Boolean Operations on 3D Selective Nef Complexes: Optimized Implementation and Experiments*

Peter Hachenberger Lutz Kettner

Abstract

Nef polyhedra in d-dimensional space are the closure of half-spaces under boolean set operations. In consequence, they can represent non-manifold situations, open and closed sets, mixed-dimensional complexes and they are closed under all boolean and topological operations.

We implemented a boundary representation of three-dimensional Nef polyhedra with efficient algorithms for boolean operations. These algorithms were designed for correctness and can handle all cases, in particular all *degeneracies*. The implementation is released as Open Source in the CGAL release 3.1.

In this paper, we present experiments in order to (i) evaluate the practical runtime complexity, (ii) illustrate the effectiveness of several important optimizations, and (iii) compare our implementation with the ACIS CAD kernel.

1 Introduction

Data structures for solids and algorithms for boolean operations on geometric models are among the fundamental problems in solid modeling, computer aided design, and computational geometry. We restrict ourselves to partitions of three space into cells induced by planes. A set of planes partition space into cells of various dimensions. Each cell may carry a label. We call such a partition together with the labelling of its cells a *Selective Nef Complex (SNC)*. When the labels are boolean($\{in, out\}$) the complex describes a set, a so-called *Nef polyhedron* [3]. Nef polyhedra can be obtained from halfspaces by boolean operations intersection and complement. Figure 1 shows a Nef polyhedron.

We gave a compact and unique representation of SNCs and algorithms realizing set operations based on this representation in [2]. The current implementation supports the construction of Nef polyhedra from manifold solids, boolean operations, topological operations, and rotations by rational rotation matrices. We follow the exact geometric computation paradigm [6] to achieve robustness.

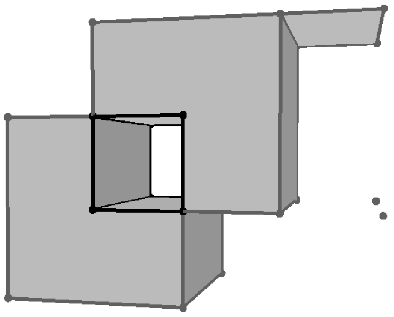


Figure 1: Nef polyhedron with non-manifold edges, a dangling facet, and two isolated vertices.

So far, we focused on the completeness of our algorithms. In this work, we evaluate their performance and present optimizations for important common cases. We evaluate their effectiveness individually and combined in a series of experiments.

Our current optimized implementation has become efficient enough to compare it with other systems. We selected ACIS R13, a common commercial CAD kernel used in many CAD systems [5].

2 Data Structures

Definition 1 (Nef polyhedron [3]) *A* Nef-polyhedron *in dimension d is a point set $P \subseteq \mathbb{R}^d$ generated from a finite number of open halfspaces by set complement and set intersection operations.*

Set union, difference and symmetric difference can be reduced to intersection and complement. Set complement changes between open and closed halfspaces, thus the topological operations *boundary*, *interior*, *exterior*, *closure*, and *regularization* are also in the modeling space of Nef polyhedra. In what follows, we refer to Nef polyhedra whenever we say polyhedra and we restrict ourselves to three dimensions.

In our representation for three-dimensional Nef-complexes, we use two main data structures:

Sphere Maps represent the local neighborhoods at each vertex. We conceptually intersect the local neighborhood of a vertex with a small ε-sphere. We obtain a planar map on the sphere (Figure 2), which

*Max-Planck Institut für Informatik, Saarbrücken, Germany, `[hachenberger|kettner]@mpi-sb.mpg.de`.

Partially supported by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces)

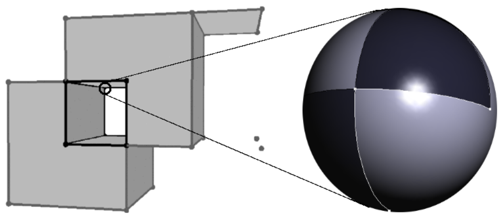


Figure 2: An example of a sphere map. The different colors indicate selected and unselected faces.

forms a two-dimensional Nef polyhedron embedded in the sphere. Sphere maps were introduced in [1].

The *Selective Nef Complex* provides a more easily accessible polyhedron representation. It additionally stores the connections between the local sphere maps. In detail it provides halfedges, facet cycles, halffacets, shells and volumes.

Additionally, we have a kd-tree associated with each Nef polyhedron. It provides fast point location and ray shooting needed in binary operations.

3 Binary Boolean Operations

Based on the SNC data structure, we can implement the binary boolean set operations. We find the sphere maps of all vertices of the resulting polyhedron and synthesize the SNC from there; in more detail:

1. Find possible candidate vertices. We take as candidates the original vertices of both input polyhedra, and we create all intersection points of edge-edge and edge-face intersections.

2. Given a candidate vertex, we find its local sphere map in each input polyhedron. If the candidate vertex is a vertex of one of the input polyhedra, its sphere map is already known. Otherwise a new sphere map is constructed on the fly.

3. Given the two sphere maps for a candidate vertex, we apply the boolean operation on sphere map to obtain the resulting sphere map.

Boolean operations on spheres are an extension of a planar sweep. Instead of a sweep line in the plane, the spheres are cut into two hemispheres and a great arc sweeps around each hemisphere.

4 Experiments

We experimentally evaluate the runtime behavior of our implementation, in particular the binary boolean operations. We have several experiments that support the expected runtime, and we have designed experiments to stress our implementation with worst-case situations. We report on a subset here.

The tests are performed on a 846 MHz Pentium III processor and 256MB RAM. We measure the total runtime and the runtime of the following subroutines:

1. **Point location:** queries kd-tree of an input polyhedron to locate a vertex of the other polyhedron.

2. **Box intersection:** intersection finding on the bounding boxes only, excludes the cost of the callback function, i.e., the intersection test on the actual edge and facet geometry.

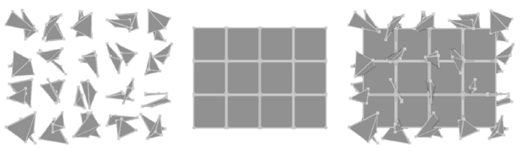
3. **Sphere sweeps:** sum of all sphere sweeps performed during boolean operations on sphere maps.

4. **Synthesizing edges:** in the synthesis step, sorts an edge representation based on Plücker coordinates.

5. **Plane sweeps:** in the synthesis step, sorts facet boundary cycles of the result polyhedron.

6. bf Kd-tree construction: in the synthesis step, initializes the kd-tree for the result polyhedron.

7. **Others:** all other parts not listed explicitly in the same graph, i.e. parts which have no critical worst-case or no interesting practical runtime.



Experiment TetGrid

1. Create a regular N^3 grid T of random tetrahedra, i.e., each tetrahedron is randomly generated in half-open fixed-size cubical area. Those areas form a regular N^3 grid.
2. Create a regular $(N-1)^3$ grid C of cubes.
3. Align T and C such that the grid nodes of C are at the centers of the grid cells of T.
4. Unite T and C. Measure time for experiment.

In our first test series, we want to examine the generic runtime behavior if the two input objects and the output object all have similar size. We capture these properties in the TetGrid experiment and measure its runtime for values $N = 3, \ldots, 17$.

In Figure 3 we see the runtime distributed over the main subroutines. The plane sweeps, the kd-tree construction and the point location comprise a major part of the total runtime.

Two further experiments are performed in order to find out about the generic runtime behavior of the binary operations. In the first, we combine two equally sized polyhedra such that a quadratic sized polyhedron results. The runtime of this experiment is mostly determined by the kd-tree construction.

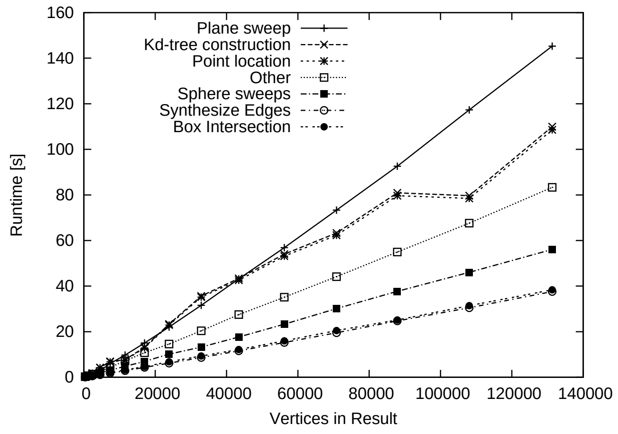


Figure 3: Runtime of the main subroutines in the TetGrid experiment.

In the other experiment, we subtract a simple polyhedron from a complex polyhedron. Here, the point location is essentially responsible for the runtime.

Furthermore, we perform several experiments in order to stress the worst case runtime behavior of the main subroutines. By this we confirm the theoretically evaluated complexity of the sphere sweep, plane sweep, point location and the ray shooting subroutine.

5 Optimizations of the Sphere Overlays

We have seen in the previous section that certain subroutines of the algorithm are very dominant. We implemented several optimizations that prevent the execution of some complex subroutines for many common and easy cases. Here, the optimization of the sphere overlays was most effective.

The sweep-line algorithm is a powerful tool; We use it in the plane for facet boundary cycles and we use it on half-spheres for the sphere maps. However, it is a comparatively costly step, although its asymptotic complexity is close to optimal.

optimizations			number of	runtime	
(i)	(ii)	(iii)	sweeps	sweeps	total
-	-	-	240470	345.12	480.34
+	-	-	14858	36.41	189.56
-	+	-	201227	301.02	437.17
-	-	+	217484	335.84	478.56
+	+	+	12880	27.67	163.65

We evaluate the contribution of the sphere sweeps to the total runtime of a binary boolean operation with a TetGrid experiment for $N = 16$. The table above lists the number of sphere sweeps performed during this operation, together with the runtime of the overlay and the total runtime. The values in the first row refer to a test run without any optimizations. The other rows refer to test runs with one or more of the following optimizations activated:

(i) The sphere sweep algorithm is used only in complex cases. In most cases, the overlay is computed by specialized algorithms.

(ii) Our sphere sweep can process one half-sphere at once. A lot of extra work has to be done to cut each sphere map into two halves and to paste the two resulting half-spheres back together. We try to get by with sweeping only one half-sphere, if possible.

(iii) For some vertices of the input polyhedra it is easy to determine that they will not appear in the resulting polyhedron. For instance, in a union operation every vertex of either polyhedron located in the inside of the other polyhedron is absorbed into this volume. We do not perform an overlay on these vertices.

6 Comparison with ACIS R13

We compare our implementation with Acis R13, a common commercial CAD kernel used in many CAD systems [5]. It should be said that we are comparing apples with oranges here. Acis is handling more general geometries and has some overhead in dispatching function calls to the specialized functions for linear geometry. On the other hand, our implementation handles Nef polyhedra in their full generality with all the potentially occurring degeneracies in the algorithms and it uses exact arithmetic to be reliable and robust.

6.1 Balanced Binary Operations

To get a general impression, we repeat the TetGrid experiment with Acis. Naturally, both algorithms perform on the same data sets.

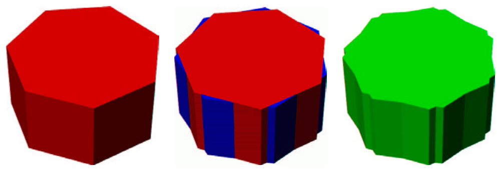
N	result vertices	runtime [s]	
		Acis R13	Nef 3D
3	352	0.29	1.01
4	1135	0.63	3.67
5	2390	1.37	8.43
6	4548	2.79	17.30
7	7383	5.29	29.20
8	11555	10.13	44.29
9	16998	14.27	70.26
10	23883	22.81	102.09
12	43418	35.58	192.12
14	70827	swapping	316.71

The table above shows that Acis is faster by a factor of 4 to 6. No obvious trend is visible.

We get quite a different result, if we subtract a simple object from a complex object. Here, the difference between Acis and our algorithm is quite pronounced with Acis being a factor of about twenty times faster. A notable difference might be in the software interface; Acis modifies the first input object to become the result, while our implementation creates the result from scratch.

6.2 Floating-Point versus Exact Arithmetic

One of the major differences between ACIS and our implementation is our use of exact arithmetic instead of floating point-arithmetic. Floating-point and interval arithmetic are the state-of-the-art in Computer Aided Design, but we are not aware of any system that uses exact arithmetic to solve the remaining cases that floating-point and interval arithmetic cannot solve. An obvious reason is the runtime cost for exact arithmetic, but also the difficulties in realizing exact and efficient solutions for more general curves and surfaces may play a role.



Experiment ROTCYLINDER

1. Create two identical right cylinder C and C', such that the base is a regular polygon with N sides parallel to the x-y plane.
2. Rotate C' around its vertical centerline by α.
3. Unite C and C'. Measure time for experiment.

We use the ROTCYLINDER experiment to demonstrate the effect of exact arithmetic; on one hand, we gain expressiveness in modeling, because we can compute results where ACIS fails very soon, and on the other hand, it shows the runtime cost for exact arithmetic, since the input coordinates grow in this series of experiments.

In this test scenario the endpoints of the intersection edges are extremely close together. Without an adequate precision it is not possible to compute an intersection point that is on both edges and different from the endpoints.

n	α	runtime ACIS R13	runtime Nef 3D
100	10^{-1}	1.08s	4.65s
	10^{-2}	1.05s	4.77s
	10^{-3}	1.08s	4.85s
	10^{-4}	1.07s	4.90s
	10^{-5}	not executable	5.03s
1000	10^{-1}	61s	93s
	10^{-2}	61s	95s
	10^{-3}	61s	97s
	10^{-4}	not executable	98s
2000	10^{-1}	252s	274s
	10^{-2}	253s	280s
	10^{-3}	255s	288s
	10^{-4}	not executable	290s
10000	10^{-7}	not executable	4433 s

The table above shows that ACIS' floating-point operations are insufficient for α smaller than 10^{-3}. On the other side, ACIS is faster, in particular for small instances. For $n = 100$ the factor of our runtime and ACIS' runtime is slightly below 5; for $n = 2000$ it is less than 1.2. Additionally, we performed a run with $n = 10000$, $\alpha = 10^{-7}$ to highlight the robustness of our arithmetic operations.

7 Conclusion

We achieved our goal of a complete, exact, correct and efficient implementation of boolean operations on a very general class of polyhedra in space. Useful extensions with applications in exact motion planning are Minkowski sums and the subdivision of the solid into simpler shapes, e.g., a trapezoidal or convex decomposition in space.

For ease of exposition, we restricted the discussion to boolean flags. Larger label sets can be treated analogously.

Nef complexes are defined by planes. It follows from the work on the *Selective Geometric Complexes* (SGC) of Rossignac and O'Connor [4] that our data structures extend immediately to complexes defined by curved surfaces. However, some of the algorithmic steps become difficult and need further work, such as the synthesis algorithm, where we need unique representations of intersection curves and where we need to sort points on intersection curves.

References

[1] K. Dobrindt, K. Mehlhorn, and M. Yvinec. A complete and efficient algorithm for the intersection of a general and a convex polyhedron. In *Proc. 3rd Work. Alg. Data Struct.*, LNCS 709, pages 314–324, 1993.

[2] M. Granados, P. Hachenberger, S. Hert, L. Kettner, K. Mehlhorn, and M. Seel. Boolean operations on 3D selective Nef complexes: Data structure, algorithms, and implementation. In *Proc. 11th Annu. Europ. Sympos. Algorithms (ESA'03)*, LNCS 2832, pages 654–666. Springer, 2003.

[3] W. Nef. *Beiträge zur Theorie der Polyeder.* Herbert Lang, Bern, 1978.

[4] J. R. Rossignac and M. A. O'Connor. SGC: A dimension-independent model for pointsets with internal structures and incomplete boundaries. In M. Wozny, J. Turner, and K. Preiss, ed., *Geom. Model. for Prod. Eng.* North-Holland, 1989.

[5] A Dassault Systèmes company Spatial Corp. *ACIS R13 Online Help*, 2004.

[6] C. Yap. Towards exact geometric computation. *Comput. Geom. Theory Appl.*, 7(1):3–23, 1997.