similarly,* $\alpha_1(6) = 5$, $\alpha_1(5) = -1$, $\alpha_1(-1) = -7$, $\alpha_1(-7) = 6$, *thus* $\langle\alpha_1\rangle(6) = \{5, -1, -7, 6\}$, *and the vertex of dart* 6 *contains darts* 5, -1, -7 *and* 6. □

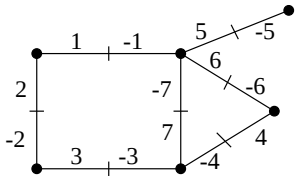


Figure 4: *A combinatorial 2-map embedded in the plane without self-intersection.*

Example 2 *For the 2-map in Fig. 4, we have* $\varphi(-7) = -3, \varphi(-3) = -2, \varphi(-2) = 1, \varphi(1) = -7$, *and thus* $\langle\varphi\rangle(-7) = \{-7, -3, -2, 1\}$ *is the face of* 7. *It corresponds to the square face of the figure. The same way, darts* 4, 7 *and* 6 *correspond to the triangular face and darts* -4, -6, 5, -5, -1, 2 *and* 3 *form the external face.* □

The topology of all subdivisions of closed oriented surfaces can be modeled by a 2-map. As for all this kind of subdivisions, the well-known Euler's characteristic $\chi(M)$ and genus $g(M)$ can be defined for any map M. If we denote by $|\sigma|$ the number of orbits of a permutation σ, then $\chi(M) = |\alpha_1| - |\alpha_0| + |\varphi|$ and $g(M) = 1 - \frac{1}{2}\chi(M)$, the genus theorem saying that $g(M)$ is always a non negative integer.

Let us notice here that we only use maps to model subdivisions of the plane. Topologically speaking, the refinement consists in building a *planar* map, i.e. a map whose genus is null, from any given map and according to its geometry.

The geometry of a map is defined through its embedding in the plane, i.e. the embedding of all its

darts. This way, vertices, edges and faces are respectively associated with points, line segments and polygons.

Formally, a 2-map M (linearly) embedded in the plane is a 4-tuple $(B, \alpha_0, \alpha_1, \pi_0)$, where (B, α_0, α_1) is a 2-map and π_0 is a function that maps each dart on the point on which its vertex is 0-embedded. The embedding of dimension 1 and 2 are defined implicitly from π_0.

Thereby, the 1-embedding $\pi_1(x)$ of x is the interior of the segment $[\pi_0(x), \pi_0(\alpha_0(x))]$. In the same way, the 2-embedding $\pi_2(x)$ of x is the interior of the polygon $\{\pi_0(y), \pi_1(y)\}_{y \in \langle\varphi\rangle(x)}$ defined by the sequence of points and segments of its boundary. Thanks to the map's orientation, for all dart x, the polygon $\pi_2(x)$ always lies on the right of the segment $\pi_1(x)$.

An embedded 2-map can model a set of segments and polygons in the plane with some incidence and adjacency relationships. These relationships may be partial, missing or even inaccurate as regards the orientation. Moreover, without geometrical constraints, a map may contain self-intersections or overlappings between its cells and connected components.

Such a map is sufficient to model the starting set of 2D subdivisions we want to refine. The 2D refinement then consists in transforming it into a map that models a partition of the plane, i.e. a planar map embedded without self-intersection. In the following, we say that such a map is *well embedded*. Let us notice that a map that can be well embedded in the plane is necessarily planar [22]. Conversely, a non planar map can never be well embedded in the plane.

Formally, let $\Pi(B) = \{\pi_0(x), \pi_1(x), \pi_2(x)\}_{x \in B}$ be the set of embeddings of a map M, then M is well embedded if:

$$(a) \quad \bigcup_{p \in \Pi(B)} p = \mathbb{R}^2;$$

$$(b) \quad \forall p, q \in \Pi(B), \quad p \neq q \Rightarrow p \cap q = \emptyset;$$

As the embeddings of edges and faces are implicitly defined, we can express these two conditions in a more practical way. Let $angle(x)$ be the angle of the segment $\pi_1(x)$ with respect to a given axis of the plane. Then, a map is well-embedded if the five following local conditions hold, for all darts x and y of M:

$$(i) \quad \langle\alpha_1\rangle(x) \neq \langle\alpha_1\rangle(y) \Rightarrow \pi_0(x) \neq \pi_0(y);$$

$$(ii) \quad \pi_0(x) \notin \pi_1(y);$$

$$(iii) \quad \langle\alpha_0\rangle(x) \neq \langle\alpha_0\rangle(y) \Rightarrow \pi_1(x) \cap \pi_1(y) = \emptyset;$$

(iv) the sequence $\left\{ \text{angle}(\alpha_1^k(x_0)) \right\}_{k \in [0,\ldots,n]}$ is increasing, n being the smallest integer such that $\alpha_1^{n+1}(x_0) = x_0$ and $x_0 \in \langle\alpha_1\rangle(x)$ such that $angle(x_0) = \min\{\text{angle}(y)\}_{y \in \langle\alpha_1\rangle(x)}$;

(v) $\pi_1(x) \neq \emptyset$.

Condition (i) means that all darts 0-embedded on the same point belong to the same vertex, which implies that the 1-links are complete. Conditions (ii) and (iii) make sure that there are neither secant edges nor vertex incident to an edge.

Condition (iv) requires some explanation: the sequence of the angles of the darts of a vertex is increasing when starting from the dart x_0 with the minimal angle. It means that as the edges of a given vertex are examined following the 1-links, the associated segments turn counter-clockwise around the vertex. A vertex that satisfies this condition is said to be *sorted* (see Fig. 5 for examples).

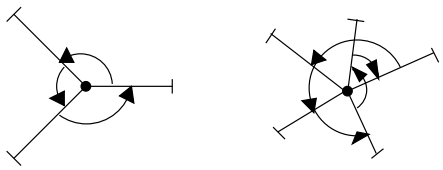


Figure 5: *Circle arcs represent 1-links*

Intuitively condition (iv) insures a good orientation of the map and implies that the faces do not fold back over themselves. All these conditions also imply that the 2-embedding do not overlap.

The last condition (v) is not necessary and is only added to obtain a minimal representation, with regard to the number of darts.

Formal specification of maps

The mathematical definition we give for maps and 2D refinement is limited because it only describes the static conditions that musr satisfy a refined map, but do not say how it may be obtained. As we said previously, our approach is to adopt a more constructive point of view.

In fact, we need to specify how maps are built and a set of operations required to perform the refinement. Thus, the second step of our formalization is to algebraicly specify [16] the combinatorial maps and the operators needed to handle them.

Various detailed specifications of maps have been proposed [23, 24]. Here, we present a simplified version as we want to focus on the refinement problem. Maps can be defined from three basic functional *generators*: v, iee, $l1$ as shown in [25]. Generator v

creates an empty map, $iee(M, x, y, p_x, p_y)$ inserts an embedded edge $\{x, y\}$ in map M, i.e. two 0-linked darts x and y respectively 0-embedded on points p_x and p_y, and $l1(M, x, y)$ 1-links dart x to dart y.

Basic *destructors*, i.e. inverse functions that remove links or edges, are also defined. All these basic atomic operations are then used to define more complex functions similar to the well-known Euler's operators [26, 27].

Table 1 shows a small piece of our specification. The language used is ad hoc but rather close to OBJ3 [28]. This specification module extends a BASE module that defined booleans and points and is not detailed here. The first three axioms indicate that the basic generators may permute. Thus, each map is represented by a class of first order congruent terms, where the applications of iee and $l1$ are the same but permuted.

A set of functional *selectors* on and *constructors* of maps are then defined through a first order equational theory. Operations are described by axioms that indicate their behavior with respect to the basic generators.

For instance, operation $e(M, z)$ tests if dart z exists in map M. For that, it breaks down the term that represents M and tests, at each step, if z has just been inserted by a heading iee generator or if it exists in the subterm. The $\alpha_0(M, z)$ selector that gives the image by α_0 of dart z in map M, is defined the same way: it breaks down the term M to find the iee generator used to insert dart z.

All the specified operators are constrained by preconditions ensuring that the constructed terms always represent 2-maps that satisfy their mathematical definition. For instance the preconditions of $iee(M, x, y, p_x, p_y)$ says that x and y should be distinct and not exist in M, which implies that the α_0 selector is really an involution.

The underlying equational logic can be used to prove these properties as explained in [29]. These proofs are mainly done by structural induction on the shape of the terms that represent the maps. Here it is impossible to give all the axioms, so only some typical ones are written in Table 1. The full specification may be found in [30].

In the following, to define the refinement, we use topological selectors to test the equality of two (topological) vertices or two edges: $eqv(M, x, y)$ and $eqe(M, x, y)$, and geometrical selectors to test the equality of the embedding of vertices or edges: $eqev(M, x, y)$ and $eqee(M, x, y)$. Let us notice that to compare two cells (vertices or edges), we only need two darts, i.e. one by cell.

For instance the specification of $eqv(M, x, y)$ uses the auxiliary eqv' function that describes the traversal of the vertex of x. Dart x_0 is the starting point and the eqv' axiom says that y is tested against all darts of the vertex of x until x_0 is reached again.

Table 1: *2-maps specification*

Spec 2MAP **extends** BASE **by**

Sorts $Dart, 2Map$

Operators

$v : \longrightarrow 2Map$
$iee : 2Map\ Dart\ Dart\ Point\ Point \rightarrow 2Map$
$l1 : 2Map\ Dart\ Dart \longrightarrow 2Map$
$e : 2Map\ Dart \longrightarrow Bool$
$\alpha_0 : 2Map\ Dart \longrightarrow Dart$
$eqv : 2Map\ Dart\ Dart \longrightarrow Bool$
$cutee : 2Map\ Dart\ Point \longrightarrow 2Map$

Axioms $(M : 2Map\ ;\ x, y, z, t : Dart\ ;$
$p_x, p_y, p_z, p_t : Point)$

$l1(l1(M, x, y), z, t) = l1(l1(M, z, t), x, y)$
$l1(iee(M, x, y, p_x, p_y), z, t) =$
$\quad iee(l1(M, z, t), x, y, p_x, p_y)$
$iee(iee(M, x, y, p_x, p_y), z, t, p_z, p_t) =$
$\quad iee(iee(M, z, t, p_z, p_t), x, y, p_x, p_y)$

$e(v, z) = false$
$e(l1(M, x, y), z) = e(M, z)$
$e(iee(M, x, y, p_x, p_y), z) =$
$\quad (z = x) \vee (z = y) \vee e(M, z)$

$\alpha_0(iee(M, x, y, p_x, p_y), z) =$
$\quad$ **if** $z = x$ **then** y
$\quad$ **else if** $z = y$ **then** x
$\quad\quad$ **else** $\alpha_0(M, z)$

$eqv(M, x, y) = eqv'(M, x, x, y)$
$\quad$ **with** $eqv'(M, x_0, x, y) =$
$\quad\quad$ **if** $x = y$ **then** $true$
$\quad\quad$ **else if** $\alpha_1(M, x) = x_0$ **then** $false$
$\quad\quad\quad$ **else** $eqv'(M, x_0, \alpha_1(M, x), y)$

$cutee(iee(M, x, y, p_x, p_y), z, p_z) =$
$\quad$ **if** $z = x \vee z = y$ **then** $l1(l1(M', x', y'), y', x')$
$\quad$ **else** $iee(cutee(M, z, x', y'), x, y, p_x, p_y)$

$\quad$ **with** $(x', y') = newdarts(M)$
$\quad\quad M' = iee(iee(M, x, x', p_x, p_z), y, y', p_y, p_z)$

Preconditions

$\quad$ **prec** $iee(M, x, y, p_x, p_y) \equiv$
$\quad\quad x \neq y \wedge \neg e(M, x) \wedge \neg e(M, y)$

End

We also use constructors that modify the topology and the embedding of a map. For instance, $cutee(M, z, p_z)$ cuts the edge of z at point p_z (see Fig. 6). To do that it breaks down the term to

find the generator *iee* used to insert the edge of z, denoted $\{x, y\}$ with $z = x$ or $z = y$. Then it removes the occurrence of this generator and replaces it with the insertion of the two new edges $\{x, x'\}$ and $\{y, y'\}$. Finally, 1-links are placed between x' and y' to build the new vertex.

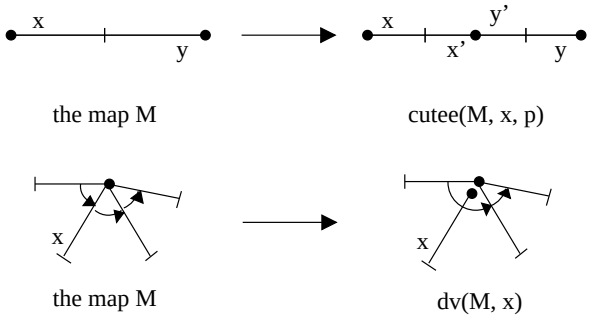
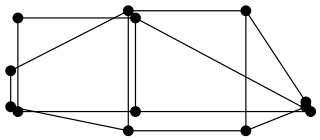


Figure 6: *Illustration of dv and cutee*

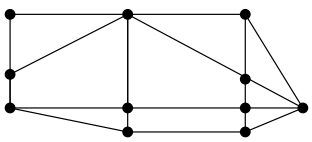
Operator $merge(M, x, y)$ merges the vertices of x and y, reordering the 1-links of these two vertices so that the final vertex is sorted. Finally, the destructor $dv(M, x)$ deletes dart x from its vertex by removing its 1-links (see Fig. 6). Other easily understandable selectors appear in the rules of the rewrite systems given later.

Rewrite system for the refinement

The refinement of a map consists in transforming it with the previously mentioned operators until it satisfies the conditions of well embedding (see Fig. 7). All algorithms that perform such a refinement use, in practice, the same kind of operations. The only real difference between the algorithms is the strategy chosen to apply these modifications.



(a) The starting map made from two overlapping subdivisions



(b) Its refinement

Figure 7: *An example of 2D refinement*

Thus, a good way to formally and completely describe the refinement, without getting bogged down in efficiency or implementation choices, consists in using rewriting techniques. Each basic transformation, and the conditions on with it can be applied, is nicely described by a *conditional rewrite rule*. All these elementary and independent rules are grouped in a *conditional modulo term rewrite system* [31], named R_{2D}, given in table 2.

Intuitively, the conditions of the rules test the existence of darts that do not verify one of the five conditions of well embedding. If it occurs, a geometric operator is applied to locally correct the embedding and to adapt the topology. This leads to a simple and generic definition of the a priori complex process of refinement.

In the following, the rewrite rules are depicted in a very visual *fractional* fashion. For each rule, the numerator represents the starting map and the denominator the map that results from the rewrite step. The rules can be applied only when the conditions described after the **if** are satisfied.

To condense the rules' description, we adopt the following convention: the term $x \in M$ means that x appears as a *Dart* parameter of some generator used to build M. Thus, $x \in M$ is a shortcut that replaces the four possible forms of M: $iee(M', x, y, p_x, p_y)$, $iee(M', y, x, \ldots)$, $l1(M', x, y)$ and $l1(M', y, x)$.

This avoids writing the same rule four times with the different shapes of M, but the same conditions and transformations. For instance, one of the four forms of rule R_1 is:

$$R_1 : \frac{l1(M, x, y)}{dee(l1(M, x, y), x)} \textbf{ if } nullee(l1(M, x, y), x)$$

Therefore, we have a true rewrite system in the sense of [31] without introducing variables in the right-hand sides, contrary to what may appear at first sight.

The rules of R_{2D} are depicted graphically in Fig. 8. Some new 0-embeddings are inserted during the refinement and deserve to be highlighted. They are displayed as boxes linked to a vertex by dotted lines and containing the name of a point as mentioned in the text.

Let us now detail the meaning of each rule and the well embedding conditions (w.e.c.), defined in section , it deals with. Rule R_1 deletes a null edge, represented in Fig. 8 as a loop, that violates w.e.c. (v). It means that if there exists a dart x in map M $(x \in M)$ that belongs to a null embedded edge $(nullee(M, x))$, then its edge is deleted from M $(dee(M, x))$.

If two edges are superposed that violates w.e.c (iii), rule R_2 deletes the second one. Formally, if there exist two darts x and z whose edges are distinct $(\neg eqe(M, x, z))$, but 1-embedded on equal segments $(eqee(M, x, z))$, then the edge of z is deleted.

Rule R_3 deals with w.e.c. (ii) and thus cuts in two parts an edge if there exists a vertex incident to

Table 2: *Rewrite system R_{2D} for the refinement of 2-maps*

$$R_1 : \frac{M}{dee(M,x)} \textbf{ if } \begin{cases} x \in M \\ nullee(M,x) \end{cases}$$

$$R_2 : \frac{M}{dee(M,z)} \textbf{ if } \begin{cases} x \in M \wedge z \in M \\ \neg eqe(M,x,z) \\ eqee(M,x,z) \end{cases}$$

$$R_3 : \frac{M}{cutee(M,x,p} \textbf{ if } \begin{cases} x \in M \wedge z \in M \\ \neg nullee(M,z) \\ incident(z,x) \end{cases}$$

with $p = gem0(M,z)$

$$R_4 : \frac{M}{cutee(cutee(M,x,i),z,i)} \textbf{ if } \begin{cases} x \in M \wedge z \in M \\ secant(M,x,z) \end{cases}$$

with $i = intersection(M,x,z)$

$$R_5 : \frac{M}{merge(M,x,z)} \textbf{ if } \begin{cases} x \in M \wedge z \in M \\ \neg eqv(M,x,z) \\ eqev(M,x,z) \\ mergeable(M,x,z) \end{cases}$$

$$R_6 : \frac{M}{dv(M,x)} \textbf{ if } \begin{cases} x \in M \\ \neg \text{sorted}(M,x) \end{cases}$$

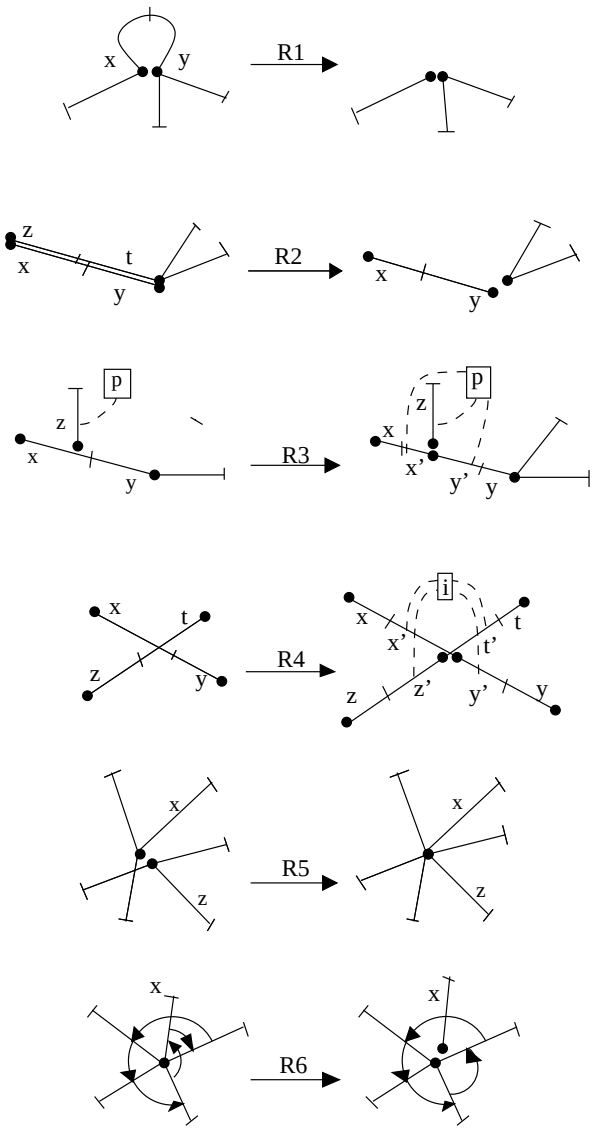


Figure 8: *Graphical illustration of R_{2D}*

it. Formally, if there exists a dart z 0-embedded on a point p that is incident to the edge of a dart x and if z does not belong to a null edge, then the edge of x is cut at point p. Point p is obtained by the selector $gem0(M,z)$ that returns the 0-embedding of dart z in M.

Rule R_4 deals also with w.e.c. (iii) and realizes the intersection cutting. If there are in M two darts x and z whose edges are 1-embedded on secant segments, then the two edges are cut at their intersection point i. This point is obtained by the $intersection(M,x,z)$ operation.

The two last rules handle vertices. Rule R_5 merges two distinct vertices that does not verify the w.e.c. (i). If there are in M two darts x and z that belong to distinct vertices ($\neg eqv(M,x,z)$) and are 0-embedded on equal points ($eqev(M,x,z)$), then the two vertices are merged. This merging can be done if the two vertices are mergeable, i.e. if they are both sorted.

Finally, rule R_6 destroys non sorted vertices. If a dart x belongs to a non sorted vertex of M, then it is deleted from this vertex. This rule is applied until the vertex is sorted or contains only one dart. After that, the deleted darts are correctly reinserted in the vertex by R_5 which allows the handling of w.e.c (iv).

Logical properties

One of the interest of this formal approach lies in the fact that at this level we are able to prove logical properties of the described process. First, the rewrite system R_{2D} is *terminating*. That means that there does not exist any infinite sequence of rule applications.

To formally prove that, we use the techniques of

[32] and construct a measure m of maps that reflects the progress of the refinement. The measure counts the number of null edges, of couples of equal or secant edges, of overlappings and of non sorted vertices. Let us note that m is also algebraicly specified. It is then easy to prove that each rule application decreases this measure with lower bound 0, which leads to the termination of the system.

Let us examines informally, the proof for instance for R_1. The conditions to start R_1 are $x \in M$ and $nullee(M, x)$. The precondition of *nullee* stipulates that x is 0-linked to another dart, we denote by y, and that x and y are embedded on points that we denote p and q.

Applying some permutations in term M, we can find a term of the class of M that has the shape $em0(em0(l0(M', x, y), x, p), y, q)$, which is proven by structural induction on the shape of the terms. This step is an inductive proof of theorems, valid in the finitely generated algebras [33] corresponding to the specification.

Rule R_1 transforms M in $dee(M, x)$ which is proven to be equal to M' due to the axioms of the specification of *dee*. Finally, the definition of the measure m is used to prove that $m(M) > m(M')$, the number of null edges being reduced. The other rules are handled, rather easily, in the same way.

The *confluence* of the rewrite system can also be proven. Roughly speaking, this means that the result of the refinement process does not depend on the order in which the rules are applied. The proof of confluence is more difficult because the rewriting is conditional and done modulo the permutations. Moreover, two different terms may represent two isomorphic maps, i.e. two maps where the links and the embeddings are the same but the names of the darts are permuted. We have to consider such maps as equal.

However, the initiating conditions of the rules are separated which simplifies the study, i.e for a given map, the same dart or pair of darts cannot initiate two distinct rules. For instance, two secant darts cannot have equal vertices.

The only way we have found to prove the local confluence is the exhaustive analysis of *critical pairs*, i.e. cases where two rules may be initiated by two different darts or couples of darts. Here again the preconditions are used to obtain the shape of the terms to be rewritten. Then, we can check by hand that critical pairs are joinable [30].

Intuitively, this proof amounts to checking that the geometrical operators may permute and do not affect the geometrical properties of the map. For instance, if there exist two distinct couples, c_1 and c_2, of secant edges, it is easy to check that applying the rule R_4 to c_1 and then to c_2 leads to the same result as applying R_4 to c_2 first.

Our rewrite system is therefore *convergent*. This implies that, for each map, the rewriting leads to one unique *normal form* modulo map congruence. Thus, the term rewrite system may be seen as a function of map normalization which projects any map into its refinement. The convergence gives us the possibility to choose any convenient or efficient strategy for rule application, which is crucial to deriving concrete algorithms.

Another important tool provided by this formalism lies in the possibility to prove that the refinement of a map leads to the expected result. Indeed, the mathematical definition of well embedding can easily be translated in the equational theory of the specification. We can then prove that the normal form of a map verifies this definition.

The proof is similar to the previous one and mixes structural induction with traditional equational logic. It is in fact a proof by contradiction. For instance, should the normal form contain a couple of secant edges, we would be able to use the specification of the *secant* function to deduce the shape of the corresponding term and then make permutations in this term so that the rule R_4 can be applied, which proves that this could not be a normal form.

Efficient strategies of evaluation

A naive use of the refinement rewrite system described above is to test for each dart or couple of darts if a rule can be executed. The corresponding algorithm has the following shape:

```
Repeat
    Choose x in M
    Try to execute rule 1 and 6 with x
    Repeat
        Choose z in M
        Try to execute rule 2 to 5 with (x, z)
    Until no rule can be executed with x
Until no rule can be executed
```

Such an abstract algorithm is not deterministic, because darts are randomly chosen. To describe a concrete, i.e. real and efficient, algorithm, we have to devise a strategy to choose darts. We achieve this goal by adding to the term rewrite system *control structures* that yield the dart or couple of darts that are going to be examined. A rewrite rule then describes the transformations of the map *and* those of the control structure.

Geometrical properties can be exploited to avoid examining couples of darts that cannot interact. If D and D' are the vertical lines that pass through the vertices of an edge $\{x, y\}$, then the darts that can interact with dart x or y are those whose vertices stand in the plane region lying between D and D'.

Table 3: *Rewrite system R_{Sweep} with plane sweep strategy*

$$R_{S_1} : \frac{X, Y, V, M}{d(d(X, x), \alpha_0(M, x)), Y, V, dee(M, x)} \textbf{ if } \begin{cases} x = first(X) \\ nullee(M, x) \end{cases}$$

$$R_{S_4} : \frac{X, Y, V, M}{i(i(i(i(X, x'), y'), z'), t'), Y, V, cutee(cutee(M, x, p), z, p)} \textbf{ if } \begin{cases} x = first(X) \\ z = gsecant(Y, M, x) \\ z \neq nil \end{cases}$$
with p = intersection(M, x, z)

$$R_{S_5} : \frac{X, Y, \{z\}, M}{X, Y, \{z\}, merge(M, x, z)} \textbf{ if } \begin{cases} x = first(X) \\ \neg eqv(M, x, z) \wedge eqev(M, x, z) \wedge mergeable(M, x, z) \end{cases}$$

$$R_{S_7} : \frac{X, Y, V, M}{d(X, x), i(Y, x), \{x\}, M} \textbf{ if } \begin{cases} \neg(R_{S_1} \vee \ldots \vee R_{S_6}) \\ x = first(X) \wedge leftdart(M, x) \end{cases}$$

$$R_{S'_7} : \frac{X, Y, V, M}{i(d(X, x), z), d(d(Y, \alpha_0(M, x)), z), \{x\}, M} \textbf{ if } \begin{cases} \neg(R_{S_1} \vee \ldots \vee R_{S_6}) \\ x = first(X) \wedge rightdart(M, x) \\ z = gbelow(Y, M, x) \wedge z' = gabove(Y, M, x) \\ interact(M, z, z') \wedge z \neq nil \wedge z' \neq nil \end{cases}$$

$$R_{S''_7} : \frac{X, Y, V, M}{d(X, x), d(Y, \alpha_0(M, x)), \{x\}, M} \textbf{ if } \begin{cases} \neg(R_{S_1} \vee \ldots \vee R_{S_6}) \\ x = first(X) \wedge rightdart(M, x) \\ z = gbelow(Y, M, x) \wedge z' = gabove(Y, M, x) \\ \neg interact(M, z, z') \vee z = nil \vee z' = nil \end{cases}$$

Following a well known denomination [3], we call them *active darts*.

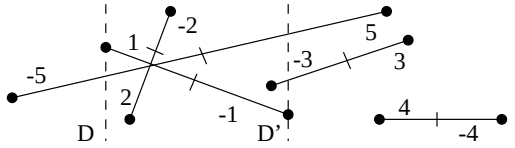


Figure 9: *In the plane region defined by edge {1,-1}, darts 2, -2 and -3 are active*

The rewrite system R_{2D} is transformed into R_{Sweep} to use this property, in a classical plane-sweep strategy [2, 3]. Darts x are examined from the left to the right. To do that, we use a *priority queue* denoted X, classically called the X-structure, as control structure. Darts z are taken in the plane region defined by the edge of x.

To avoid examining all active darts, we sort their edges from bottom to top. This way, there are only two darts to examine, namely the dart whose edge is just below edge $\{x, y\}$ and the one whose edge is just above.

To avoid computing for each current x the sorted *set of active darts*, this set is maintained by each rewrite rule as a *dictionary* denoted Y. This dictionary is sometimes called the Y-structure.

Moreover, as equal vertices are behind each other in X, the merging can only occur for two consecutive vertices. Therefore, we add a new structure, denoted V, that only contains the last used vertex (in fact the last used dart of this vertex).

The rewrite system R_{Sweep} displayed in table 3, begins with all darts sorted from the left to the right, put in the priority queue X and with Y and V empty.

Here, only the rules R_{S_1}, R_{S_4} and R_{S_5} are given. They correspond to the rules R_1, R_4 and R_5 of the first system enhanced to take into account the plane-sweep strategy. The other ones are similarly enhanced.

The changes concern updating of the control structure and the deterministic choice of darts. Two operations, i and d, are used to handle the control structures. They are used respectively to insert darts in and delete darts from X or Y.

In rule R_{S_1}, the current dart x is the first element of X and obtained by the function $first(X)$. When the rule is applied, the darts of the edge of x, i.e. x and $\alpha_0(M, x)$, that are deleted from M are also removed from X. Thus X only contains darts that exist in M.

In rule R_{S_4}, x is also the first element of X. Dart z is obtained by the $gsecant(Y, M, x)$ operator that

searches within Y if either the edge just below or just above x is secant to the edge of x and returns it or *nil* if it cannot be found. After the cutting out, the new darts x', y', z' and t' are inserted in X. This way X always reflects the set of darts of the current map.

The last changes appear in rule 5. As we said before, the only dart that can be merged with x is the last examined dart that has been placed in V. Thus, z is taken in V.

Rules R_{S_7}, $R_{S_7'}$ and $R_{S_7''}$ are used to maintain X, Y and V when the rewrite system sweeps from the left to the right, which occurs when rules R_{S_1} to R_{S_6} cannot be executed. This last condition is the negation of the concatenation of the conditions of these rules and is condensed by the $\neg(R_{S_1} \vee \ldots \vee R_{S_6})$ formula. In these three rules dart x that becomes the last used dart, after the sweep, is placed in V.

Rule R_{S_7} handled the case where x is a left dart, i.e. x is the left vertex of its edge ($leftdart(M, x)$). In this case, it becomes active and thus is deleted from X and inserted in Y.

On the other hand, if x is a right dart it was an active dart of Y and is deactivated, i.e. removed from X and Y. In this case, the edges that are just below and above the edge of x in Y may not yet have been checked together and may interact as shown in Fig. 10. These darts z and z' are respectively obtained by the $gbelow(Y, M, x)$ and $gabove(Y, M, x)$ functions.

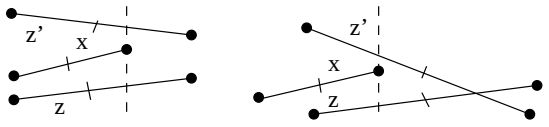


Figure 10: *Right darts inactivation*

If there is no dart below x the function *gbelow* returns *nil* and conversely for *gabove*. Thus, if z or z' are equal to *nil* or if they do not interact there is nothing more to do ($R_{S_7''}$). Else, dart z, whose edge is just below x, is removed from Y and reinserted in X to be examined once more and especially against z' ($R_{S_7'}$).

This enhanced rewrite system is a complete description of a plane-sweep strategy. It is obtained by simple transformations of the rules of the generic rewrite system R and the addition of control rules.

The interest of this approach is the clear separation between data structures used to handle maps and data structures used to improve the control and reduce the complexity of the algorithms.

A classical algorithm [1, 2] is directly derived from the rewrite system. Its complexity is in $O((n + i)\ln(n))$, where n is the number of darts and i the number of intersections. The structure of the algorithm is obtain by interpreting the control rules $R_{S_7}, R_{S_7'}$ and $R_{S_7''}$ and has the following classical shape:

```
X = darts of M sorted by x-coordinates
Y = ∅
V = ∅
While X ≠ ∅
  | x = first(X)
  | Try to execute rule 1 and 6 with x
  | get z from Y and x
  | Try to execute rule 2 to 4 with x and z
  | get z from V
  | Try to execute rule 5 with x and z
  | remove x from X
  | V = {x}
  | If x is a leftdart
  |   | Then add x to Y
  | Else {x is a rightdart}
  |   | remove α0(M,x) from Y
  |   | z is the edge below x in Y
  |   | z' is the edge above x in Y
  |   | If z and z' interact and
  |   |      z ≠ nil and z' ≠ nil
  |   |   | Then { remove z from Y
  |   |   |        insert z in X}
  |   | Endif
  | Endif
End {while X}
```

More efficient and complex algorithms, like those of [3] that adds vertical edges to make the subdivision's faces convex and have a complexity in $O(n\ln(n) + i)$ due to this improvement, can be rigorously designed that way.

Thus, we have proposed a general mechanism to constuct and describe any concrete refinement algorithm. Different control structures lead to different algorithms. A classification can thus be done, based on the kind of structures and the kind of search functions that are used.

The 3D refinement

Definition of 3-maps

We model the subdivisions in volumes of the 3D space with embedded combinatorial 3-maps. They are defined, like the 2-maps, in terms of darts and permutations, and are equivalent to the radial edges structure [21].

Thereby, a 3-map is a 4-tuple $M = (B, \alpha_0, \alpha_1, \alpha_2)$ where B is a finite set of darts, α_0, α_1 and α_2 are permutations on B.

These three α_i functions must satisfy some integrity constraints. Firstly, α_0 and α_1 are involutions, i.e. $\alpha_0(\alpha_0(x)) = x$ and $\alpha_1(\alpha_1(x)) = x$, for all x. Secondly, α_0 is without fixed point, i.e. $\alpha_0(x) \neq x$, for all x. Thirdly, α_2 is such that $\alpha_2 \circ \alpha_0$ is an involution.

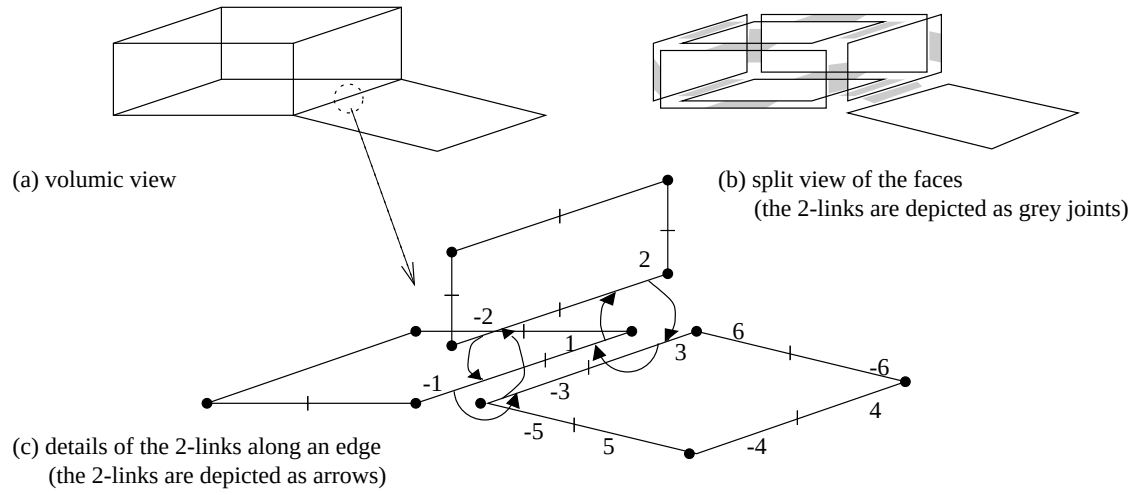


Figure 11: *A map and the details of the 2-links in a multiple edge*

As in the 2D case, darts are interpreted as oriented half-edges. The α_i functions i-links darts to form simple cells in the 3-map. Those cells are also defined as orbits with respect to α_0 and α_1.

This way two 0-linked darts make a *simple edge* and two 1-linked darts form a *simple vertex*. A sequence of simple edges linked by α_1 forms a face (see Fig. 12). In the following, we consider that α_1 is also without fixed point, i.e. that all faces are closed.

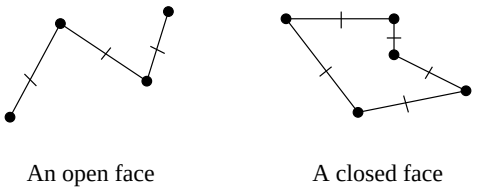


Figure 12: *Example of faces in a 3-map*

Faces are 2-linked to each others, along simple edges, to form volumes. The constraint defining $\alpha_2 \circ \alpha_0$ as an involution simply means that the 2-links are placed pairwise, on whole simple edges, and with opposite orientations (see Fig. 11).

The classical notions of (non simple) vertices and edges are defined, for the 3-maps, through an extended notion of orbits and called multiple topological vertices and edges.

If σ and τ are two permutations, then the orbit $\langle \sigma, \tau \rangle(x)$ of x is the set $\{\sigma^k(\tau^l(x))\}_{k,l \in \mathbb{N}}$. Intuitively, is contains all the darts that can be reached from x by any applications of the two functions σ and τ.

Thus, the (multiple) vertices are the orbits of $\langle \alpha_1, \alpha_2 \rangle$. Intuitively, a vertex contains all the simple vertices that are 2-linked to each other or, in other words, all the darts that can be reached following the 1- and 2-links.

The same way, the (multiple) edges are the orbits of $\langle \alpha_0, \alpha_2 \rangle$. Intuitively, an edge corresponds to a sequence of pairwise 2-linked simple edges. Let us notice that any number of faces may be 2-linked pairwise together around the same edge.

Example 3 *Fig. 11 shows a 3-map (a), its faces (b) and the detail of the 2-linkings in a multiple edge (c). For the numbered darts the permutations are, in cyclic notation:*

- $\alpha_0 = (\text{-}1, 1)(\text{-}2, 2)(\text{-}3, 3)(\text{-}4, 4)(\text{-}5, 5)(\text{-}6, 6)$*;*
- $\alpha_1 = (\text{-}3, \text{-}5)(3, 6)(4, \text{-}6)(\text{-}4, 5)\ \ldots$*;*
- $\alpha_2 = (\text{-}4)(4)(\text{-}5)(5)(\text{-}6)(6)(\text{-}1, \text{-}3, \text{-}2)(1, 2, 3)$.

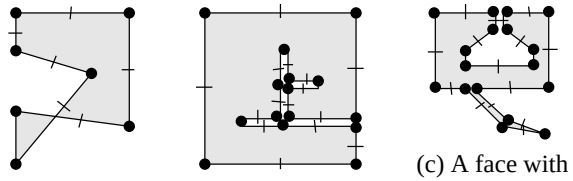
Here we have: $\alpha_0(1) = \text{-}1$ *and* $\alpha_0(\text{-}1) = 1$*, i.e.* $\{1, \text{-}1\}$ *is a simple edge;* $\alpha_1(\text{-}3) = \text{-}5$ *and* $\alpha_1(\text{-}5) = \text{-}3$*, i.e.* $\{\text{-}3, \text{-}5\}$ *is a simple vertex;* $\alpha_2(\text{-}4) = \text{-}4$*, i.e. the simple edge* $\{4, \text{-}4\}$ *is not 2-linked;* $\alpha_2(1) = 2, \alpha_2(2) = 3$ *and* $\alpha_2(3) = 1$*, i.e. the faces that contain darts* $1, 2$ *and* 3 *are 2-linked. Finally,* $\{1, 2, 3, \text{-}1, \text{-}2, \text{-}3\}$ *forms a triple edge.* □

As shown in Fig. 11, this model naturally supports the dangling faces. The fact that volumes are explicitly defined avoids the problem of non manifold edges. However our 3-maps may not contain dangling simple edge because we stipulate that the faces are closed.

The non manifold vertices can be handled by the addition of faces between the volumes sharing a non

manifold vertex. This aspect is not further detailed here as we want to focus on our methodologies.

As in the 2D case, the function $\varphi = \alpha_1 \circ \alpha_0$ gives the successor of a dart in its face. This function here is used to define an orientation of faces. Thereby, each face contains two *oriented faces*. For instance, in Fig. 11 the face {-4, 4, -6, 6, 3, -3, -5, 5} contains two oriented faces: {-4, -6, 3, -5} and {4, 5, -3, 6}.



(a) A face non connected (b) A face with interior edges (c) A face with a degenerate branch

Figure 13: *Examples of embedded faces*

Embedding and properties

The geometry of 3-maps is defined by their (linear) *embedding* in $\mathbb{R}^3$. As before, we associate to each vertex a point (0-embedding), to each edge the interior of the segment joining its two vertices (1-embedding), to each face the interior of the polygon formed by its edges and vertices (2-embedding) and, by extension, the 3-embedding of a volume is the polyhedra defined by the embeddings of its boundary.

Due to the orientation the interior of a face always lies to the right of its edges during the traversal of both oriented faces. Indeed, the two oriented faces are associated with opposite normal vectors computed by application of the right-hand rule.

Naturally, we have to impose geometric conditions to insure that the embedding remains well shaped and can be implicitly defined. On the one hand, the polygons on which the faces are 2-embedded have to be planar. Thus, as the polygons are implicitly defined, the vertices of a face have to lie in the same plane. On the other hand, the face boundaries have to be simple, i.e. not self-crossing, to insure the correct definition of their interiors.

To be precise, the interior of each face must be connected and have a non null area to make possible the computation of a normal. Thus, we reject the eight-shaped face of Fig. 13 (a) whose face interior is not connected.

But we accept faces that contain internal edges or have a degenerate embedding as shown in Fig. 13 (b) and (c) where the multiple edges have been moved further apart to be visible.

In both cases the interior and degenerate simple edges are in fact 2-linked together in multiple edges. The (c) case can possibly be used to model dangling (multiple) edges.

Let us notice that these conditions are equivalent to the conditions of well embedding for the 2-maps, limited to single faces. Thus, 2D refinement may be used as a pre-treatment of non simple faces.

For instance the face of Fig. 13 (a) can be seen as a 2-map that contains a bounded face (folded back over itself) and an unbounded one corresponding to its two oriented faces. Its 2D refinement transforms it into two bounded faces and the unbounded one linked by α_1. After deleting these 1-links the two bounded faces can be put back in the 3-map.

As previously, a 3-map embedded without other constraints may contain secant or overlapped faces and volumes. Such a 3-map is used to model the starting set of subdivisions. Its 3D refinement consists in transforming it into a well embedded map.

By extension of the 2D case, a well embedded 3-map is a map whose embedding realizes a partition of $\mathbb{R}^3$, i.e. a 3-map embedded without self-intersection of overlapping. Formally, it has to satisfy the following conditions, for all darts x and y:

(i) $\langle\alpha_1, \alpha_2\rangle(x) \neq \langle\alpha_1, \alpha_2\rangle(y) \Rightarrow \pi_0(x) \neq \pi_0(y)$;

(ii) $\langle\alpha_0, \alpha_2\rangle(x) \neq \langle\alpha_0, \alpha_2\rangle(y) \Rightarrow \pi_1(x) \cap \pi_1(y) = \emptyset$;

(iii) $\langle\varphi\rangle(x) \neq \langle\varphi\rangle(y) \Rightarrow \pi_2(x) \cap \pi_2(y) = \emptyset$;

(iv) $\pi_0(x) \notin \pi_1(y)$;

(v) $\pi_0(x) \notin \pi_2(y)$;

(vi) $\pi_1(x) \cap \pi_2(y) = \emptyset$;

(vii) the sequence $\left\{\text{angle}(\alpha_2^k(x_0))\right\}_{k\in[0,\ldots,n]}$ is increasing, n being the smallest integer such that $\alpha_2^{n+1}(x_0) = x_0$ and $x_0 \in \langle\alpha_2\rangle(x)$ such that $angle(x_0) = \min\{\text{angle}(y)\}_{y\in\langle\alpha_2\rangle(x)}$.

Conditions (i), (ii) and (iii) mean that if x and y do not belong to the same vertex, edge or face, then their 0-, 1- or 2-embeddings respectively have an empty intersection. That notably implies that their embeddings are distinct and thus that the 2-links are complete, because darts embedded on equal point, segment or polygon should respectively belong to the same vertex, edge or face.

Condition (iv), (v) and (vi), with the first three ones, assure that there is neither intersection nor overlapping between two embeddings.

Finally, condition (vii) is similar to the condition (iv) of the 2D case. Here the $angle(x)$ function returns the angle between the plane containing the face of x and a fixed plane containing the edge of x. That means that the faces that share a common edge are correctly ordered around the axis formed by this edge. This implies a good orientation of the

3-map and assures that volumes do not fold back over themselves. We say that an edge that satisfies this condition is well sorted.

These conditions can be expressed in a more useful way with the aim of building local tests for the refinement. For two distinct and non coplanar faces f and g, we denote their intersection line by $\Delta(f,g)$ (or simply Δ when the context is clear). The intersection cases excluded by conditions (i) to (vi) may only occur along Δ and, more precisely, for some segments of this line.

Thus, to exhibit these simplified conditions, let us consider the intersection of f with Δ. This intersection is a set of *section segments* that we denote $Seg(f,\Delta)$. It can be divided into two subsets: $SegIn(f,\Delta)$ and $SegOn(f,\Delta)$ that are respectively composed of segments whose interior is completely included in the interior of f and segments that lie on the boundary of f and correspond to edges of f.

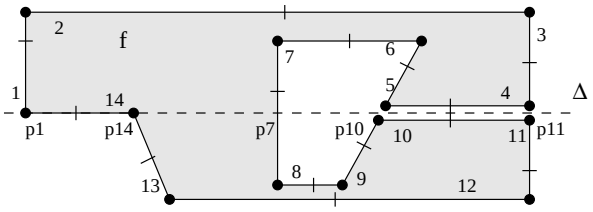


Figure 14: *Intersection between a face and a line*

Example 4 *For the face f of Fig. 14 the segment sets of the intersection between f and the line Δ are: $SegIn(f,\Delta) = \{[p_{14}, p_7]\}$ and $SegOn(f,\Delta) = \{[p_1, p_{14}], [p_{10}, p_{11}]\}$.* □

The points used to define these segments are associated with the darts whose embedding cut Δ. This way, in Fig. 14, point p_1 is the 0-embedding of (the vertex of) dart 1 and point p_7 is the intersection between Δ and the 1-embedding (of the edge) of dart 7. Thus, in the following the section segments are denoted $[p_{x,\Delta}, p_{x',\Delta}]$ (or simply $[p_x, p_{x'}]$), where $p_{x,\Delta}$ (or simply p_x) is the point defined by dart x.

Let us remark that the definition of segments is not unique. For instance, in Fig. 14, $[p_{10}, p_{11}]$ denote the same segment as $[p_5, p_4]$ or $[p_5, p_{11}]$.

For two (or more) coplanar faces, the conditions (i) to (vi) come down to the well-embedding of 2-maps. Indeed, if we consider these coplanar faces isolated from the rest of the 3-maps and embedded in their underlying plane, they form connected components of a derived 2-map.

Moreover, the 2-links existing between the darts of these faces can be interpreted as 1-links in the derived 2-map. Thus, the conditions of well-embedding given for 2-maps can be translated for coplanar faces of a 3-map, without difficulties and thus are described in detail.

Therefore, a 3-map M is well-embedded if, for all darts f and g, representing two distinct faces, and for all segments $s = [p_x, p_{x'}] \in Seg(f, \Delta(f,g))$ and $t = [p_y, p_{y'}] \in Seg(g, \Delta(f,g))$, the following conditions hold:

(i') $s \in SegIn(f,\Delta) \Rightarrow s \cap t = \emptyset$;

(ii') $t \in SegIn(g,\Delta) \Rightarrow s \cap t = \emptyset$;

(iii') $s \in SegOn(f,\Delta) \wedge t \in SegOn(g,\Delta) \Rightarrow$
$s \cap t = \emptyset$
$\vee \{s = t \wedge \langle\alpha_0,\alpha_2\rangle(x) = \langle\alpha_0,\alpha_2\rangle(y)\}$
$\vee \{p_{x'} = p_y \wedge \langle\alpha_1,\alpha_2\rangle(x') = \langle\alpha_1,\alpha_2\rangle(y)\}$
$\vee \{p_{y'} = p_x \wedge \langle\alpha_1,\alpha_2\rangle(x) = \langle\alpha_1,\alpha_2\rangle(y')\}$;

(iv') if f and g are coplanar then the derived 2-map obtained from the restriction to f and g of M is well-embedded;

(v') the sequence $\{\text{angle}(\alpha_2^k(x_0))\}_{k\in[0,\ldots,n]}$ is increasing, n being the smallest integer such that $\alpha_2^{n+1}(x_0) = x_0$ and $x_0 \in \langle\alpha_2\rangle(x)$ such that $angle(x_0) = \min\{\text{angle}(y)\}_{y\in\langle\alpha_2\rangle(x)}$;

Conditions (i'),(ii') and (iii') imply that two non coplanar faces can only have intersections along shared multiple edges and vertices. Precisely, conditions (i') and (ii') say that the section segments that belong to the interior of the faces f and g have to be disjoint from any other section segments.

Condition (iii') expresses that two segments on the boundaries of f and g are either disjoint or equal and defined by darts that belong to the same multiple edge or share a common extremity that is defined by darts that belong to the same multiple vertex.

To be complete, let us say that in condition (iii') we have only written, to simplify, $p_{x'} = p_y$ (or $p_{y'} = p_x$) to express that s and t have a common extremity. We should have added that the interior of s and t have to be disjoint, i.e. their intersection reduces to a point, and that we suppose that the two points used to define each segment are ordered from the left to the right, to avoid the handling of the symmetrical cases.

Basic geometric operators

As for the 2-maps, the 3-maps, and the operators needed to build and handle them, are algebraicly specified. This specification is quite similar to the previously given one, with one more basic generator

$l2$ used to insert the 2-links [24]. As another specification example would be of little interest here, we only informally present the operators we use further.

To carry out the 3D refinement, we need only four simple operators on top of the classical ones that allow the building and handling of maps. Those four geometrical operators are presented in Fig. 15 which depicts their actions on a map M.

The operation $iv(M, x, p)$ inserts, in M, a vertex embedded on p in the edge of x. The $ie(M, x, p, q)$ operation inserts a double edge embedded on segment $[p, q]$ in such a way that the vertex embedded on p is inserted in the edge of x.

The operation $i2e(M, x, p, q, r)$ is a shortcut to insert two double edges, embedded on $[p, q]$ and $[q, r]$. It is used to insert the edge corresponding to $[q, r]$ in the interior of the face of x, attaching it to the face boundary at point p. The last operation $cutf(M, x, y, p, q)$ cuts a face in two parts separated by a double edge embedded on $[p, q]$, between the darts x and y.

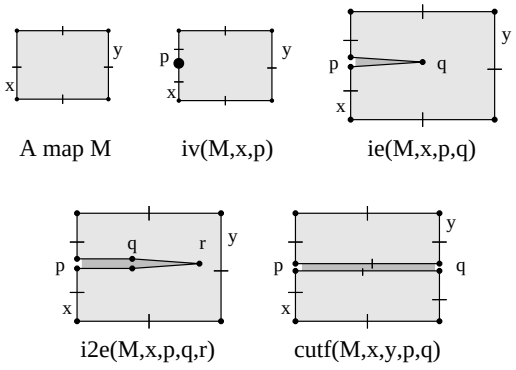


Figure 15: *Geometrical operators on 3-maps*

Finally, let us note that it can be formally proved that these operations preserve all the characteristic topological properties given in the definition of 3-maps. There is no special geometrical condition for the used points p and q except that they have to lie in the plane of the face of the given darts x or y.

However, to obtain the expected geometric results, p should belong to the edge of x and segments $[p, q]$ and $[q, r]$ should completely lie in the face interior and not cut the face boundary. These geometrical conditions are always satisfied in the following 3D-refinement process.

The high level 3D refinement

As previously, the 3D refinement is defined through a set of elementary and independent operations described by the rules the system R_{3D}. Table 4 contains the formal descriptions of rules that are graphically depicted in Fig. 16.

Each rule describes a high-level transformation of the map that is described further and only involves the four operators detailed in the previous section.

As we want to focus on face intersections, we do not detail here the aspects of rewriting techniques that we have already presented for the 2D case. Thus, we directly use the simplified syntax presented for the 2D case.

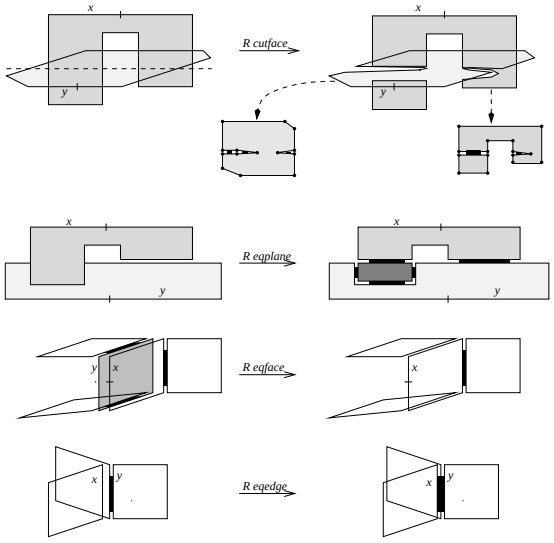


Figure 16: *Graphical illustration of R_{3D} (2-links are drawn as black joints)*

The first rule, $R_{cutface}$, realizes the intersection of faces. When the planes of the faces of two darts x and y are secant, the faces of x and y are cut along the intersection line. This operation is detailed in the next section. Small replicas of the subdivided faces are separately depicted down flat, in Fig. 16.

The second rule, $R_{eqplane}$, treats the case of two coplanar faces. As said before, this operation amounts to intersecting two polygons in a plane and is not detailed further. It uses the 2D refinement and conversion procedures to transform faces of a 3-map into a 2-map and, conversely, the bounded faces of a 2-map into faces of the 3-map.

The third rule, R_{eqface}, completes the second rule and is used when two faces are embedded on two equal polygons. In this case, one of the two faces, here the face of y, is deleted by the $delface(M, y)$ operator. The faces that are 2-linked to the deleted face are then 2-linked to the other face, by R_{eqedge}.

The last rule, R_{eqedge}, merges edges embedded on equal segments and not already 2-linked. In fact, the 2-links of edges of x and y are merged so as to sort the 2-linked faces of x and y around the axis formed by the edges.

Table 4: *Rewrite system R_{3D} for the refinement of 3-maps*

$$R_{cutface}: \frac{M}{cutface(M,x,y)} \textbf{ if } \begin{cases} x \in M \wedge y \in M \\ secant(M,x,y) \end{cases} \qquad R_{eqplane}: \frac{M}{fusionface(M,x,y)} \textbf{ if } \begin{cases} x \in M \wedge y \in M \\ eqplane(M,x,y) \\ \neg eqface(M,x,y) \end{cases}$$

$$R_{eqface}: \frac{M}{delface(M,y)} \textbf{ if } \begin{cases} x \in M \wedge y \in M \\ eqpoly(M,x,y) \\ \neg eqface(M,x,y) \end{cases} \qquad R_{eqedge}: \frac{M}{fusionedge(M,x,y)} \textbf{ if } \begin{cases} x \in M \wedge y \in M \\ eqseg(M,x,y) \\ \neg eqedge(M,x,y) \end{cases}$$

Roughly speaking, these last two rules are mainly used to link adjacent volumes. The faces of the volumes are first subdivided by the first two rules. The contact zone between the volumes is thus reduced to a set of equal faces that are pairwise merged.

To merge two faces, the rewrite system successively executes the third and fourth rules. This way, the second face is deleted by R_{eqface} and then each edge of the first face is merged by R_{eqedge} with the edges that were incident to the second one.

The rule R_{eqedge} is also used to complete the missing adjacency relationships and thus to achieve the topological restructuring of the map.

The refinement of a given map amounts to trying to execute each rule with each couple of darts (i.e. each couple of faces or of edges). This naturally leads to an algorithm whose complexity is in $O(n^2)$, where n is the number of darts.

Classical strategies that limit the number of tested couples can improve the efficiency of computations, as for instance the use of bounding boxes. In this case, a multidimensional searching of the k couples of secant boxes can be done in $O(k+n.ln^2n)$, where n is the number of boxes [34].

Such strategies can be described formally with the use of control structures enhancing the initial rewrite system, as explained in the 2D case.

Intersection of transversal faces

The more delicate part of the refinement is the intersection of non coplanar faces. In fact, the high level rule $R_{cutface}$ is defined as a set of rewrite rules, given in Table 5, that locally subdivide the two faces as shown in Fig. 17.

The principle of the subdivision is to detect the section segments of the intersection line that violate the condition of well embedding. When such a segment is found, the two faces are simultaneously subdivided by the insertion of new edges embedded on this segment and merged in a multiple edge.

By now, let us assume that we have a function $\Delta(M,x,y)$ that returns the intersection line Δ between the faces of x and y in map M, and two functions, $SegIn(M,x,\Delta)$ and $SegOn(M,x,\Delta)$, that return respectively the section segments of the intersection line Δ that are inside and on the boundary of the face of x, in a map M.

When two section segments of Δ overlap, the way the two faces will be subdivided depends on the relative arrangement of the extremities of the segments along the line Δ. Assuming that we have a coordinate system on Δ, then comparing the points amounts to comparing their abscissae in this coordinate system.

In the following and in Table 5 we write $p_x = p_y$, $p_x < p_y$, and $p_x \leq p_y$, to express that the abscissa of the point p_x defined by dart x is equal to, strictly lower than, and lower or equal to, that of p_y.

The rewrite system for the face/face intersection detailed where and how the faces are subdivided, by describing all the positional cases for two overlapped section segments $[p_x, p_{x'}]$ and $[p_y, p_{y'}]$ of the face of x and y. This amounts to studying all possible arrangements of p_x, $p_{x'}$, p_y and $p_{y'}$ along Δ.

In the rules of Table 5, x and y represent two secant and distinct faces (the conditions of $R_{cutface}$). Their conditions express in which set the section segments are chosen and the order of their extremities. Rule R_{case_1} is fully detailed. In the other ones only the pertinent information are written to avoid an overflow of formulae.

Rule R_{case_1} inserts, with the iv operator, a new vertex in the edges of x' and y that are secant at point $p_{x'}$ on Δ. In the case of R_{case_2}, the segment $[p_y, p_{x'}]$ lies in both face interiors. Thus, the rule inserts, in the two faces, a new edge whose vertices are embedded on p_y and $p_{x'}$. This operation is performed by the combination $ie(ie(M,x',p_{x'},p_y),y,p_y,p_{x'})$ of the ie operator.

For R_{case_3}, the segment $[p_y, p_{y'}]$ is completely included in segment $[p_x, p_{x'}]$ and thus lies in the interior of the face of x. This segment being already included in the boundary of the face of y, nothing has to be done for this face. On the other hand, it lies inside the face of x, but is not incident to this face boundary. The $i2e$ operator is thus used

Table 5: *Rewrite system for faces intersection*

$$R_{case_1}: \frac{M}{iv(iv(M,x',p_{x'}),y,p_y)} \textbf{ if } \begin{cases} x \in M \wedge y \in M \\ [p_x,p_{x'}] \in SegIn(M,x,\Delta(M,x,y)) \\ [p_y,p_{y'}] \in SegIn(M,y,\Delta(M,x,y)) \\ p_x \leq p_{x'} < p_y \leq p_{y'} \end{cases}$$

$$R_{case_2}: \frac{M}{ie(ie(M,x',p_{x'},p_y),y,p_y,p_{x'})} \textbf{ if } p_x < p_y < p_{x'} < p_{y'}$$

$$R_{case_3}: \frac{M}{cutf(i2e(M,x,p_x,p_y,p_{y'}),y,y',p_y,p_{y'})} \textbf{ if } \begin{cases} [p_y,p_{y'}] \in SegOn(M,y,\Delta(M,x,y)) \\ p_x < p_y < p_{y'} < p_{x'} \end{cases}$$

$$R_{case_4}: \frac{M}{cutf(cutf(M,x,x',p_x,p_{x'}),y,y',p_y,p_{y'})} \textbf{ if } p_x = p_y < p_{x'} = p_{y'}$$

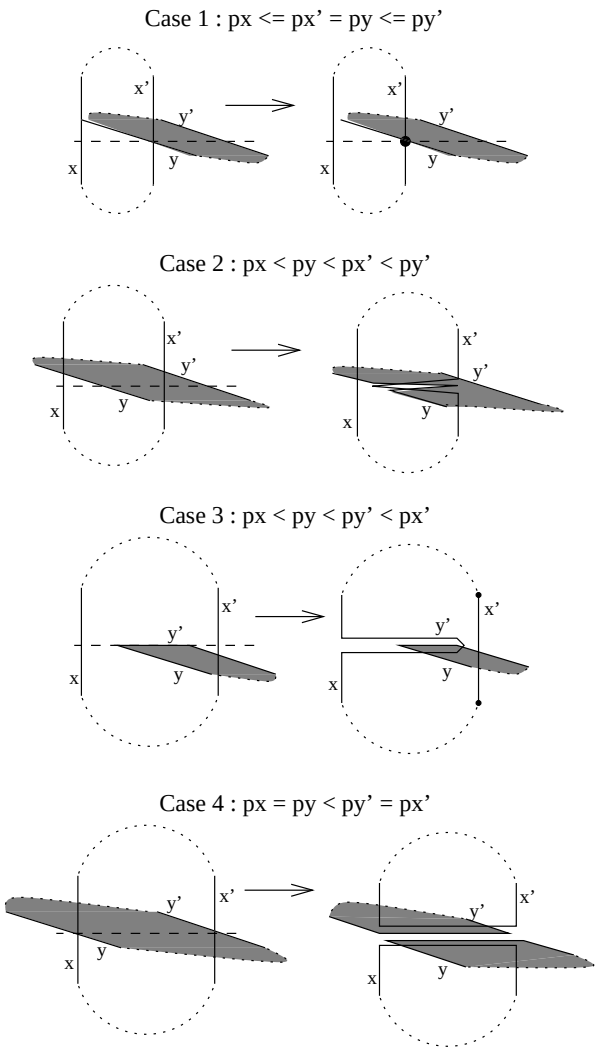


Figure 17: *Four cases of faces intersection*

to insert two double edges as previously explained.

In the last rule, R_{case_4}, the segment $[p_x,p_{x'}]$ that is equal to segment $[p_y,p_{y'}]$ separates the two faces. A double edge is thus inserted between x and x', and a second one between y and y', to cut the two faces.

There are three other cases not given here that have comparable symmetrical shapes. They all use the operators described and treat the other possible arrangements of points $p_x, p_{x'}, p_y$ and $p_{y'}$.

Each of these 7 arrangement cases is subdivided in 4 cases depending on the set, $SegIn$ or $SegOn$, in which the section segments are taken. For segments taken in $SegIn$ the cutting out are performed as described in the rules.

For a segment $[p_x,p_{x'}]$ taken in $SegOn$, the edge of x is on the Δ line and the vertices or edges insertions are not necessary and nothing is done (as shown for the face of y in R_{case_3}). The subcases study does not involve difficulties and is not detailed. Altogether, the rewrite system for the face/face intersection contains 28 rules $(4*7)$.

The computation of the two needed segment sets, $SegIn(M,x,\Delta)$ and $SegOn(M,x,\Delta)$, is done in three steps. First, the intersection of the edges and vertices of all darts of the face of x with the Δ line are computed. Then, the intersection points are sorted by increasing abscissa along Δ. Finally, these points are pairwise grouped to form the expected segments.

The only difficulty is to eliminate the possible multiple definitions of section segments that may occur when a multiple edge lies on Δ, as pointed out in the definition of $SegOn$.

An efficient strategy can be defined to intersect two faces. It consists in building first the set of section segments for the two faces. Then, to sort the segments along Δ. During this sorting the over-

lapped segments can easily be found and the rules applied where needed. This strategy has been developed in a previous paper [35] and is not detailed here.

Examples of 3D refinement

Faults refinement

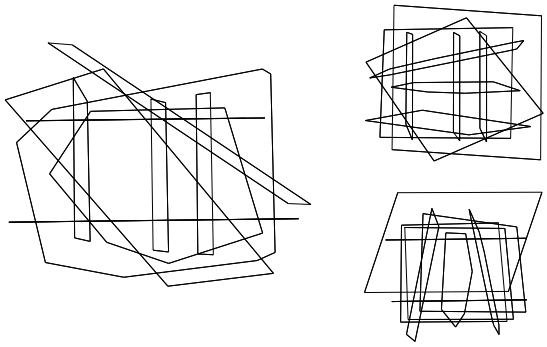


Figure 18: *A set of secant faces*

The first example comes from an experimentation of refinement in the domain of geology. Faults in the subsoil are modeled by different secant faces of a 3-map. They correspond to boundaries between geological strata. An example is given in Fig. 18. The left subfigure is a front view of the faults, while the two right ones show them from two other points of view.

The 3D refinement produces a subdivision in volumes of the modeled subsoil. Fig. 19 shows the result of the refinement. Let us notice that it contains a large number of dangling faces that, in fact, belong to the exterior volume. The computed interior volumes are highlighted in the right subfigures.

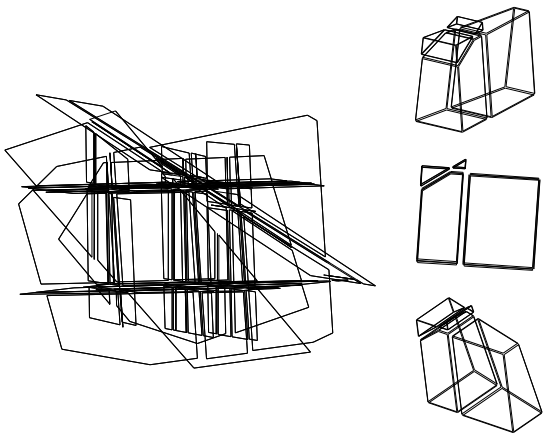


Figure 19: *Its refinement and the computed volumes*

In this limited case our 3D refinement is similar to previous ones restricted to the geology domain [36].

Polyhedral volumes refinement

In this second example, we present the refinement of more classical solids similar to those used in CAD. Fig. 21 represents four secant polyhedral solids modeled as distinct and overlapped connected components of the same 3-map.

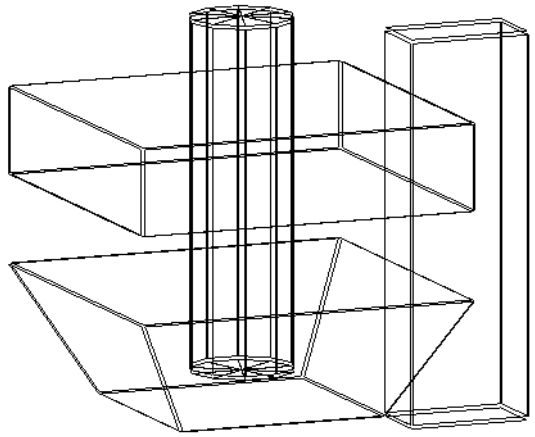


Figure 21: *A 3-map modeling polyhedral solids*

Let us notice that this map contains overlapping faces, that the cylinder goes through the flat box interior without cutting its boundary and that the rightmost long parallelepiped touches the bottom one in a single vertex.

Fig. 22 shows the resulting 3D refinement. The holes created in the faces of the flat box are modeled by inserted edges that link the outside boundary of each face to the hole boundary lying inside of it.

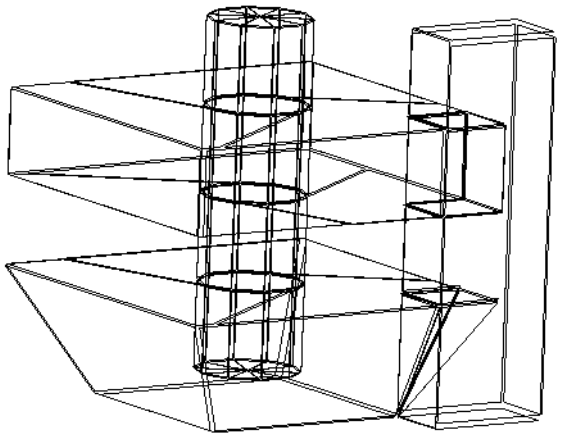


Figure 22: *The refinement of the solids*

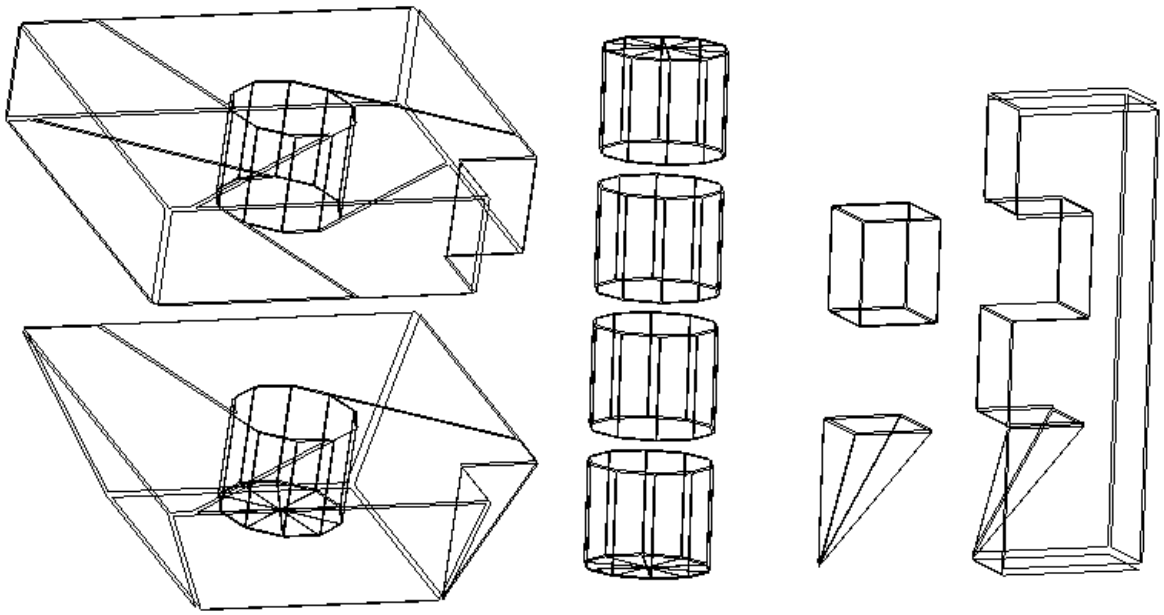


Figure 20: *Exploded view of the inner volumes*

Fig. 20 depicts the inner volumes of the refinement. We have deleted the exterior volume to highlight the interior ones. This exterior volume groups the faces that form a shell around the whole object and is implicitly defined in all 3-maps.

Thus, the figure shows the interior volumes computed by the refinement, that is to say the internal structure of the object. We have shifted them to improve the view. Some of the volumes have been rotated to provide a better viewpoint.

Intersection of two cones

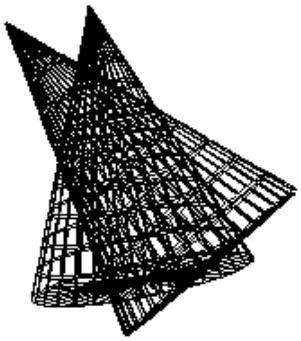


Figure 24: *Two secant cones*

The last example is the use of the 3D refinement to compute the intersections between two cones modeled with a large number of faces. Fig. 24 shows the initial objects put in a same map before their refinement.

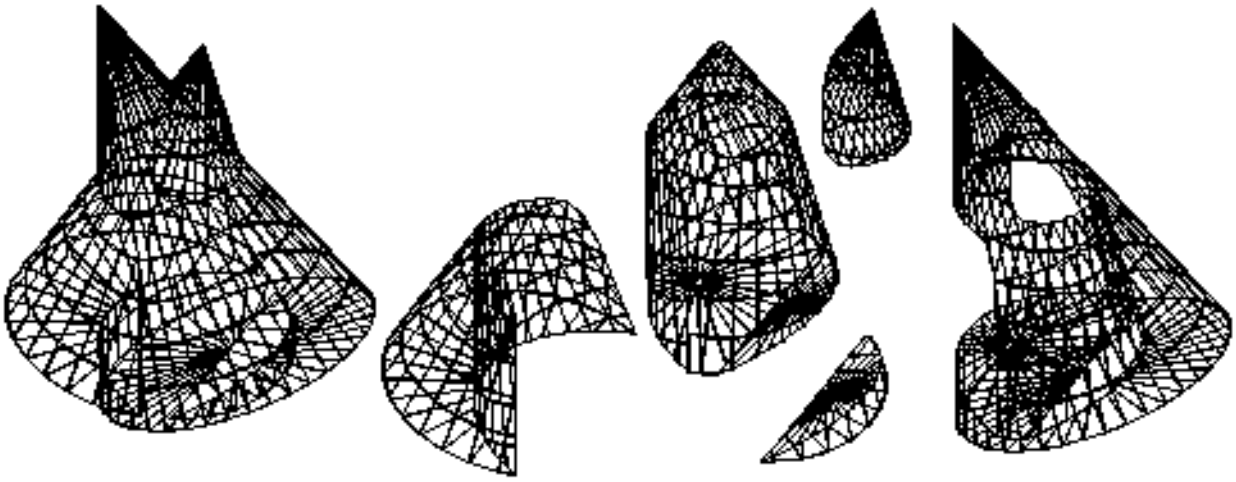
Fig. 23 shows the resulting 3D refinement where the internal structures have been separated out. The volumes obtained correspond to the possible boolean operations between the starting objects. The leftmost solid (a) is the union of the two cones, the middle one (c) is their intersection and the other ones, placed around, are the pieces that form the symmetrical difference between the two cones, i.e. (e) is the first difference and (b) plus (d1/d2) form the second one.

Conclusion

First, we have defined the refinement of subdivisions in faces and volumes of 2D and 3D spaces. The mathematical models of combinatorial maps, mainly described by permutations and orbits, have helped us to express formal conditions of well embedding in both cases.

Secondly, by the way of algebraic specifications, we have formally defined topological and geometrical operations to build and handle embedded maps. At this level we were able to prove that the manipulated objects and functions satisfy the integrity constraints given for the mathematical model.

Finally, the use of rewrite systems has lead us to express a complex problem as a collection of elementary and independent transformations. We have thus completely and formally described 2D and 3D refinements of embedded maps. We have shown how some logical properties of the described



(a) (b) (c) (d1/d2) (e)

Figure 23: *Exploded view of the volumes*

process can be proved.

Beyond the refinement of a set of subdivisions, this operation is in fact a general process of normalization of 2D and 3D subdivisions and, in fact, of all equivalent models in boundary representation.

Operations on maps and rewrite rules have first been implemented in a graphical Prolog. This logical prototyping allowed us to quickly verify in a practical way, but with reduced input data, the validity of our specifications.

Thus the formalism we used, allied to rewriting expressiveness, makes easier a proper and rigorous design of algorithms. It is also a nice starting point for rapid prototyping and further design of efficient algorithms.

The two refinements have been implemented in the C language for a Sun workstation, where different kinds of strategies were studied for efficient execution of the rewrite systems. Firstly, we have done a straightforward implementation of the simple algorithm whose complexity is in $O(n^2)$.

Next, classical methods, such as the plane sweep algorithms in 2D and the use of bounding boxes for the 3D, have been implemented to improve the complexity until achieving the best known result. Here again the clear separation of logic and control has helped us.

Questions of approximation we have not dealt with do not cast doubt on the structure of the rewrite system, because they are precisely localized. In fact, they only take place in the geometrical tests that appear in the conditions of the rewrite rules. Therefore, the use of an exact arithmetic would only affect these parts and would of course be without effect on the topological treatments.

The extensions we are currently working on aim to add to each dart an attribute that enumerates the object(s) it belong to, which extends the *active* attribute of the cells of SGC [11] and is similar to the *historic* attribute of [8]. This enhancement has already be done for the 2D case [37]

The algorithms will be completed to take into account this attribute during the refinement stage. Our goal is to directly evaluate, after refinement, the result of a classical boolean operation between many objects modeled in the same map. To sum up, the interest of this method is a sound definition of efficient boolean operations rooted in the topology of objects.

References

[1] J.L. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Computer*, 28:643–647, 1979.

[2] J. Nievergelt and F.P. Preparata. Plane-sweep algorithms for intersecting geometric figures. *Com. of ACM*, 25(10):739–747, 1982.

[3] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *Journal of ACM*, 39(1):1–54, 1992.

[4] A.A.G. Requicha and H.B. Voelcker. Boolean operations in solid modeling: boundary evaluation and merging algorithms. In *Proc. of IEEE*, volume 73, january 1985.

[5] M. Mäntylä. Boolean set operations on 2-manifolds through vertex neihborhood. *Transaction on Graphics*, 5(1):1–29, 1986.

[6] C.M. Hoffmann, J.E Hopcroft, and M.S. Karasick. Robust set operations on polyhedral solids. *IEEE Computer Graphics & Applications*, 9(6):50–59, 1989.

[7] E.L. Gursoz, Y. Choi, and F.B. Printz. Boolean set operations on non-manifold boundary representation objects. *Computer-Aided Design*, 23(1):33–39, 1991.

[8] G.A. Croker and W.F. Reinke. An editable nonmanifold boundary representation. *IEEE Computer Graphics & Applications*, pages 39–51, 1991.

[9] M.O. Benouamer, D. Michelucci, and B. Péroche. Error-free boundary evaluation based on a lazy rational arithmetic: a detailled implementation. *Computer-Aided Design*, 26(6):403–416, 1994.

[10] L.K. Putnam and P.A. Subrahmanyam. Boolean operations on n-dimensional objects. *IEEE Computer Graphics & Applications*, 6(6):43–51, 1986.

[11] J.R. Rossignac and M.A O'Connor. SGC: A dimension-independent model for pointsets with internal structures and incomplete boundary. *Computer-Aided Design*, 1991.

[12] C.K. Yap. Towards exact geometric computation. *Computational Geometry, Theory and Applications*, 7:3–23, 1997.

[13] L. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry: building robust algorithms from imprecise computations. In *5th ACM Symposium on Computional Geometry*, Québec, 1989.

[14] V.J. Milenkovic. Practical methods for set operations on polygons using exact arithmetic. In *Proc. of Canadian Conf. on Computational Geometry*, Québec, 1995.

[15] S. Fortune. Polyhedral modelling with multiprecision integer arithmetic. *Computer-Aided Design*, 29(2):123–133, 1997.

[16] H. Ehrig and B. Mahr. *Fundamentals of algebraic specification 1. Equations and initial semantics*, volume 6 of *EATCS Monograph on Theoretical Computer Science*. Springer-Verlag, 1985.

[17] J.A. Bergstra, J. Heering, and P. Klint. *Algebraic specification*. ACM Press, 1989.

[18] W.R. Mallgren. *Formal specification of interactive graphic programming languages*. ACM Dist. Dissertation. MIT Press, USA, 1982.

[19] N. Dershowitz and J.P. Jouannaud. Rewrite systems. In *Formal models and semantics*, Handbook of Theoretical Computer Science, chapter 6, pages 243–320. Elsevier, 1990.

[20] B. Brüderlin. Using geometric rewrite rules for solving geometric problems symbolically. *Theoretical Computer Science*, 116:291–303, 1993.

[21] P. Lienhardt. Topological models for boundary representation: a comparison with n-dimensional generalized maps. *Computer-Aided Design*, 23(1):59–82, 1991.

[22] W. Tutte. Graph theory. In *Encyclopedia of Mathematics and its Applications*, chapter 21. Cambridge University Press, 1984.

[23] J.F. Dufourd. Algebraic map-based topological kernel for polyhedron modellers: algebraic specification and logic prototyping. In *Proc. of Eurographics*, pages 649–662, 1989.

[24] Y. Bertrand and J.F. Dufourd. Algebraic specification of a 3D-modeler based on hypermaps. *CVGIP : Graphical Models and Image Processing*, 56(1):29–60, 1994.

[25] J.F. Dufourd. Algebras and formal specifications in geometric modeling. *The Visual Computer*, 1997.

[26] M. M¨

antyl¨

a and R. Sulonen. Gwb: a solid modeler with euler operators. *IEEE Computer Graphics & Applications*, 2(7):17–31, 1982.

[27] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoï diagrams. *ACM Trans. on Graphics*, 4(2):74–123, April 1985.

[28] J.A. Goguen, T. Winkler, J. Meseguer, K. Futasugi, and J.P. Jouannaud. *Introducing OBJ*, cambridge university press edition, 1992.

[29] J.F. Dufourd. An OBJ3 functional specification for the boundary representation. In ACM Press, editor, *First ACM-SIGGRAPH Symp. on Solid Modeling*, pages 61–72, Austin, Texas, 1991.

[30] D. Cazier. *Construction de Système de réécriture pour les opérations booléennes en modélisation géométrique*. PhD thesis, Université L. Pasteur, Strasbourg, 1997.

[31] N. Dershowitz and M. Okada. A rational for conditional equational programming. *Theoretical Computer Science*, 75:111–138, 1990.

[32] E. Bevers and J. Lewi. Proving termination of (conditional) rewrite systems. A semantic approach. *Acta Informatica*, 30:537–568, 1993.

[33] M. Wirsing. Algebraic specifications. In *Formal models and semantics*, Handbook of Theoretical Computer Science, chapter 13, pages 675–788. Elsevier, 1990.

[34] K. Mehlhorn. *Multi dimensional searching and computational geometry*, volume 3 of *Data structures and algorithms*. Springer-Verlag, 1984.

[35] D. Cazier and J.F. Dufourd. Reliable boolean operations on polyhedral solids defined as rewrite systems. In *Proc. of WSCG'97*, pages 40–49, Plzen, 1997.

[36] Y. Halbwachs, G. Courrioux, X. Renaud, and P. Repusseau. Topological and geometric characterization of fault networks using 3-dimensional generalized maps. *Mathematical Geologie*, 28(5):625–656, 1996.

[37] D. Cazier and J.F. Dufourd. Term rewrite systems to derive set boolean operations on 2d-objects. In *Proceedings of the 4th Formal Methods Europe Symposium*, pages 605–623, Gratz, 1997. LNCS 1313.