3D-Boolean Operations in *CGAL* (draft-specification)

Carl Van Geem, Wolfgang Freiseisen

6. March 1997

Chapter 1

Boolean Operations

1.1 Boolean Operations on Polyhedra

CGAL also provides boolean operations for polyhedra in the 3-dimensional Euclidean space. Similarly to the 2-dimensional case we have a non-closure version working on simple polyhedra and a closure version working on 3-dimensional subdivisions. We start with the non-closure version that will work on simple 3-dimensional polyhedra.

Definition

For further discussions we should clarify the notion of polyhedron and polytope, since these definitions are not standardized in literature. The following definition could be found in [6]:

Polyhedron: A polyhedron in $\mathbb{E}^3$ is a bounded subset of $\mathbb{E}^3$ and it consists of its interior and its boundary. The boundary of a polyhedron is defined by a finite set of plane polygons, such that every edge of a polygon is shared by exactly one other polygon and no subset of polygons has this same property. The vertices and edges of polygons are the vertices and edges of the polyhedron and the polygons are the facets of the polyhedron.

higher dimensions: In the literature polyhedra are also defined for higher dimensions, but for our purposes this 3-dimensional definition is sufficient. A polyhedron is simple if there is no pair of nonadjacent facets sharing a point. A simple polyhedron is convex if its interior is a convex set.

holes and orientation: A polyhedron as described here now can be oriented (as objects in the real world are). This allows to represent in a practical way (internally in the algorithms for boolean operations as well as externally to the user) a polyhedron P with holes as a list *(P_0, P_1, P_2, ..., P_h)* of polyhedra whereby *P_0* determines the outer boundary of P and whereby each of the following ones determine the boundary of one of the holes: the orientation of *P_0* will be opposite to the orientation of each of the other polyhedra.

The result of a boolean operation on polyhedra may also lead to holes of dimension lower than 3, that is in "missing" facets, edges, or vertices, or "missing" polygons, segments, or points, or a combination of these. "Missing" facets and polygons will have clockwise ordered edges.

Convex Polytope: The intersection of a finite number of half-spaces in $\mathbb{E}^d$ is called a *(convex) polytope*.

Bounded Convex Polytope: The convex hull of a finite set of points in $\mathbb{E}^d$ is a *bounded (convex) polytope*.

remark: Obviously a bounded convex polytope is a convex polytope. Note however that the boundary of a bounded polytope belongs to the polytope, whereas this is not necessarily so for any (convex) polytope.

A d-polytope: A polytope P is called a *d-polytope* iff $dim(P) = d$.

examples: a convex polygon is a 2-polytope, a convex polyhedron is a 3-polytope.

In the following we will consider boolean operations on simple polyhedra. A polyhedron on which the boolean operations will be performed can be one of the following CGAL-objects:

- *CGAL_Tetrahedron_3*<*R*> ... 3D tetrahedron.
- *CGAL_Polyhedron_3*<*R*> ... a simple 3D polyhedron as defined above.

For two simple polyhedra A and B, the boolean operations defined are:

Intersection test of two polyhedra (*CGAL_do_intersect(A,B)*): Checks if the two polyhedra A and B do intersect without computing the intersection area and returns *true* if they do intersect. returns *true* if the polyhedra A and B do intersect, otherwise *false* will be returned.

Intersection of two polyhedra (*CGAL_intersection(A,B)*): performs the operation $C := A \cap B$ and returns the result C. The result C is a possibly empty list of *CGAL_Object*s. The list can contain objects of type *CGAL_Point_3*<*R*>, *CGAL_Segment_3*<*R*>, *CGAL_Polygon_3*<*R*>, and *CGAL_Polyhedron_3*<*R*>.

Union of two polyhedra (*CGAL_union(A,B)*): performs the operation $C := A \cup B$ and returns the result C. The result C is a possibly empty list of *CGAL_Object*s. If $A \cap B$ is empty, then the empty list will be returned. The list can contain objects of type *CGAL_Polyhedron_3*<*R*>, whereas the first polyhedron gives the outer boundary of the contour of the union and all (if any) following polyhedra define the inner contours (the holes).

Difference of two polyhedra (*CGAL_difference(A,B)*): performs the operation $C := A \setminus interior(B)$ and returns the result C as a maybe empty list of *CGAL_Object*. The list can contain objects of type *CGAL_Point_3*<*R*>, *CGAL_Segment_3*<*R*>, *CGAL_Polygon_3*<*R*>, and *CGAL_Polyhedron_3*<*R*>. If $A \cap B$ is empty then A will be returned. If $A \cap B = A$ (i.e. $A \subseteq B$), then the empty list will be returned. If $A \cap B = B$ (i.e. $B \subseteq A$, that means B is a hole of A), then a list containing (two) objects (A, B) of type *CGAL_Polyhedron_3*<*R*> will be returned, whereas the second one represents a hole.

Operations

The operations work on tetrahedra, and simple polyhedra, in general. However if we have for instance two convex polyhedra the operations switch (if possible) to algorithms working more efficient for this kind of polyhedra.

bool *CGAL_do_intersect(CGAL_Tetrahedron_3*<*R*> *A, CGAL_Tetrahedron_3*<*R*> *B)*

returns *true* if the tetrahedra A and B do intersect.

bool *CGAL_do_intersect(CGAL_Polyhedron_3*<*R*> *A, CGAL_Polyhedron_3*<*R*> *B)*

returns *true* if the polyhedra A and B do intersect.

CGAL_Object

CGAL_intersection(CGAL_Tetrahedron_3<*R*> *A, CGAL_Tetrahedron_3*<*R*> *B)*

computes the intersection of two tetrahedra and returns the result as a *CGAL_Object*, which can be a point (*CGAL_Point_3*), a segment (*CGAL_Segment_3*), a polygon (*CGAL_Polygon_3*), a tetrahedron (*CGAL_Tetrahedron_3*), a convex polyhedron (*CGAL_Polyhedron_3* with *is_convex() == true*), or empty intersection.

list<CGAL_Object>

CGAL_intersection(CGAL_Polyhedron_3<R> A, CGAL_Polyhedron_3<R> B)

computes the intersection of two simple polyhedra and returns the result as a maybe empty list of *CGAL_Object*.
Precondition: A and B are simple polyhedra.

CGAL_Object

CGAL_union(CGAL_Tetrahedron_3<R> A, CGAL_Tetrahedron_3<R> B)

computes the union of two tetrahedra and returns the result as a *CGAL_Object*, which can be a tetrahedron (*CGAL_Tetrahedron_3*), a simple polyhedron (*CGAL_Polyhedron_3*), or empty (if $A \cap B$ is empty).

list<CGAL_Object>

CGAL_union(CGAL_Polyhedron_3<R> A, CGAL_Polyhedron_3<R> B)

computes the union of two simple polyhedra and returns the result as a maybe empty list of *CGAL_Object*.
Precondition: A and B are simple polyhedra.

list<CGAL_Object>

CGAL_difference(CGAL_Tetrahedron_3<R> A, CGAL_Tetrahedron_3<R> B)

computes the difference ($A \setminus B$) of two tetrahedra and returns the result as a maybe empty list of *CGAL_Object*, where each object be a tetrahedron (*CGAL_Tetrahedron_3*), or a simple polyhedron (*CGAL_Polyhedron_3*). If $B \subseteq A$ a empty list will be returned.

list<CGAL_Object>

CGAL_difference(CGAL_Polyhedron_3<R> A, CGAL_Polyhedron_3<R> B)

computes the difference of two simple polyhedra ($A \setminus B$) and returns the result as a maybe empty list of *CGAL_Object*.
Precondition: A and B are simple polyhedra.

Implementation

As in the 2-dimensional case it will be decided internally which of the algorithms will be used for the computation. The choice depends on the type of the polyhedra.

Internally one of the basic data structures represents a polyhedron without holes. This will be based upon the data structure for 3D polyhedral surfaces in a half-edge representation (for a surface being an orientable 2-manifold).

1.2 Boolean Operations on Three-Dimensional Subdivisions

Definition

Similar to 2-dimensional boolean operations we will use subdivisions for performing the closure case, i.e. the boolean operations should be applied again on the preceding result. In the 2-dimensional case we used planar maps (*CGAL_Planar_map_2<R>*) as 2-dimensional subdivisions. This concept leads immediately to 3-dimensional subdivisions, so called 3D-hyper maps [1].

Each simple polyhedron can be embedded into a 3-hyper map. For an detailed definition we refer to [2].

Operations

bool *CGAL_do_intersect(CGAL_Hyper_map_3<R> A, CGAL_Hyper_map_3<R> B)*

returns true if the intersection of the two 3-hyper maps is not empty.

CGAL_Hyper_map_3<R>
CGAL_intersection(CGAL_Hyper_map_3<R> A, CGAL_Hyper_map_3<R> B)

performs an intersection of two 3-hyper maps and returns the result as a new 3-hyper map.

CGAL_Hyper_map_3<R>
CGAL_union(CGAL_Hyper_map_3<R> A, CGAL_Hyper_map_3<R> B)

performs an union of two 3-hyper maps and returns the result as a new 3-hyper map.

CGAL_Hyper_map_3<R>
CGAL_difference(CGAL_Hyper_map_3<R> A, CGAL_Hyper_map_3<R> B)

performs a difference (A without B) of two 3-hyper maps and returns the result as a new 3-hyper map.

Implementation

A 3-hyper map is implemented as a 3-dimensional "planar" subdivision consisting of vertices, edges, facets (or polygons), and regions (or 3-polytopes) whereas each region can be filled.

[1] The notion hyper map is not standardized in the literature, however we think that it is most suitable for our purposes.

1.3 Remarks and Questions

- The definition of the polytope and the polyhedron should be clear now.
- The representation of a polyhedron (*CGAL_Polyhedron_3*) and the underlying data structure is quite important. It seems that the basic structure might/will become a half-edge data structure for 3D polyhedral surfaces. Is an orientation of the polyhedron foreseen in *CGAL_Polyhedron_3* (as it is for *CGAL_Polygon_2*)?
- It would be elegant to use *CGAL_Polyhedron_3* for the 4-dimensional objects in the way we use *CGAL_Polygon_2* for *CGAL_Polyhedron_3*. Is this also desirable from a practical point of view – especially when it comes to the representation of orientation.
- What about other 3D-objects like cube or iso-oriented cube?
- Here the definition of a 3-hyper map is not worked out yet, since the exact one will depend on that of the planar map.

Bibliography

[1] Erik Brisson, *Representing Geometric Structures in d Dimensions: Topology and Order*, Proceedings of the fifth annual symposium on computational geometry, 1989, pp.218–227.

[2] David P. Dobkin, Michael J. Laszlo, *Primitives for the Manipulation of Three-Dimensional Subdivisions*, Proceedings of the third annual symposium on computational geometry, Canada, June 8-10, 1987, pp.86–99.

[3] L. Guibas, J. Stolfi, *Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams*, ACM Trans. on Graphics, Vol. 4, No. 2, April 1985, pp. 75–123.

[4] David M. Mount, *Storing the Subdivision of a Polyhedral Surface*, Proceedings of the second annual symposium on computational geometry, New York, June 2-4, 1986, pp.150–158.

[5] Thomas Ottmann, Peter Widmayer, Derick Wood, *A Fast Algorithm For Boolean Mask Operations*, Forschungsbericht 112, Institut für angewandte Informatik, Universität Karlsruhe, Juni 1982.

[6] Franco P. Preparata, Michael I. Shamos, *Computational Geometry. An Introduction*, Springer-Verlag, New York, 1985.