Reliable boolean operations on polyhedral solids thanks to a 3D refinement *

David Cazier and Jean-Franois Dufourd

Laboratoire des Sciences de l'Image, de l'Informatique
et de la Tldtection (LSIIT, URA CNRS 1871)
Universit L. Pasteur, 7, rue Descartes, 67084 Strasbourg
{cazier, dufourd}@dpt-info.u-strasbg.fr

Abstract: *We present a new approach to define reliable boolean operations on polyhedral solids. Firstly, it is based on the refinement of embedded combinatorial 3-maps. Secondly, it uses algebraic specifications and rewriting techniques that leads to a proper formal definition. This approach permits us, in a first time, to concentrate ourselves on the logic of the problem, and then, to consider many control strategies and an efficient implementation.*

Keywords: geometrical modeling, boolean operations, combinatorial maps, algebraic specifications, conditional rewriting.

1 Introduction

Boundary representation (B-rep) and Constructive Solid Geometry (CSG) are the most used models of polyhedral solids. A lot of modelers jointly implement both to offer a wide range of applications. In the two models, boolean operations are an important task either to evaluate the B-rep of a CSG tree, or to compute the union, intersection or difference between solids in B-rep. This question has been tackled many times [1, 2, 3, 4], but problems still arise for a reliable and complete definition and for the robustness of the operations [5, 6]. To solve the first kind of problems, we study a new approach based on the refinement of embedded combinatorial 3-maps and on the use of formal methods. Our paper focuses on boolean operations between solids in B-rep, even if this work applies to general CSG when the primitives of the CSG trees are given in B-rep.

The robustness problems are mainly rooted in approximation and round-off errors due to the floating-point arithmetic used in the calculations. To solve them, Hoffmann and Hopcroft [5] propose to limit the redundancy of numerical data, considering only plane equations, and add to the geometrical tests a symbolic reasoning system insuring against geometrical decisions that are in contradiction with previous ones. Other purely numerical approaches aim to control and limit the round-off errors, for instance by the use of floating-point intervals or rational numbers. Thus, a lazy rational arithmetic proposed by Benouamer *et al.* [6] successfully answers these questions.

On the other hand, serious doubts remain about the exact domain of application of the proposed boolean operations. Precisely, one has to know the kind of objects in

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

input or output and to insure the taken into account of all particular cases. Thus, it seems desirable to accept any data in input, in particular coming from a capture without prior control, but to return as output an object whose total correctness is guaranteed. Our work stands in this framework, where the boolean operations, and in particular our refinement procedure, are defined for checking data and restructuring them in the form of topologically and geometrically sound objects.

To guarantee the best possible rigor, we formally describe the handled objects and the corresponding algorithms with algebraic specifications [7] allied to rewriting [8]. This approach allows us to focus on the conceptual and logical aspects of the problem. We create abstract data types and operations whose behavior is rigorously described as equations or rewrite rules. Moreover, a correct orientation of equations as rewrite rules, allied to a completion [8], permits to execute the specification to flush out possible design errors.

The use of these techniques has been fulfilling in computer graphics languages [9, 10], mechanical proof in geometry [11, 12], geometric modeling [13, 14, 15, 16], and in our works on 2D refinement [17, 18]. Finally, our approach allows to precisely localize the places where the numerical approximations have an important role, in particular because the maps and the employed formalism clearly separate the topological and embedding aspects. Thus, the previous solutions [5, 6] could easily be inserted into our approach.

In Section 2, we describe the expected results and formalize the 3D refinement. In Section 3, we define the embedded combinatorial 3-maps and operations to handle them. In Sections 4 and 5, we go about the problem of 3D refinement and describe this operation as convergent rewrite systems. Finally, in Section 6, we give some conclusions.

2 Definition of the 3D refinement

Historically, the boundary representation of polyhedral solids was limited to 2D-manifolds, for instance objects modeled by a winged-edge structure [19]. Boolean operations between such objects, A and B, take place in four stages [1]: (i) *Intersection:* to research existing intersections between A and B; (ii) *Classification:* to class as *in* or *out* the parts of A depending on whether they belong to the interior or to the exterior of B, and conversely for the parts of B with respect to A; (iii) *Cutting-out:* to select and to cut the parts of A and B that will belong to the result (for instance to compute $A \cap B$, the selected parts are $AinB$ and $BinA$); (iv) *Topological restructuring:* to assemble the selected parts what build the expected result.

As well as they are limited to 2D-manifolds, these methods are not reliable. Indeed, as shown by Hoffmann and Hopcroft [5], numerical approximations, in particular during the cutting-out stage, may conduct to incompatible boundaries for the parts of A and B to be sewn and thus prevent a correct restructuring.

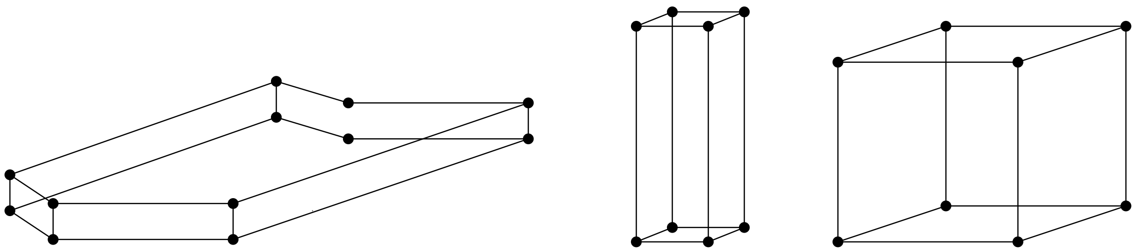
Another method introduced by Weiler [20] for the 2D, and extended in 3D [3], takes place in three stages, the first of which mainly consisting in the *corefinement* of the edges, faces and volumes of A and B, i.e. in the cutting of these cells in smaller ones, compatible with the two objects: (i) *Corefinement:* to cut secant cells of A and B and to restructure their adjacency relationships; (ii) *Classification:* to classify as *in* or *out* the parts of the corefinement depending on whether they respectively belong to the interior and/or the exterior of A and B; (iii) *Selection:* to select the parts of the corefinement of the result.

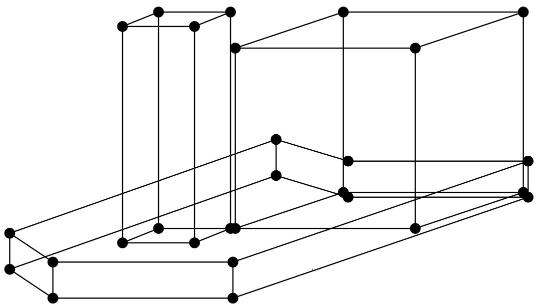
This method requires a model of objects whose boundaries are not 2D-manifolds, and in particular, that may contain internal structures. It leads to use models of 3D-space subdivisions or of 3D-manifolds as for instance the radial-edge structure [21], the facet-edge structure [22] or the editable nonmanifold boundary representation of [3]. The SGC model (Selective Geometric Complexes) introduced by Rossignac, based on the notion of cells, is used with a similar kind of corefinement whose framework is given in [23].

The 3D corefinement has the advantage to lead to algorithms that control the topology of the handled objects, what make them less sensitive to errors of approximations. Indeed, two secant cells of A and B are cut in the same time, what allows to control and to conserve the validity of the whole topology. Thus, our interest lies in the refinement of 3D objects modeled by *embedded* 3-*maps* that have been proven equivalent to the radial-edge structure [24, 13, 14].

Let us explain the notion of corefinement for two 3D objects considered as 3D space subdivisions in volumes, faces, edges and vertices. It consists in producing a new subdivision that contains the parts of the two first ones that are subdivised to take into account the intersections and superpositions that may exist between them. In fact, superposed vertices, edges or faces are merged, secant edges or faces are cut along their intersections that are added to the resulting subdivision as new vertices or edges, and finally, the edges and faces that overlap some vertices are cut at these incidence points. Thus, the corefinement no more contains intersections, incidences or superpositions. The result of the boolean operations is then obtained from the corefinement by a selection of the required parts.

The corefinement is generalized to any number of subdivisions grouped in a set of vertices, edges, faces and volumes that contains intersections or overlapings. The 3D *refinement* of this set transforms it into a set that correctly models a space subdivision. In Fig 1, three subdivisions (a), are superposed to form such a set (b) that is refined (c).


(a) Three subdivisions


(b) Their superposition as a set of vertices, edges and faces

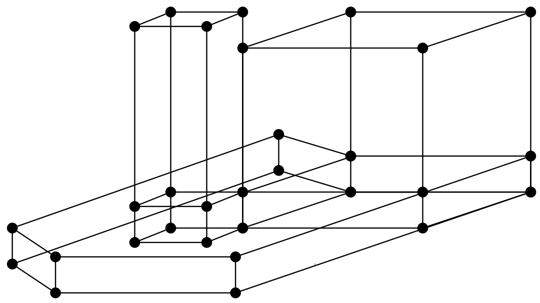

(c) The refinement of this set

Figure 1: *An example of 3D refinement*

In addition to its generality, such a definition of refinement helps to easily edit the computed boolean operations. Indeed, let us consider a CSG tree, with primitives at its leaves and boolean operations at its nodes. The 3D refinement of this tree is computed once. Then all CSG trees with the same primitives can be evaluated by a single selection in the refinement, what makes possible an interactive edition of the trees [3].

To control the influence of approximations, it is worth to distinguish topology and embedding. Topology defines the parts of objects, i.e. their vertices, edges, faces and volumes, as well as their incidence and adjacency relationships. Embedding defines position and shape of these parts, the more often point coordinates for vertices, Jordan arcs for edges and surface patches for faces. The two aspects exist in embedded combinatorial 3-maps that allows an easy, precise and concise description of volumic subdivisions. A comparison with other known models, like the radial-edge structure [21], is proposed in [24, 14].

3 Embedded combinatorial 3-maps

3.1 Combinatorial 3-maps

We recall here some basic notions on map (see [14, 15] for more details and a deep formalization). A 3-*map* is a quadruple $M = (B, \alpha_0, \alpha_1, \alpha_2)$ where B is a set, element of which are called *darts*, α_0 and α_1 are involutions on B, i.e. permutations such that for all x we have $\alpha_0(\alpha_0(x)) = x$ and $\alpha_1(\alpha_1(x)) = x$, and finally α_2 is permutation on B such that $\alpha_2 \circ \alpha_0$ is an involution.

Two darts are said *linked* by α_0 (resp. α_1), or 0-*sewn* (resp. 1-*sewn*), when they are mapped on each other by α_0 (resp. α_1). Darts are generally viewed as half-edges. Thus two 0-sewn darts make a *simple edge* and a link by α_1 corresponds to the 1-sewing of two edges at a *simple vertex*. A sequence of darts sewn by α_0 and α_1 forms a face. Fig. 2 shows the conventions used to represent darts and links by α_0 and α_1. In the following, we consider that all faces are closed, i.e. α_1 is without fixed point.

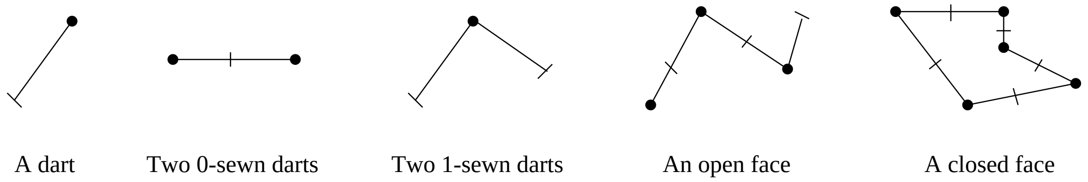


Figure 2: *Conventions and graphical representations*

A link by α_2 corresponds to the 2-sewing of two faces along simple edges. The fact that $\alpha_2 \circ \alpha_0$ be an involution means that the 2-sewings are made on whole simple edges. Faces are 2-sewn to each other to form volumes. Notice that more than two faces may be 2-sewn together. All the simple edges (or vertices) whose darts are 2-sewn, form a multiple edge (or vertex) of the 3-map.

Example 1 *Fig. 3 shows a 3-map (a), its faces (b) and the detail of the 2-sewings in a multiple edge (c). For the numbered darts the permutations are, in cyclic notation:* α_0 *= (-1,1) (-2,2) (-3,3) (-4,4) (-5,5) (-6,6),* α_1 *= (-3,-5) (3,6) (4,-6) (-4,5) ..., and* α_2 *= (-4) (4) (-5) (5) (-6) (6) (-1,-3,-2) (1,2,3). So we have :* α_0*(1) = -1 and* α_0*(-1) =1, i.e.*

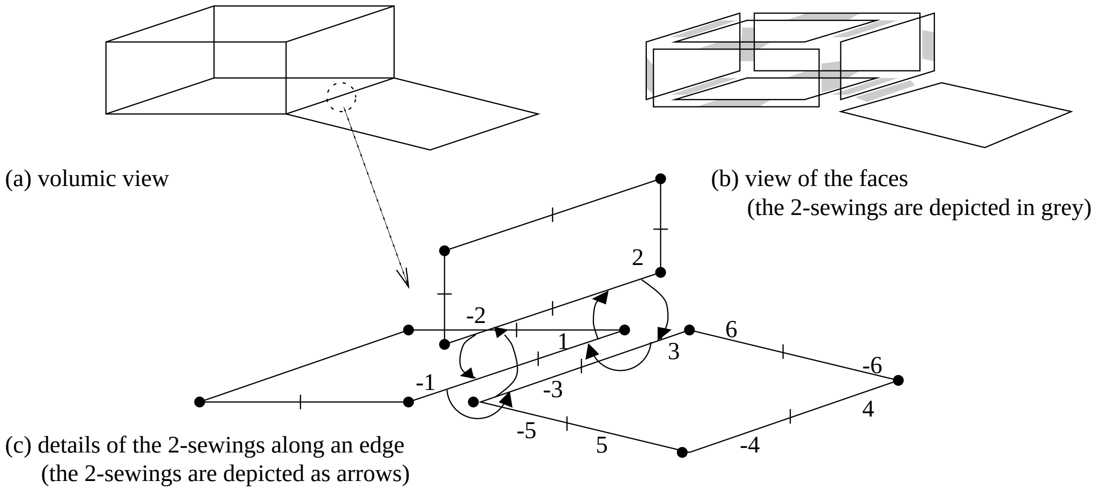


Figure 3: *A map and details of a 2-sewing*

{1,-1} is a simple edge; α_1*(-3) = -5 and* α_1 *(-5) = -3, i.e. {-3,-5} is a simple vertex;* α_2*(-4) = -4, i.e. the simple edge {4,-4} is not 2-sewn;* α_2*(1) = 2,* α_2*(2) = 3 and* α_2*(3) = 1, i.e. the faces that contain darts 1, 2 and 3 are 2-sewn. Finally, {1,2,3,-1,-2,-3} forms a triple edge.* □

3.2 Embedding

The geometry of objects is defined by the *embedding* of combinatorial maps. It consists in associating to each topological element of a map a geometrical object of the same dimension. In this work, we associate to each vertex a point (0-embedding), to each edge the segment joining its two embedded vertices (1-embedding) and to each face the polygon formed by its embedded edges and vertices. Note that an embedded 3-map may contain dangling faces. Although this notion can be formally defined, we simply say that a 3-map is *well embedded* if it has an embedding without self-intersection or self-overlapping.

3.3 Formal specifications of embedded 3-maps

To formalize the notion of subdivision and to avoid the desentangling problems due to concrete data structures and special features of programming languages, maps and their operations are defined through an algebraic specification [7]. This one describes the behavior of a set of constructors and observators of maps through a first order equational theory. To cut short, we only informally explain the effects of some of them. Details on algebraic specifications of maps and on basic operations may be found in [13, 14, 15, 16].

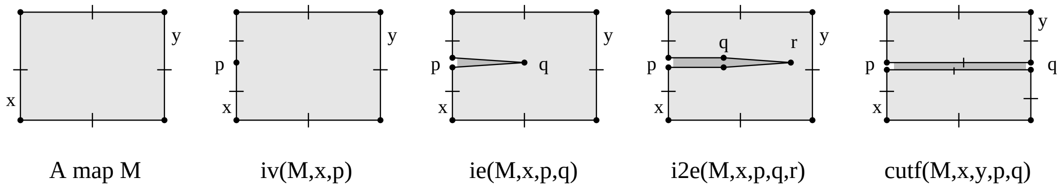


Figure 4: *Topological operators on 3-maps*

Fig. 4 depicts a map M and the action of four operators we use in the following. The operation $iv(M, x, p)$ inserts 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. Note that it can be formally proved that these operations preserve all the characteristic properties of the 3-maps.

4 The refinement of maps

We define the refinement of embedded maps through a set of elementary and independent operations nicely described by the rules of a *conditional modulo rewrite system* [25, 26]. We use for the rules the syntax :

$$R_{example} : \frac{M}{M'} \textbf{ if } \begin{cases} condition_1 \\ \ldots \\ condition_n \end{cases}$$

The numerator M represents a map and the denominator M' represents it after a rewrite step. Rules can apply only when all the conditions are satisfied. Table 1 contains the formal rule descriptions, the first two being graphically depicted in Fig. 5.

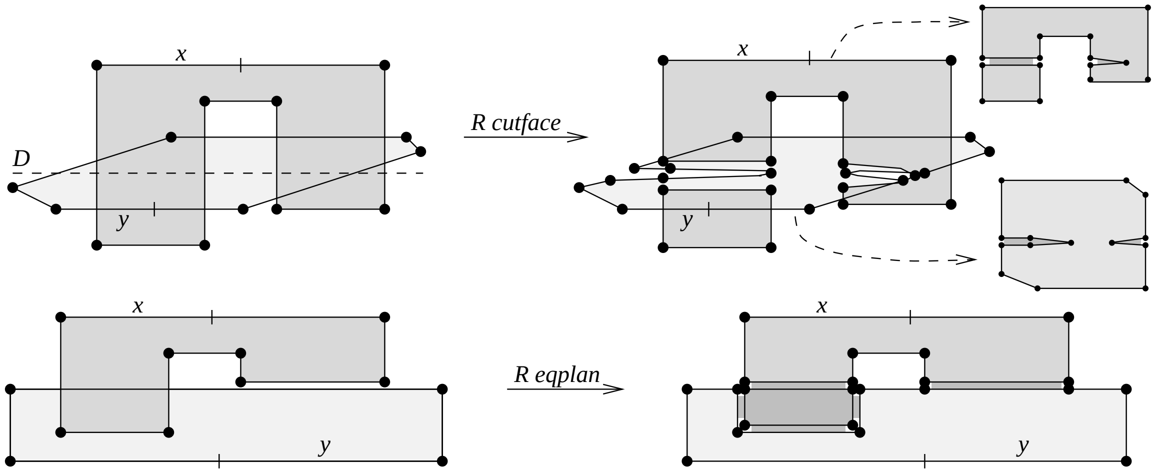


Figure 5: *A graphical illustration of rewrite rules*

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 (separately depicted on the top-left of figure 5) are cut along the intersection line. This operation is detailed in the next section. The second rule, $R_{eqplane}$, does the same when the two faces are coplanar. This operation amounts to intersect two polygons in a plane. It has been designed by a rewrite subsystem, as for $R_{cutface}$ in section 5 and transformed into a line-sweep algorithm in the way of [20, 27]. Details can be found in our papers on the 2D refinement [17, 18].

The third rule, R_{eqface}, is usable 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-sewn to the deleted face are then 2-sewn to the other face, by R_{eqedge}.

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

Table 1: *Rewrite system for the refinement of maps*

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

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

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

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

The refinement of a given map amounts to try 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 $O(k + n.ln^2 n)$, where n is the total number of boxes [28].

5 Intersection of non coplanar faces

The more delicate part of the refinement is the intersection of faces. Our way to go about this question looks like the one of [6]. But we benefit from a precise topological structure, the one of 3-maps, and from a set of algebraically specified operators that allow us to simplify and rigorously control the definition of this process. When two non coplanar faces intersect along a line D, their intersection amounts to sort segments along D. Each face is intersected with the plane of the other one, what supplies a set of segments of D that are in the interior of the face or on its boundary.

Two faces are secant if two segments of their respective intersection with the plane of the other face overlay each others. A sweep of D is used to detect the couples of segments that overlay. The configuration of these segments is then used to cut the two faces. A formal description of the system, only based on the study of configurations has been done. We present here a system improved by a vertical refinement (in the software engineering sense) and that takes into account the strategy of D sweeping.

The intersection, along a line D equipped with a coordinate system $(O, \vec{\imath})$, of a face f and a plane P is obtained from the list of classifications of edges of f with respect to P. To each edge e being in contact with P, a quadruple (x, p, k, θ) is added to the list, where x is a dart of e, p is the point of contact of e with P, k is the abscissa of p in the coordinate system $(O, \vec{\imath})$ and θ the angle of e with respect to $\vec{\imath}$. If e is included in P, the point of contact is the vertex of smaller abscissa of e. Finally, if e cuts P, p is the point of intersection of e with P.

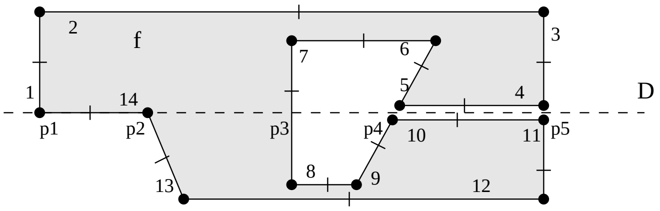


Figure 6: *An example of intersection of a face with a plane*

The resulting list is then sorted by increasing abscisses k and angle θ and transformed in a *state list*, i.e. a list of triplets $[(x, px), (x', px'), state]$. In this notation, x and x' are two darts of f whose edges are in contact with P at points px and px' and such that the segment $[px, px']$ of D is completely in the interior of f ($state = in$) or on the boundary of f ($state = on$). Segments that are in the exterior of f are suppressed from this list during its computation. For the intersection in Fig. 6 the result is the list :

$$[(1, p1), (14, p2), on]; [(14, p2), (7, p3), in]; [(5, p4), (11, p5), on]$$

The intersection of f and g is carried out by an ordered merging of their two state lists. If these lists have for first element $[(x, px), (x', px'), state_x]$ and $[(y, py), (y', py'), state_y]$ respectively, the topological operations that have to be done to cut f and g are obtained from a case by case study of the respective position of px, px', py and py' on D.

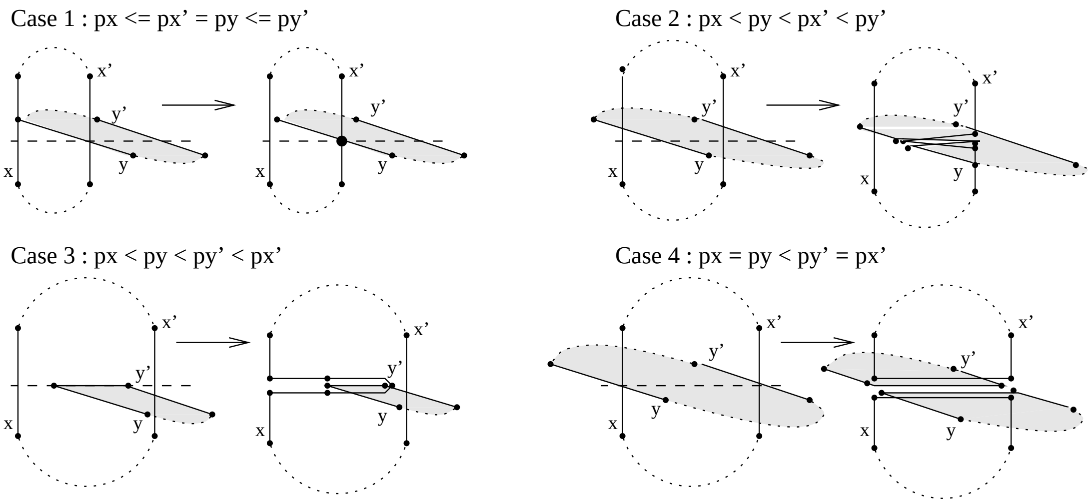


Figure 7: *A representation of four cases of faces intersection*

To simplify the notations, we write in the following $px = py$, or $px < py$, to express that, in the coordinate system $(O, \vec{\imath})$ of the line D, the abscissa of point px equals the py one, or is strictly lower than the py one. Notice that the equality may be strict or apart from a threshold ε depending on the kind of used geometry, but in all cases $px < py \Rightarrow \neg(px = py)$ to avoid conflicts between these relations.

To formally describe the different cases and the corresponding operations, we use a rewrite system that takes in input the map M and the two state lists. Thus, the initialization of the $cutface(M,x,y)$ operation is done by the rule :

$$R_{init} : \frac{cutface(M,x,y)}{M \ ; \ state\_list(M,x) \ ; \ state\_list(M,y)} \textbf{ if } true$$

This notation means that map M and the two state lists are, in a way, given as parameters to the system that carries out the intersection of the faces of x and y. This last system is detailed in Table 2, where we note by $e|L$ the list whose head is e and whose queue is L. The principal cases are depicted in Fig. 7.

Table 2: *Rewrite system for faces intersection*

$$R_{fin} : \frac{M \ ; \ L_x \ ; \ L_y}{M} \textbf{ if } L_x = empty\_list \vee L_y = empty\_list$$

$$R_{symmetry1} : \frac{M \ ; \ [(x,px),(x',px'),state_x]|L_x \ ; \ [(y,py),(y',py'),state_y]|L_y}{M \ ; \ [(y,py),(y',py'),state_y]|L_y \ ; \ [(x,px),(x',px'),state_x]|L_x} \textbf{ if } py < px$$

$$R_{symmetry2} : \frac{M \ ; \ [(x,px),(x',px'),state_x]|L_x \ ; \ [(y,py),(y',py'),state_y]|L_y}{M \ ; \ [(y,py),(y',py'),state_y]|L_y \ ; \ [(x,px),(x',px'),state_x]|L_x} \textbf{ if } py = px \wedge py' < px'$$

$$R_{null} : \frac{M \ ; \ [(x,px),(x',px'),state_x]|L_x \ ; \ [(y,py),(y',py'),state_y]|L_y}{M \ ; \ L_x \ ; \ [(y,py),(y',py'),state_y]|L_y} \textbf{ if } px \leq px' < py \leq py'$$

$$R_{case1} : \frac{M \ ; \ [(x,px),(x',px'),in]|L_x \ ; \ [(y,py),(y',py'),in]|L_y}{iv(iv(M,x',px'),y,py) \ ; \ L_x \ ; \ [(y,py),(y',py'),in]|L_y} \textbf{ if } px \leq px' = py \leq py'$$

$$R_{case2} : \frac{M \ ; \ [(x,px),(x',px'),in]|L_x \ ; \ [(y,py),(y',py'),in]|L_y}{ie(ie(M,x',px',py),y,py,px') \ ; \ L_x \ ; \ [(y,py),(y',py'),in]|L_y} \textbf{ if } px < py < px' < py'$$

$$R_{case3} : \frac{M \ ; \ [(x,px),(x',px'),in]|L_x \ ; \ [(y,py),(y',py'),on]|L_y}{cutf(i2e(M,x,px,py,py'),y,y',py,py') \ ; \ L_x \ ; \ [(y,py),(y',py'),on]|L_y} \textbf{ if } px < py < py' < px'$$

$$R_{case4} : \frac{M \ ; \ [(x,px),(x',px'),in]|L_x \ ; \ [(y,py),(y',py'),in]|L_y}{cutf(i2e(M,x,px,py,py'),y,y',py,py') \ ; \ L_x \ ; \ L_y} \textbf{ if } px = py < px' = py'$$

The intersection of the two faces is completed when one of the two state lists is empty. Rule R_{fin} detects it and returns to the first rewrite system the map M in which the intersection has been done. Rules $R_{symmetry1}$ and $R_{symmetry2}$ exchange the two state lists to divide the number of cases thanks to symmetries. In the other rules, we always have $(px < py)$ or $(px = py \wedge px' \leq py')$.

Rule R_{null} goes to the next element in the first list if no overlapping occurs between the two current segments. Rule R_{case1} inserts a new vertex in the edges of x' and y cutting D at point px' $(= py)$. The system goes on with the next segment of the first list.

In the case of R_{case2}, segment $[py, px']$ lies in the interior of the two faces. A new edge is thus inserted in the two faces whose vertices are respectively embedded on points px' and py. This is done by the ie operator.

For rule R_{case3}, $[py, py']$ lies in the interior of the face of x. As this segment is already in the boundary of the face of y, nothing more is done for this face. On the other hand, it is in the interior of the face of x, but is not incident to an edge of this face. So, two double edges are inserted by the $i2e(M, x, px, py, py')$ operator.

In the last case, R_{case4}, the segment $[px, px']$, equal to segment $[py, py']$, separates the two faces. Two double edges are thus inserted respectively between x and x', and between y and y', to locally separate out the two faces. The last three cases are not detailed here, but have the same shape and use the same operators to insert edges or to cut faces. They correspond to the other possibilities of sorting the points px, px', py and py'.

Each of these 7 cases is subdivided in 4 cases for each value of $state_x$ and $state_y$. In fact, if $state_x = in$, the described operations apply. But, if $state_x = on$, as in case 3, then the edge of x is on D, the insertion of vertices or edges is not necessary and nothing is done. This also holds for y. Altogether the sub-system contains 32 rules $(4 + 4 * 7)$.

Intuitively, this process *terminates* because the sum of the two list lengths decreases at each rewriting step. Finally, it is easy to prove that each rule application on a 3-map returns a 3-map, what ensures the *correctness* of the intersection process. The *confluence* of the rewrite system [8, 26] is ensured by the fact that the conditions of the rules are separated, i.e. two different rules cannot be used in the same time.

6 Conclusion

We have defined topological and geometrical operations to build and handle subdivisions of volumes, with the mathematical model of combinatorial maps. The joint use of algebraic specifications and rewriting conducts us to express a complex problem as elementary and independent transformations. We have thus completely and formally described the refinement of an embedded 3-map. Beyond the corefinement of two volumic subdivisions, this operation is in fact a general process of normalization of volumic subdivisions.

Operations on maps and rewrite rules have been first implemented in 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 of specifications allied to rewriting expressiveness makes easier a proper and rigorous design of abstract algorithms. It is also a nice starting point for a further design of concrete and efficient algorithms.

A prototype has thus been written in C for a Sun workstation and the X11 library, where different kinds of strategies were studied for an efficient execution of the rewrite system. Firstly, we have done a straightforward implementation of the simple algorithm whose complexity is in $O(n^2)$. This has been made easier thanks to the logical prototyping. Next, classical methods, such as the bounding boxes one recalled before, have been implemented to improve the complexity until achieving the best known result. Here again the clear separation of logic and control have helped us.

Questions of approximation we have not deal with do not cast doubt on the structure of the rewrite system, because they are precisely localized. For instance, for the non coplanar faces intersection they only take place in the state lists building and in the evaluation of rules conditions. So, 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, what extends the *active* attribute of the cells of SGC [23] and is similar to the *historic* attribute of [3]. The rewrite system will be completed to take into account this attribute during the refinement stage. Our goal is to directly obtain, after refinement, the result of a classical boolean operation between many objects modeled in a 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] 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.

[2] D. Badouel and G. Hgron. An evaluation of CSG trees based on polyhedral solids. In *Proc. Eurographics*, Nice, 1988.

[3] G.A. Croker and W.F. Reinke. An editable nonmanifold boundary representation. *IEEE CG & App.*, pages 39–51, 1991.

[4] Y. Gardan and E. Perrin. An algorithm reducing 3D boolean operations to a 2D problem: concepts and results. *Computer-Aided Design*, 28(4):277–287, 1996.

[5] 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.

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

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

[8] 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.

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

[10] D.A. Duce and F. Paterno. Lotos description of GKS-R functionality. In Springer-Verlag, editor, *Eurographics workshop on formal methods in computer graphics*, Marina di Carrara, Italy, juin 1991.

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

[12] P. Balbiani, V. Dugat, L. Fariñas del Cerro, and A. Lopez. *Elments de gomtrie mcanique*. Herms, 1994.

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

[14] 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.

[15] J.F. Dufourd. Formal specification of topological subdivisions using hypermaps. *Computer-Aided Design*, 23(2):99–116, 1991.

[16] Y. Bertrand, J.F. Dufourd, J.F. Françon, and P. Lienhardt. Algebraic specification and development in geometric modeling. In *LNCS*, EATCS conf. TAPSOFT, pages 75–89, Orsay, France, 1993. Springer-Verlag.

[17] D. Cazier and J.F. Dufourd. A rewrite system to build planar subdivisions. In *Proc. Canadian Conf. on Computational Geometry*, pages 235–240, Qubec, 1995.

[18] D. Cazier and J.F. Dufourd. Rewrite-based derivation of efficient algorithms to build planar subdivisions. In *Proc. Spring Conf. on Comp. Graphics*, pages 45–54, 1996.

[19] B.G. Baumgart. A polyhedron representation for computer vision. *National Computer Conference*, 44:589–596, 1975.

[20] K.J. Weiler. Polygon comparison using a graph representation. In *Proc. of ACM Siggraph Conf., Computer Graphics*, volume 14, pages 10–19, 1980.

[21] K.J. Weiler. The radial edge structure : a topological representation for non-manifold geometric modeling. In *Geometric Modeling for CAD Applications*, pages 37–68. Springer Verlag, 1986.

[22] D.P. Dobkin and M.J. Laszlo. Primitives for the manipulation of three-dimensional subdivisions. *Proc. of 3rd ACM Symp. on Computional Geometry*, pages 86–99, 1987.

[23] 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.

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

[25] J.A. Bergstra and J.W. Klop. Conditional rewrite rules: confluence and termination. *JCSS*, 32:323–362, 1986.

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

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

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