Proceedings of the TMCE 2006, April 18–22, 2006, Ljubljana, Slovenia, Edited by I. Horváth and J. Duhovnik
©2006 Millpress, Rotterdam, ISBN

REFLECTIONS ON THE IMPLEMENTATION OF BOOLEAN OPERATIONS WITH POLYHEDRAL SOLIDS

Herbert J. Koelman
SARC BV, Bussum
H.J.Koelman@sarc.nl

ABSTRACT

A practical CAD application may significantly benefit from the facility to perform Boolean operations with the modelled objects. Fortunately, the literature provides a plurality of discussions and methods for such operations with polyhedral solids and sculptured solids, while also attention is paid to the possible pitfalls of numerical incompatibilities. So the implementation of a Boolean facility seems to be a matter of implementation rather than a matter of research. However, in the real world, situations may occur for which the published methods offer no solution. To some extent this can be related to numerical issues, but partly it is fundamental, especially with solids that do not really intersect, but only touch each other. This paper identifies and illustrates these problems, and formulates suggestions for modifications to the solution strategies. Furthermore, solutions for the numerical incompatibilities are proposed, and application examples are presented. Finally, it is concluded that our proposed methods provide no fundamental solution, they are practical workarounds instead.

KEYWORDS

Boundary Representation, Boolean Operation, Polyhedral solid, Solid Modelling.

1. INTRODUCTION

The activities of our company are located in the field of ship design and naval architecture. For that purpose we have been developing a dedicated set of computer programs, which are marketed under the names `PIAS` and `Fairway`. It will be obvious that such design software relies heavily on the proper modelling of the geometric model of the ship, both the exterior and the interior. In agreement with the naval architectural tradition, initially the chosen hull shape modelling method was based on cross-sectional modelling. In a later stage of program development this sectional model was replaced by an extended B-rep model, see (Koelman et al., 2001). However, the most labor-intensive and error-prone activity in the initial design stage is not the hull modelling activity, but the modelling of the interior, such as holds, tanks and other spaces. In order to enable quick modelling our system offered the possibility to specify only the coordinates of the eight extreme corners of a box, whereafter the program calculated the intersection between that box and the cross sections of the hull. Because in reality internal spaces can be more complex than box-shaped ones, multiple boxes can be merged into compartments. In this respect the boxes are only logical entities, the compartment is a physical one. An example of such a sectional-based compartment model is presented in Fig. 1. However, now that a solid model of the hull is available, it is also possible to model the interior in a more complete way. This could be done with surfaces, as employed by most naval architectural programs, but since we have a solid model of the hull available, an implementation on the basis of solids may also be feasible. Such an approach, which is also applied in (Lee et al., 2003), offers the following advantages:

- With the conventional methods it is the responsibility of the program user that the boxes do not overlap. With the proposed method the user will be allowed to model the boxes in a much more relaxed fashion; the system will deal with the overlapping parts.

- Enhanced representation accuracy, especially in the curved forward and aft regions of the vessel. Besides the fact that a more complete representation will be an aid for the program user in the judgement of the correctness of the model, it may also enhance the accuracy of the calculation. On the other hand, a lack of accuracy was not a specific defect of the conventional method; with the use of second-order integration schemes, a relatively high accuracy could always be combined with a modest section density.

- An implementation with solid models is to be preferred above a surface representation. With the latter it is the task of the user to specify explicitly which surfaces must be considered as boundaries of a space, with solids this knowledge is an intrinsic part of the representation.

An example of a compartment represented by a solid which is composed as the union of elementary boxes, is given in Fig. 2.

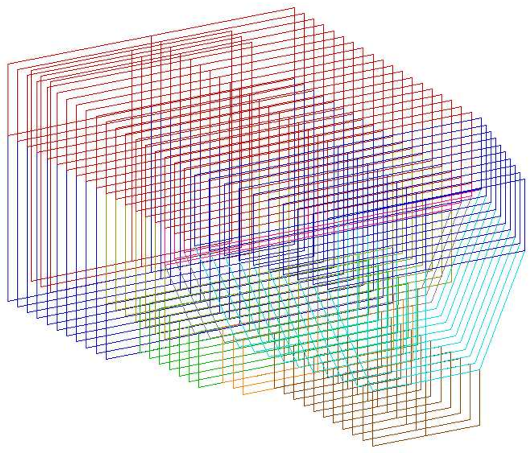

Figure 1 Sectional representation of cargo hold

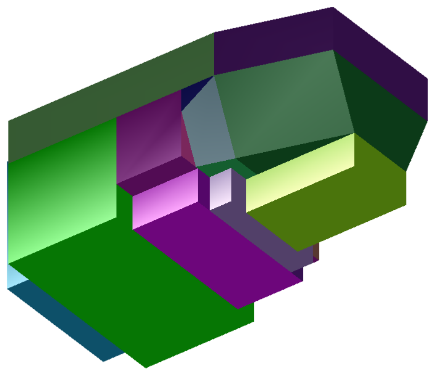

Figure 2 Solid representation of the cargo hold of Fig. 1

2. A SURVEY OF METHODS FOR BOOLEAN OPERATIONS

Boolean operations on solids are widely discussed in literature. For the author it is not certain whether the plethora of publications on this subject is a sign of matureness, or an indication that the field is still under development. Anyway, for our intended task we need to survey the literature, although the available space forces us to briefness. Furthermore we have not taken standard available solid modelling kernels, such as ACIS or Parasolid, into account. Although the use of kernels may offer a solution with respect to the implementation, they can only be used as black boxes and offer no insight in the applied methodology. With the focus on polyhedral solids also CSG-based methods have not been considered.

2.1. Boolean operations on polyhedral solids

In (Requicha, 1980) the observation was made that the conventional Boolean set operations $\cap$, $\cup$ and - are not algebraically closed. However, the regularized intersection union and difference (denoted $\cap^*$, $\cup^*$ and $-^*$) are closed. In (Requicha & Voelcker, 1984) the regularized Boolean set operations are further elaborated and applied on polyhedral solids. A concise discussion of the regularized Boolean set operations is given in e.g. (Foley et al., 1990). In (Chiyokura, 1988) the solid modelling system `DESIGNBASE` is presented, including a detailed discussion of its polyhedral Boolean set capacities. With `DESIGNBASE` only the $\cup^*$ operation is implemented, with the steps a) generating intersection lines, b) generating intersection points, c) removing the included part of one solid and d) joining the two solids. The $\cap^*$ and $-^*$ operations are related to the $\cup^*$ as follows:

$$A \cap^* B \;=\; \neg(\neg A \cup^* \neg B) \qquad (1)$$
$$A -^* B \;=\; \neg(\neg A \cup^* B) \qquad (2)$$

(where $\neg$ denotes the negation)

In (Mäntylä, 1988) the modeller `GWB` ((Mäntylä & Sulonen, 1982)) is described, its Boolean set methods are elaborated in (Mäntylä, 1986). `GWB` is founded upon a polyhedral Boundary Representation (B-rep). Given two solids A and B, with bounding polyhedra $\mathcal{A}$ and $\mathcal{B}$, four objects can be distinguished, viz. $\mathcal{A}$inB (the part of $\mathcal{A}$ within B), $\mathcal{A}$outB, $\mathcal{B}$inA and $\mathcal{B}$outA. These objects are sufficient for conventional Boolean operations, but for *regularized* operations four more components must be classified, viz. $\mathcal{A}$on$\mathcal{B}^+$ (the part of $\mathcal{A}$ which lies on $\mathcal{B}$ with coinciding face normals), $\mathcal{A}$on$\mathcal{B}^-$ (with opposite face normals), $\mathcal{B}$on$\mathcal{A}^+$ and $\mathcal{B}$on$\mathcal{A}^-$. The regularized Boolean combinations are computed by

$$A \cup^* B = \mathcal{A}out\mathcal{B} + \mathcal{B}out\mathcal{A} + \mathcal{A}on\mathcal{B}^+ \quad (3)$$
$$A \cap^* B = \mathcal{A}in\mathcal{B} + \mathcal{B}in\mathcal{A} + \mathcal{A}on\mathcal{B}^+ \quad (4)$$
$$A -^* B = \mathcal{A}out\mathcal{B} + \neg\mathcal{B}in\mathcal{A} + \mathcal{A}on\mathcal{B}^- \quad (5)$$

The set operation algorithm comprises four steps, namely a) generate new vertices on the intersections of $\mathcal{A}$ and $\mathcal{B}$, b) perform the described 8-way classification of $\mathcal{A}$ and $\mathcal{B}$, c) connect the newly generated vertices and d) create the result with Eqs. 3 to 5.
In (Hoffmann et al., 1987) and (Hoffmann, 1989) a method is described which is more or less comparable to DESIGNBASE and GWB. The consistency and robustness of this method is enhanced by a particular sequence of operations, which avoids that the same geometrical operation is performed more than once.
In practice it appeared that the Boolean set operations are very critical to numerical round-off effects. In order to avoid these pitfalls, interval arithmetic can be applied (see e.g. (Hu et al., 1996) or (Tsuzuki & Shimada, 2003)). Other variant methods have also been published, e.g. in (Segal & Sequin, 1988), where the objects A and B are processed without intermediate classifications, and in (Gardan & Perrin, 1996), where the 3D geometric intersection algorithm is reduced to a 2D one.

2.2. Boolean operations on sculptured solids

When a non-polyhedral modelling method is applied, the Boolean set operations must also be capable for objects with a sculptured shell. The most obvious approach is to approximate the sculptured surface by a polyhedron, and then apply a conventional polyhedral method (see e.g. (Toriya et al., 1991)). A more accurate result might be obtained by direct processing or the curved surface representation of the shell. In (Krishnan et al., 2001) such a system, BOOLE, is described, with some impressive examples. Additionally, in (Keyser et al., 2002) a descendant of BOOLE is presented, where thanks to enhanced precision some failures of BOOLE are avoided.

3. CHOICES, AND CONDITIONS FOR IMPLEMENTATION

For the implementation of Boolean set operations in our naval architectural design system the following conditions have been formulated:

1. Because the average naval architect is not acquainted with detailed aspects of topology or Boolean arithmetic, the mathematical backgrounds and peculiarities must be hidden for the user.

2. For practical reasons the adopted method must as much as possible be compatible with the classes and methods of our design system. Because our system is based on an extended B-rep, and supports sculptured faces, the Boolean set operations must be suitable for, or at least expandable to, non-polyhedral solids.

3. Especially the operations on touching objects must perform well. The reason is that the Boolean functions are heavily applied for the creation of compartments, which are composed from user-defined boxes (called sub-compartments in our context). And the user has two motives to define the sub-compartments merely touching instead of overlapping. The first reason is historical, the users are trained to do so because our previous wireframe modelling system allowed no overlapping sub-compartments. Secondly, if the number of sub-compartments rises, it appears to be easier for an average human to imagine a collection of 'nicely' touching building blocks, than one where they wildly overlap.

4. General requirements are robustness and processing speed. However, concerning the latter aspect, we must realize that in our domain of application very high numbers of solids are not expected. A typical internal layout of a ship may contain several hundreds of internal solids, which is modest compared to the more than 5000 solids of the Bradley Fighting Vehicle, as reported in (Keyser et al., 2002). So, in the first development stage we have considered performance not of the *prime* importance.

With the aim of step-wise development, we have chosen a three-stage approach: a) a conventional polyhedral method, b) refinement for robustness and efficiency and c) extension to pseudo-sculptured objects, by means of polyhedral approximation. At first sight there appears to be little fundamental difference between the various methods from subsection 2.1. Our choice for the first development stage fell on the method of (Mäntylä, 1988), because a lot of the employed basic classes and methods were already available in our software.

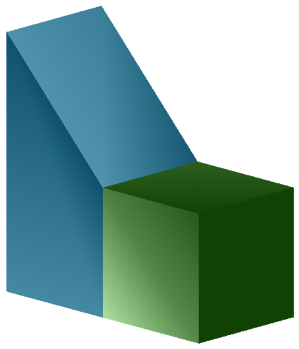

Figure 3 Union of two objects

4. EXPERIENCES AND MODIFICATIONS

The initial implementation performed well on solids which either intersect properly, or do not intersect at all. The next test was a union operation with a cube and another hexahedron which only touch, see Fig. 3. This simple case failed. For convenience we have left our sources resting for the time being, and experimented a bit with a similar Java implementation which is available on Internet, (Bhardwaj & Malik, 1997). This implementation succeeded in the case of Fig. 3, but it failed in some other cases, for example with the object of Fig. 4a. Once this object is intersected with a box with a touching lower plane and an upper plane which cuts off the objects top, a superfluous intersection plane is created in the side, which is shown as the vertical wedge in Fig. b. So there was a problem, we had two different implementations of, more or less, the same method and both failed, on different points. Obviously the time for superficial observations was over, and a more in-depth study was required. The different subjects that have been brought forward in this process will be discussed in the next sub-sections.
Besides, a number of minor modifications have been made. Because an implementor may benefit from our efforts, they are included in the appendix as tips.

4.1. Reclassification of edges

The major steps of the algorithm have been summarized in Sub-section 2.1. One of those steps is the classification of edges and vertices, that means to detect for each of these elements whether they belong to the part $\mathcal{A}$in$B$, $\mathcal{A}$out$B$, $\mathcal{B}$in$A$ or $\mathcal{B}$out$A$. Time and again problems can be traced back to this reclassification step. It can be derived that the reclassification scheme should be as in Table 1.

In the reclassification process three cases can be distinguished:

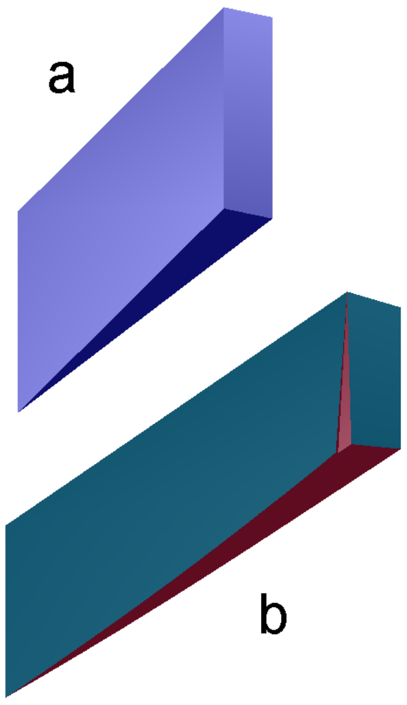

Figure 4 A solid (a) and its intersection with a larger solid (b)

Operation	$\mathcal{A}$on$\mathcal{B}^+$	$\mathcal{A}$on$\mathcal{B}^-$	$\mathcal{B}$on$\mathcal{A}^+$	$\mathcal{B}$on$\mathcal{A}^-$
$\cup^*$	$\mathcal{A}$out$B$	$\mathcal{A}$in$B$	$\mathcal{B}$in$A$	$\mathcal{B}$in$A$
$\cap^*$	$\mathcal{A}$in$B$	$\mathcal{A}$out$B$	$\mathcal{B}$out$A$	$\mathcal{B}$out$A$
$-^*$	$\mathcal{A}$in$B$	$\mathcal{A}$out$B$	$\mathcal{B}$out$A$	$\mathcal{B}$out$A$

Table 1 Standard reclassification table

- Sector-sector coincidence, that means that when a sector of one solid coincides with a sector of the other solid (where a sector is defined as the part of a face between two edges in the neighbourhood of their common vertex). This case is covered by Table 1.

- Edge-sector coincidence. In this case an edge (E) of one solid coincides with a sector (S) of the other solid, but not with an edge of the other solid. This case is rather easily covered by detecting if the two sectors which are located on both sides of E are situated on different sides of S. If that is the case the solids intersect at E, otherwise they don't.

- Edge-edge coincidence. In this case the presence of an intersection at the coinciding edges is determined by an analysis of the orientation of the sectors around the coinciding edges. If the sectors appear in a mixed order, than the two solids intersect at the edges, otherwise they do not intersect, but only touch each other. A complicating possibility is that two involved *sectors*, each from a different solid, may also coincide. In that case the order cannot be determined anymore. One should expect that Table 1 must be re-applied, but in practice this fails. However, if Table 2 is used instead, the whole process appears to function well. The author has no reasonable explanation for this discrepancy.

Operation	$\mathcal{A}$on$\mathcal{B}^+$	$\mathcal{A}$on$\mathcal{B}^-$	$\mathcal{B}$on$\mathcal{A}^+$	$\mathcal{B}$on$\mathcal{A}^-$
$\cup^*$	$\mathcal{A}$in$B$	$\mathcal{A}$out$B$	$\mathcal{B}$in$A$	$\mathcal{B}$out$A$
$\cap^*$	$\mathcal{A}$in$B$	$\mathcal{A}$out$B$	$\mathcal{B}$out$A$	$\mathcal{B}$in$A$
$-^*$	$\mathcal{A}$in$B$	$\mathcal{A}$out$B$	$\mathcal{B}$out$A$	$\mathcal{B}$in$A$

Table 2 Reclassification table for edge-edge coincidence

4.2. Repairs

In the implementation of (Bhardwaj & Malik, 1997) quite some effort has been put in a mechanism to repair cases where it is not possible to connect newly generated vertices. An obvious cause for such failure is that the previous, reclassification, step of the algorithm was imperfect; that is the same stage which caused us most trouble. Because the reclassification step is potentially unreliable, a repair possibility is always welcome. For this repair algorithm the Java source is available, but unfortunately the underlying assumptions and methods have not been published. In addition the authors of the program were not available for an illumination, so we have applied the repair mechanism without fully understanding it. Particularly there is an intriguing distinction between the treatment of 4-sided faces and that of other faces, for the background of which we cannot even speculate. However, it can be tracked that the repair steps are occasionally called upon (or, to be more precise, two of the four possible repair steps) and apparently they help.

4.3. Geometric consistency

Multiple authors have warned for the pitfall of geometric inconsistencies, and although the major burdens have been experienced with topological issues, the numerical issue also played a role in our efforts. Geometric inconsistency can be experienced with sectors that nearly coincide. Say, we have two sectors, S and T of different solids, and some arbitrary tolerance t. When evaluating S against the plane through T, the sectors are assumed to intersect if the distances of the three vertices from S to that plane T are smaller than t. At another stage of the algorithm a similar evaluation of T against the plane through S is performed and in theory both conclusions should be the same, but occasionally the two evaluations give a different outcome. Unfortunately there is no smart choice of t, either constant or algorithmically, to prevent this inconsistent behavior.
The occurrence of this phenomenon can be detected, because the reclassification of vertices and edges fails, while the repairs of Sub-section 4.2 provide no solution. For these cases we have implemented a 'retry' facility, where the reclassification of a sector is reprocessed, with each time a slightly different strategy. The standard method is as described: vertices of one sector are evaluated against the plane through the vertices of the other sector. On failure the edges of the sectors are normalized, which results in a different position of the end-vertices of each sector. Now these modified vertices are evaluated against the planes. If this mechanism also fails, in a preprocessing step for each sector pair, the end-vertices of each sector are explicitly projected into the plane of the other sector. This last step makes the sectors geometrically compatible, however at the price of a modification of the local geometry.
A second provision against geometrical inconsistency is performed prior to the Boolean operation itself. In this preprocessing step those vertices of one solid are detected which *nearly* coincide with an element of the other solid. These vertices are geometrically modified, to make them *fully* coincide. A drawback of this mechanism is that the geometry of the solids is slightly modified, which may be undesirable. In our application it may be known beforehand which solids can be modified, and which solids must remain unchanged. The sculptured hull for instance is polyhedralized into many small faces, so a change in the location of a modest number of vertices of the hull is not noticeable. A compartment on the other hand can be defined as one large single box, and modification of one of its vertices can be perceptible. So, one solid can be declared dominant over the other, and in the preprocessing step only the vertices of the non-dominant solid are touched.

4.4. Implementation details

Preprocessing and postprocessing

The Boolean operations are particularly sensitive to anomalies in the constituting solids. In this context with *anomaly* an allowed, but unnecessary and unexpected element is meant, for instance, two coinciding vertices in one solid. So, in order to make the Boolean operations more stable, in a preprocessing step the following anomalies are removed: coinciding vertices, 1- or 2-sided faces, strut-edges, neighbouring faces with face normals of opposite orientation, self-intersecting faces and empty inner loops.
Furthermore, after the Boolean-operation itself, a couple of postprocessing operations need to be done. The first is necessary after a union operation where one solid is enclosed entirely in the face of the other one. See for example Fig. 5, where in the resulting solid the lowest face of the upper part is an internal loop of the upper face of the lower part. Such a situation is topologically valid, but it appears to cause trouble at subsequent Boolean operations. For that reason the inner loop is topologically removed by connecting it with the outer loop.
A last postprocessing step is required to recognize or generate the information that makes our B-rep *extended*, such as continuous curves through adjacent edges, and their curved geometry.

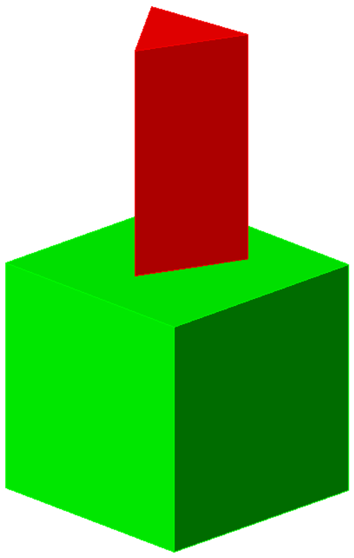

Figure 5 Two constituting parts meet at an inner loop

Speed enhancement

Our implementation is equipped with a rudimentary performance enhancing technique. Such mechanisms are not uncommon for solid modelling systems, see e.g. (Mäntylä & Tamminen, 1983) where a spatial cell structure is maintained which keeps track of geometric entities of the processed solids. Such a structure speeds up the geometric process, because it identifies which elements are not in each other's vicinity, so they can a priori be left out the investigations for intersection.
We applied a much simpler technique, with the advantage that it does not need a separate data structure. Our face and edge data structures are extended with a 'space' structure, which stores the center and the radius of a ball that contains the entire edge or face. Prior to each geometrical process between faces or edges it is checked whether their balls overlap. If that is not the case, they cannot intersect, and the intersection process is aborted. This mechanism seems to be working quite efficiently, but unfortunately there has not yet been an opportunity for quantification of the effect.

5. EXAMPLES

An example of a cargo hold which was composed by union operations on 11 non-overlapping sub-compartments was already presented in Fig. 2. In Fig. 6 a side compartment of a ship is shown, modelled as the union of non-overlapping sub-compartments, and subsequently intersected with the faceted sculptured ship hull. An example that includes the subtraction of a solid, which results in a through-hole, is shown in Fig. 7. Finally an example that shows a complete compartment model of a vessel is presented in Fig. 8

6. WEAKNESSES AND UNSOLVED PROBLEMS

Even after the discussed extensions and improvements there are a couple of problematic areas left, which will be discussed in the next sub-sections:

6.1. Non-polyhedral solids

Occasionally, in a ship, compartments are constructed which have warped boundaries, as in Fig. 9, and the software should accommodate this. At this moment our software detects such non-plane faces, and recursively triangulates them until the deviation between the warped surface and the points of the triangle are less than a certain tolerance (cf. (Toriya et al., 1991)).

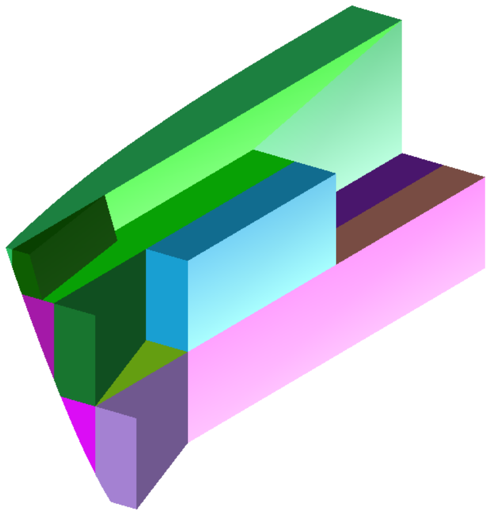

Figure 6 A realistic side tank

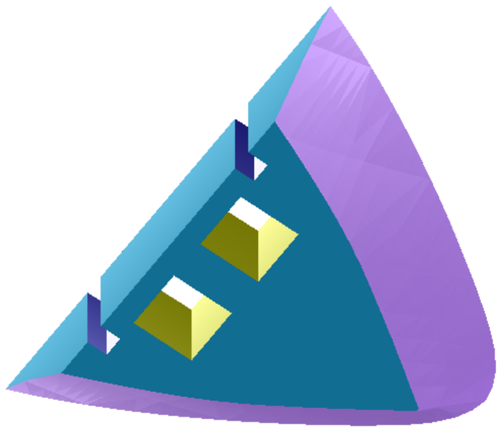

Figure 7 The forecastle of a ship

Although such a polygonal approximation is sufficient for visualization purposes and for Boolean operations with either properly intersecting or non-intersecting surfaces, with touching warped surfaces there is a problem. The reason is that two coinciding surfaces are triangularized independently, so their triangles do not coincide. This results in two dense triangle networks which are entwined in each other and generate an enormous amount of intersections. Apart from the question if the software works properly on such a configuration, the result with many resulting void pockets and intersection edges is undesirable.

One could consider to accommodate for solids with generally sculptured surfaces, but for our purposes that would be overkill. It will suffice to adapt the software to handle bi-linear surfaces explicitly. The representation of such a surface, and the geometric operations on it (e.g. intersection with a line), are well described in e.g. (Liming, 1979).

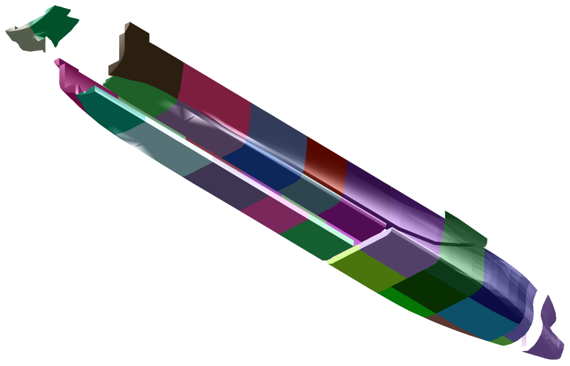

Figure 8 All fluid-containing compartments of a ship

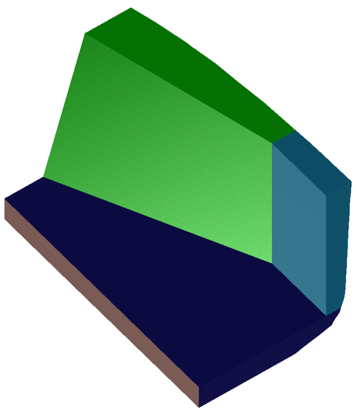

Figure 9 Compartment with warped boundary

6.2. Multishell objects

The B-rep data structure allows an object which consists of multiple unconnected shells, actually there is neither an explicit provision nor an implemented recognition mechanism for them. So all basic operations can be applied with such multishell objects, but at a certain stage during the Boolean operation all faces must be assigned to $\mathcal{A}\text{in}B$, $\mathcal{A}\text{out}B$, $\mathcal{B}\text{in}A$ or $\mathcal{B}\text{out}A$. Because at that stage the intersection edges and the topology of their neighbourhoods are known, each face which is bounded by an intersection edge is assigned accordingly. Subsequently the adjacent faces are given the same classification, in a recursive manner so that the neighbours will classify their neighbours and so on. But in this way a shell which is not connected to the shell which contains the intersection edge is never reached.

This effect can be repaired by some additional bookkeeping: a list of processed faces can be maintained, and faces that are finally unprocessed must belong to a distinct shell. Obviously, this shell does not intersect the other solid, so it suffices to investigate whether this shell is completely inside or completely outside the other solid, and to classify it accordingly.

A question we have to ask in this respect is if multishell objects do actually occur in our application. The answer is positive, a first possibility is with compartments which are intended to contain multiple shells, for instance a tank containing liquid which consists of two distinct parts that are connected by a pipe, but the dimensions of the pipe are so small that actually modelling it would be a bit overdone. Secondly, multishell object can unintendedly be created when a complex compartment is modelled, from the union or difference between sub-compartments, with a typical number of 10 to 20. The sub-compartments are defined by the user in an arbitrary way, thus the definition sequence can lead to an intermediate solid containing multiple shells. The fact that at a later stage another sub-compartment will be added which unites the two shells again, leaves unaffected that the possibility of multiple shells must be supported. The alternative, to burden the user with the responsibility to choose a sequence that will not create multiple shells, is not particularly user-friendly.

6.3. Nonmanifold topology

The applied methods are designed for solids with manifold topology. For our purposes that is sufficient because the objects we aim at do not have nonmanifold characteristics, such as more that 2 faces meeting at a common edge. However, in the course of the composition process, unintendedly, a nonmanifold object may occur in an intermediate step. Take again the compartment of Fig. 2, which is a manifold solid. Depending on the definition sequence of the sub-compartments an intermediate solid could be created as in Fig. 10, where the last two sub-compartments (which are colored purple in Fig. 2) still have to be added. But in this solid four edges meet at a common edge, so it is not manifold anymore, which hampers further processing. We will try to tackle this problem by automatically sorting the sub-compartments, so that each intermediate solid will remain manifold.

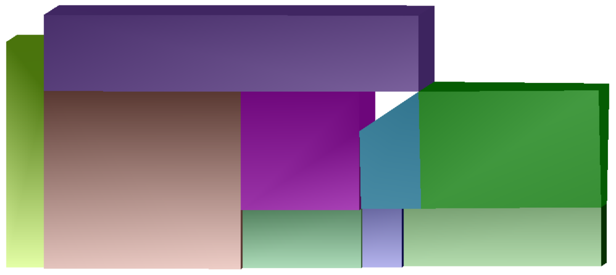

Figure 10 Intermediate stage in the construction process of the hold of Fig. 2

7. CONCLUSION

In this paper we have reported the quest for a practical implementation of Boolean set operations on solids bounded by plane and bilinear surfaces. The job took quite some effort, which was rather unexpected because in the literature a plurality of methods is presented and described into detail, which gave us the impression that this particular subject is rather outengineered. That appeared not to be the case, there are still many pitfalls, not always major ones, but even a small one can make a Boolean operation to fail completely. However, in hindsight there have been a couple of indications for the non-perfectness. In a private communication with an implementor of an extensive modelling package he remarked that it is occasionally necessary to give touching solids a minuscule rotation in order to finish a Boolean operation successfully. Also in (Segal & Sequin, 1988) it is mentioned that a user is signalled if the program has difficulty to determine whether a vertex is included in a face. It is a very weak ‘solution’, but also our system can now show the message *Intersections with this model failed. Please try a tiny shift of one of the bounding planes*.

Anyway, with the proposed modification a workable system emerged, although a number of items still has to be finished or improved:

- All three operations $\cup^*$, $\cap^*$ and $-^*$ are explicitly included. The program could be more transparent, and possibly more reliable, with only one implemented function, in combination with relations as in Eqs. 1.
- The explicit support for multishell objects, and

warped faces, as discussed in Section 6.

- A quantitative analysis of the speed enhancements of Sub-section 4.4.
- Investigate whether the users can be persuaded to apply more fully intersecting objects, instead of touching ones. An obstacle in this respect is that the users are educated to a touching configuration, because in the previous, wireframe-oriented, version of the software this was the only allowed method. To make things worse, even today the higher performance of the wireframe model is sometimes a reason to use this representation for calculation intensive tasks, such as the calculation of damage stability. So either the wireframe method must be modified so that it also accommodates intersecting sub-compartments, or the user must maintain two alternative sub-compartment configurations for each compartment.

And of course the last couple of bugs have to be removed from our software. Surely they are present, we only don’t know where yet.

Appendix A. SOURCE CODE MODIFICATIONS

During the development process a couple of improvements on published algorithms have been made, which are, for the record, discussed in this appendix.

Appendix A.1. Insertion of null edges

On page 292 of (Mäntylä, 1988) program 15.11 is listed, which handles the insertion of null edges. It determines where to insert new halfedges, and distinguishes between the cases that new halfedges must be inserted between two distinct existing halfedges, or only to one existing halfedge. The first case is processed by function `separ1`, the second by `separ2`. However, the case that in *both* solids A and B halfedges must be inserted to one existing halfedge is not covered. The next modification will do:

```
if (ha1 == ha2)
{
  separ2(ha1,0)
  if (hb1 == hb2) then
    separ2(hb1,1)
  else
    separ1(hb1,hb2,1)
} else if(hb1 == hb2)
{
  separ2(hb1,1)
  separ1(ha2,ha1,0)
}
else
{
  separ1(ha2, ha1, 0)
  separ1(hb1, hb2, 1)
}
```

Furthermore the implementation of `separ2` consists of one call to `separ1`. This appeared to be insufficient because multiple nulledges may be connected to a single vertex. In the subsequent joining process arbitrary nulledges can be connected, which causes the vertices to be connected to appear in distinct faces. But situated in distinct faces these vertices cannot be connected anymore. The phenomena is tackled by placing each strut edge in a private (internal) ring:

```
void       separ2(he, type)
HalfEdge   *he;
int        type;
{
  HalfEdge   *he2;
  he2 = he->prv;
  lmev(he,he,++maxv,
       he->vtx->vcoord[0],
       he->vtx->vcoord[1],
       he->vtx->vcoord[2]);
  lmev(he,he,++maxv,
       he->vtx->vcoord[0],
       he->vtx->vcoord[1],
       he->vtx->vcoord[2]);
  he2 = he2->nxt;
  lkemr(he2,mate(he2));
  if(type==0)
    sonea[nedgea++] = he->edg
  else
    soneb[nedgeb++] = he->edg
}
```

Appendix A.2. Multiple intersecting sectors

The same nulledge insertion function traverses the list of sectors until two intersecting ones are found. The implicit mechanism is that two subsequent sectors of the list form a matching pair. With more than two sectors in the list this does not necessarily have to be the case; the solution is to treat the list as a closed loop, and repetitively scan it until all matching pairs are found.

Appendix A.3. Edge-edge coincidence

In (Mäntylä, 1986), §6.2.2.2, the case of coinciding edges is discussed. Additional to the basic method a provision must be made for the recognition of on-edges. In a first implementation the identifiers `s1a` etc. (from the structure `nsectors` on page 279 of (Mäntylä, 1988)) have been utilized for this purpose, but due to the fact that these identifiers may have been redefined in the previous sector-sector processing part, they are not suitable. Instead we use additional identifiers which save the status of the initial edge-sector comparison of program 15.7.
Furthermore one has to bear in mind that on-edges can either indicate that the edges do actually coincide, as in Fig. 11.a, or that the edges share only a common line as indicated in Fig. 11.b.

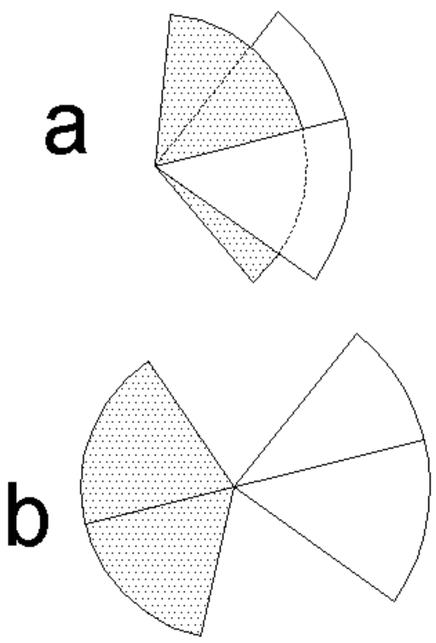

Figure 11 Two cases of coinciding edges

Appendix A.4. Loop gluing

On page 212 of (Mäntylä, 1988) the loop gluing algorithm is listed. In one particular case an outer loop with a strut appeared, combined with an inner loop with two coinciding vertices. This was a fail-case, so as a preprocessing step these anomalies will be removed. Furthermore, a degenerated face (a face with only two edges) needs extra attention. This is implemented in the procedure as partially listed below:

```
void   loopglue(fac)
{
  int  degenerated_face;
  .............
  degenerated_face =
    h1->nxt == mate(h1);
  while(h1->nxt != h2)
  {   ........
      if (degenerated_face == 1)
        break;
  }
  if (degenerated_face == 0)
    lkef(mate(h1),h1);
}
```

REFERENCES

Bhardwaj, A. & Malik, I. (1997). Java Applet for Constructive Solid Geometry. Technical report, Computer Science Department, Cornell University.

Chiyokura, H. (1988). Solid Modelling with Designbase. Addison Wesley.

Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. (1990). Computer Graphics: Principles and Practice. Reading, MA: Addison-Wesley. second edition.

Gardan, Y. & Perrin, E. (1996). An algorithm reducing 3d boolean operations to a 2d problem: concepts and results. Computer Aided Design, 28(4), pp. 277–287.

Hoffmann, C. (1989). Geometric and Solid Modeling. San Mateo, CA: Morgan-Kaufmann.

Hoffmann, C. M., Hopcroft, J. E., & Karasick, M. T. (1987). Robust Set Operations on Polyhedral Solids. Technical Report 87-875, Cornell University, Computer Science Department, Ithaca, New York.

Hu, C.-Y., Patrikalakis, N. M., & Ye, X. (1996). Robust interval solid modeling, Part II: boundary evaluation. Computer Aided Design, 28, pp. 819–830.

Keyser, J., Culver, T., Foskey, M., Krishnan, S., & Manocha, D. (2002). ESOLID-A System for Exact Boundary Evaluation. In Proc. 7th ACM Symposium on Solid Modeling and Applications Saarbrücken, Germany.

Koelman, H. J., Horváth, I., & Aalbers, A. (2001). Hybrid Representation of the Shape of Ship Hulls. International Shipbuilding Progress, 48(3), pp. 247–269.

Krishnan, S., Manocha, D., Gopi, M., Culver, T., & Keyser, J. (2001). BOOLE: A Boundary Evaluation System for Boolean Combinations of Sculptured Solids. International Journal of Computational Geometry and Applications, 11(1), pp. 105–144.

Lee, K.-Y., Lee, S.-U., Cho, D.-Y., Roh, M.-I., & Kang, S.-C. (2003). An innovative compartment modeling and ship calculation system. In Proc. 8th International Marine Design Conference Athens Greece.

Liming, R. A. (1979). Mathematics for Computer Graphics. Fallbrook, CA: Aero Publishers.

Mäntylä, M. J. (1986). Boolean Operations of 2-Manifolds through Vertex Neighborhood Classification. ACM Transactions on Graphics, 5(1), pp. 1–29.

Mäntylä, M. J. (1988). An Introduction to Solid Modeling. Rockville, MD, USA: Computer Science Press.

Mäntylä, M. J. & Sulonen, R. (1982). GWB: A solid modeler with Euler operators. IEEE Comput. Graph. Appl., 2(5), pp. 17–31.

Mäntylä, M. J. & Tamminen, M. (1983). Localized set operations for solid modeling. Computer Graphics, 3(17), pp. 279–288.

Requicha, A. A. G. (1980). Representations of rigid solids: Theory, methods, and systems. ACM Comput. Surv., 12, pp. 437–464.

Requicha, A. A. G. & Voelcker, H. B. (1984). Boolean operations in solid modelling: boundary evaluation and merging algorithms. Report, College Engrg. Appl. Sci., Univ. Rochester, Rochester, NY.

Segal, M. & Sequin, C. H. (1988). Partitioning polyhedral objects into nonintersecting parts. IEEE Comput. Graph. Appl., 8(1), pp. 53–67.

Toriya, H., Takamura, T., Satoh, T., & Chiyokura, H. (1991). Boolean operations for solids with free-form surfaces through polyhedral approximation. Visual Computer, 7, pp. 87–96.

Tsuzuki, M. & Shimada, M. (2003). Geometric classification tests using interval arithmetic in b-rep solid modelling. J. Braz. Soc. Mech. Sci. & Eng., 25(4).