A Topologically Robust Boolean Algorithm Using Approximate Arithmetic

Julian M. Smith * Neil A. Dodgson †

Abstract

We present a previously unpublished, topologically robust algorithm for Boolean operations on polyhedral boundary models. The algorithm can be proved always to generate a result with valid connectivity if the input shape representations have valid connectivity, irrespective of the type of arithmetic used or the extent of numerical errors in the computations or input data. The main part of the algorithm, known as the basic Boolean algorithm, is based on a series of interdependent operations. The relationship between these operations ensures a consistency in the intermediate results that guarantees correct connectivity in the final result. Either a triangle mesh or polygon mesh can be used. The algorithm described can be extended naturally to the problem of computing the overlay of two cellular subdivisions, and also to operate in higher-dimensional domains.

1 Introduction

Problems of robustness are a major cause for concern in the implementation of computational geometry algorithms ([2, 5, 6, 7]). Most geometrical algorithms are a mix of numerical and combinatorial computations, and the approximate nature of the former often leads to logical decisions that are inconsistent, thus hindering the construction of a combinatorially correct result. In the context of boundary representations, inconsistent computations can lead to connectivity faults. There is also the problem that numerical errors can lead to the computed boundary intersecting itself.

A number of proposals have been put forward to address this problem. One approach favoured, particularly in the polyhedral domain, is to rely on exact arithmetic to avoid altogether the problems of numerical errors and inconsistencies [1, 4, 8]. Floating point filters are often used in conjunction to reduce the overhead of exact computations.

Certain methods take the topology-oriented approach. These are designed to guarantee a topologically or combinatorially correct result, irrespective of the extent of any numerical error in the computations or in the input data. As such they can be implemented using standard floating point arithmetic. Sugihara et al. [9] refer to such techniques for the Voronoi and Delaunay problems, the convex hull problem, and also for the intersection of convex polyhedra.

The algorithm presented here can be classed as topology-oriented. The main part of the algorithm, the *basic Boolean algorithm*, consists of a series of interdependent operations guaranteed to yield consistent intermediate results. This in turn ensures the final result has valid connectivity, provided both input structures have valid connectivity. The algorithm can be implemented using either a triangle or polygon mesh. For the triangle mesh variant of the basic algorithm there is a requirement to break up non-triangular facets into triangles.

The structure generated by the basic algorithm may have features that are redundant or are close to making the structure geometrically invalid. For that reason it is generally preferable to apply a data smoothing process to the structure to make it suitable for downstream operations.

The basic Boolean algorithm is described in section 2. Section 3 briefly discusses topological robustness and also the need for data smoothing. The process for triangulating facets is not covered.

2 The basic Boolean algorithm

The basic algorithm for the Boolean operation between two shapes A and B is performed as a series of interdependent operations. Each operation determines the relation between two entities, one from each shape, an *entity* being a vertex, edge (or half-edge), facet, or the entire shape. Hence there are 16 types of operation: one for each pairing of the four types of entity. Each operation type is considered as belonging to a particular level, 0 to 6, equal to the sum of the manifold dimensionalities of the two entities, o_A and o_B. Each operation has a similar pattern: the result is influenced by the results of those operations one level lower, concerning (1) each boundary component of o_A in turn, and o_B; and (2) o_A, and in turn each boundary component of o_B. This leads to a dependency hierarchy between the types of operation, as shown in figure 1. Operations at level 0-3 determine the point at which the entities intersect, while those at level 3-6 work towards constructing the result.

The operations at level 3 take a pivotal role between

*Rainbow Group, Computer Laboratory, University of Cambridge, jms222@cl.cam.ac.uk

†Rainbow Group, Computer Laboratory, University of Cambridge, nad@cl.cam.ac.uk

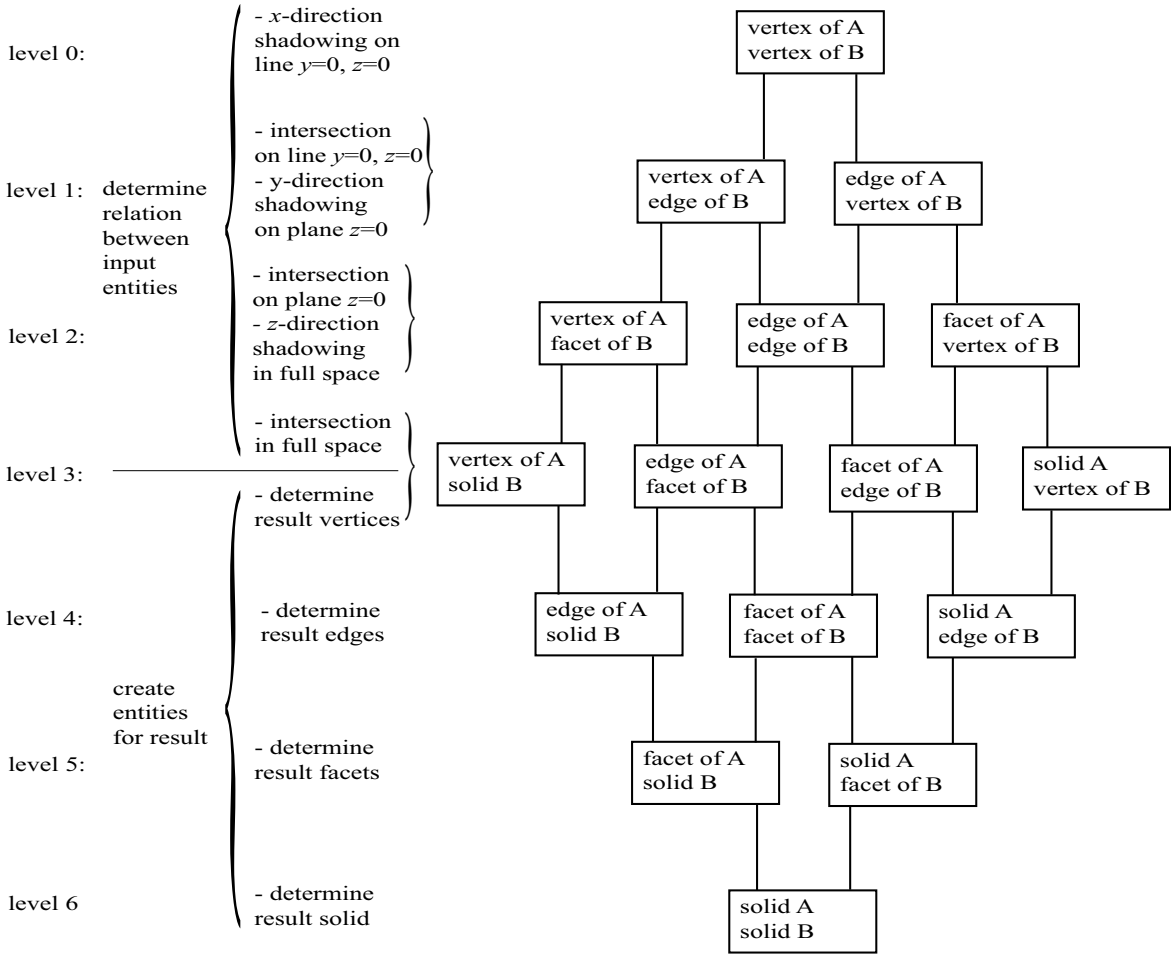

Figure 1: The hierarchy of operations for the basic Boolean algorithm.

the two stages. They determine whether a vertex of one solid lies inside or outside the other solid, and whether an edge of one solid intersects a facet of the other solid. The latter operation also determines the point of intersection. All operations assume object B to be perturbed by an infinitesimal distance in each of the three axis directions. This resolves the problem of degeneracy, in the manner described by Edelsbrunner and Mücke [3] in the context of exact computations. Thus a point is deemed to lie inside or outside a solid, never on the boundary; likewise, an edge either intersects a facet or it does not, they are never deemed simply to touch. This makes special case handling unnecessary.

An *intersection function*, $X_{ij}(o_A, o_B)$, ascertains whether entities o_A and o_B intersect, i and j indicating the manifold dimensionality of each entity. A non-zero function value indicates that the entities intersect; the sign of the function value indicates the nature of the intersection, such as whether the edge enters or exits the solid through the facet. Intersection functions are used at lower levels too, where $i+j<3$. In general terms, $X_{ij}(o_A, o_B)$ operates in $(i+j)$-dimensional space or subspace, as specified by the first $i+j$ coordinate values of the Cartesian representation used in the computation. Hence level 3 intersection functions ($i+j=3$) operate in full 3D space. Level 2 intersection functions ($i+j=2$) operate in effect in the (x, y) plane, so z coordinate values are ignored when determining if the entities intersect. However if they do intersect, the z-values of the intersection point are determined for the use in subsequent calculations. Two such values are required: one for each entity. In turn, level 1 intersection functions operate simply on the (x) line, and two sets of y and z values are determined at any intersection

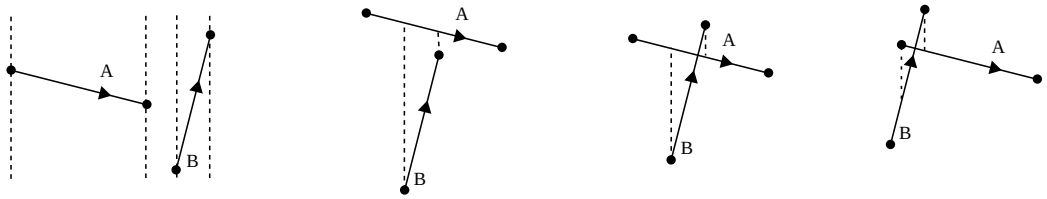

Figure 2: Demonstration of how intersection between two edges in (x, y) space is determined. The two edges intersect only if they overlap in (x) space ($\max x_B \geq \min x_A$ and $\min x_B < \max x_A$) and if $y_B \geq y_A$ at one end of the overlap range in (x) and $y_B < y_A$ at the other (i.e. at $x = \max(\min x_A, \min x_B)$ and $x = \min(\max x_A, \max x_B)$).

point. One can also consider there to be a level 0 'intersection function', $X_{00}(v_A, v_B)$, that operates at the origin point; it always takes the value 1, with the 'intersection point' located at v_A and v_B.

The calculations involve *shadow functions*, $S_{ij}(o_A, o_B)$, that operate in $(i+j+1)$-dimensional space or subspace as defined by the first $i+j+1$ coordinate values. If $X_{ij}(o_A, o_B) = 0$ then $S_{ij}(o_A, o_B) = 0$. If $X_{ij}(o_A, o_B) \neq 0$, the value is determined by considering the next coordinate value at the point of intersection: ξ_A on o_A and ξ_B on o_B, where ξ is x if $i+j=0$, y if $i+j=1$, or z if $i+j=2$. $S_{ij}(o_A, o_B) = X_{ij}(o_A, o_B)$ if $\xi_B \geq \xi_A$, or 0 otherwise.

The intersection function $X_{ij}(o_A, o_B)$ is evaluated as the sum, with appropriate $+/-$ signs, of each of the shadow function values: $S_{i-1,j}(c, o_B)$ for each boundary component c of o_A (if $i > 0$); and $S_{i,j-1}(o_A, c)$ for each boundary component c of o_B (if $j > 0$). The individual formulae are listed in table 1.

The intersection point associated with a non-zero value of $X_{ij}(o_A, o_B)$ is determined by considering two lower-level intersection points associated with a non-zero value of $X_{i-1,j}$ or $X_{i,j-1}$, one for which $\xi_B \geq \xi_A$ and one for which $\xi_B < \xi_A$. The intersection point for X_{ij} is obtained by linearly interpolating the two lower-level intersection points so that $\xi_B = \xi_A$. Figure 2 demonstrates this for the case $X_{11}(e_A, e_B)$.

Following the determination of the level 3 intersection points the structure representing the result is constructed. The result contains two types of vertices: retained vertices, which are copies of original vertices that lie on the appropriate side of the other solid (outside for the union operation, inside for the intersection operation), and intersection vertices, which are new vertices located at each point where an edge intersects a facet.

The operations at level 4 construct the edges: retained edges (and part-edges), and intersection edges (between two facets). For each operation, a series of start-vertices and end-vertices is obtained from the level 3 operations, and these are paired up to form edges in the new structure.

Retained facets (or part-facets) are determined at level 5; these are bordered by half-edges computed at level 4. For the triangle mesh variant of the algorithm, any non-triangular facet must be broken up into triangles. For the polygon mesh variant, the polygonal region is retained as it is.

Finally, at level 6 the resulting solid is generated from the retained facets.

3 Topological robustness, numerical accuracy, and implementation

Topological robustness of the basic algorithm is assured by the formulae used, assuming A and B both have valid connectivity. For the calculations at levels 1, 2 and 3, it can be proved that if $X_{ij} \neq 0$, then there will be at least one lower-level intersection where $\xi_B \geq \xi_A$ and also one where $\xi_B < \xi_A$, hence it will always be possible to interpolate to determine the point of intersection. At level 4, it can be proved that the number of start-vertices will always equal the number of end-vertices, so it will always be possible to determine segments from which to construct edges. At level 5, it can be shown that every end-vertex of a half-edge of a particular facet is also the start-vertex of a half-edge of the same facet, and at level 6, every half-edge going from vertex v to w is matched by a half-edge going from vertex w to v. Hence the structure generated satisfies the constraints that ensure it has correct connectivity. Proofs of these statements will be published separately.

Geometrical validity of the result is *not* assured by the basic Boolean algorithm. The structure generated can have artifacts that can be considered unsuitable: coincident vertices, coincident and opposing facets, coincident and opposing half-edges within a facet, zero-length edges, and zero-area facets. Coincident and opposing facets or half-edges make the structure on the borderline of being geometrically invalid; it may even turn out invalid due to numerical errors in computing the result, or else it may become invalid after applying a subsequent operation such as a repositioning translation. It is therefore appropriate to process the structure to remove the artifacts before passing on the data structure to any process likely to be adversely affected by geometrical errors. This data

level	*formula*		*situation leading to value of* $+1$
0	$X_{00}(v_A, v_B) =$	1	
1	$X_{01}(v_A, e_B) =$	$S_{00}(v_A, v_e(e_B)) - S_{00}(v_A, v_s(e_B))$	v_A lies within e_B, with e_B going left to right
	$X_{10}(e_A, v_B) =$	$-S_{00}(v_e(e_A), v_B) + S_{00}(v_s(e_A), v_B)$	v_B lies within e_A, with e_A going left to right
2	$X_{02}(v_A, f_B) =$	$-\sum_{h\in\partial f_B} S_{01}(v_A, h)$	v_A lies within f_B, with f_B going anti-clockwise (denoting that it faces upwards)
	$X_{11}(e_A, e_B) =$	$S_{01}(v_e(e_A), e_B) - S_{01}(v_s(e_A), e_B)$ $+S_{10}(e_A, v_e(e_B)) - S_{10}(e_A, v_s(e_B))$	e_A crosses e_B from left to right
	$X_{20}(f_A, v_B) =$	$\sum_{h\in\partial f_A} S_{10}(h, v_B)$	v_B lies within f_A, with f_A going anti-clockwise
3	$X_{03}(v_A, B) =$	$\sum_{f\in\partial B} S_{02}(v_A, f)$	v_A lies within B, with faces facing outwards
	$X_{12}(e_A, f_B) =$	$-S_{02}(v_e(e_A), f_B) + S_{02}(v_s(e_A), f_B)$ $-\sum_{h\in\partial f_B} S_{11}(e_A, h)$	e_A crosses f_B going to the outer side
	$X_{21}(f_A, e_B) =$	$-\sum_{h\in\partial f_A} S_{11}(h, e_B)$ $+S_{20}(f_A, v_e(e_B)) - S_{20}(f_A, v_s(e_B))$	e_B crosses f_A going to the outer side
	$X_{30}(A, v_B) =$	$-\sum_{f\in\partial A} S_{20}(f, v_B)$	v_B lies within A, with faces facing outwards

Table 1: Formulae for intersection functions. ∂f denotes the collection of half-edges that border facet f; $v_s(e)$ and $v_e(e)$ denote the start- and end-vertex of edge (or half-edge) e.

smoothing process consists of a number of Euler-type operations (as described in [5]), simplifying the structure while preserving topological correctness, and restricting any change to the shape boundary (which may be necessary) to within specified bounds.

The first author devised and successfully implemented the Boolean algorithm for use within a widely circulated commercial CAD product, for which Boolean operations are used principally to convert CSG models of industrial plant components to polyhedral approximations. The product and algorithm continue to be used extensively several years after release. The general polygonal mesh variant of the algorithm was adopted, since it was found to be more efficient than an initial trial version based on the triangular mesh. It was found to be more efficient to apply the data smoothing process once at the end of a sequence of basic Boolean operations rather than after each individual operation. The data smoothing process that was implemented does not have a full theoretical backing, so it is not provably fully robust in performing the task required of it, but it has turned out to be sufficiently reliable. Users of the released product only ever reported one fault that turned out to be related to the data smoothing process: a problem of non-termination, which was resolved by enforced early termination.

References

[1] CGAL. http://www.cgal.org/, 2006.

[2] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry*. Springer, 1997.

[3] H. Edelsbrunner and E. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1), January 1990.

[4] S. Fortune. Polyhedral modelling with exact arithmetic. In *Proc. 3rd Symp. Solid Modeling*, pages 225–234. ACM Press, NY, 1995.

[5] C. Hoffmann. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann Publishers, Inc., 1989.

[6] C. Hoffmann. Robustness in geometric computations. *Journal of Computing and Information Science in Engineering*, 1:143–156, 2001.

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

[8] K. Mehlhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[9] K. Sugihara, M. Iri, H. Inagaki, and T. Imai. Topology-oriented implementation—an approach to robust geometric algorithms. *Algorithmica*, 27:5–20, 2000.