Representation and Computation of Boolean Combinations of Sculptured Models *

Shankar Krishnan
krishnas@cs.unc.edu

Atul Narkhede
narkhede@cs.unc.edu

Dinesh Manocha
manocha@cs.unc.edu

Department of Computer Science,
University of North Carolina,
Chapel Hill, NC 27599-3175

Abstract

We outline an algorithm and implementation of a system that computes Boolean combinations of *sculptured* solids. We represent the surface of the solids in terms of trimmed and untrimmed spline surfaces and a connectivity graph. Based on algorithms for trapezoidation of polygons, partitioning of polygons, surface intersection and ray-shooting, we compute the boundaries of the resulting solids after the Boolean operation.

1 Introduction

The field of solid modeling deals with design and representation of physical objects. The two major representation schemata used in solid modeling are constructive solid geometry (CSG) and boundary representations (B-rep). Both these representations have different inherent strengths and weaknesses and for most applications both these representations are desired. At the same time, the field of geometric modeling has been developed to model classes of piecewise surfaces based on particular conditions of shape and smoothness. Such models are called *sculptured* solids. There is considerable interest in building complete solid representations from spline surfaces and their Boolean combinations [Hof89]. However, the major bottlenecks are in performing robust, efficient and accurate Boolean operations on the sculptured models. The topology of a surface patch becomes quite complicated when Boolean operations are performed and finding a convenient representation for these topologies has been a major challenge. As a result, most of the current solid modelers use polyhedral approximations to these surfaces and apply existing algorithms to design and manipulate these polyhedral objects. Not only does it lead to data proliferation, but the resulting algorithms are inefficient and inaccurate.

We outline a system for representation and efficient computation of Boolean operations on CSG models. Every CSG object is built from a set of primitive objects which are of a simpler structure, and we assume that each of these primitives is an *oriented solid* representable as a collection of parametric surface patches. This class of models includes all polyhedral models, and primitive solids like cones, spheres, cylinders and tori. Apart from the primitive objects, each intermediate and final solid in the CSG hierarchy is represented as a collection of *trimmed* surface patches. Along with the surface definition, every *trimmed* patch contains a *trimming* curve on its domain which unambiguously determines which part of the surface is to be retained. We also create a *connectivity* graph that maintains the connectivity information between the various surfaces composing the solid. The entire algorithm proceeds in two steps. Initially, the complete intersection curve between the two solids (at each level of the CSG tree) is determined. If the set operation on the two solids is well-defined, the intersection curve partitions the surface of each solid. The second step of the algorithm uses the graph structure of each solid, and depending on the set operation, computes the new solid and its associated graph structure. The overall algorithm makes use of a number of recently developed geometric algorithms for linear programming [Sei90], trapezoidation of polygons [Sei91], partition of polygonal domains and ray-shooting.

2 Geometric Algorithms used in the System

The CSG algorithm is based on a number of geometric operations. These include trapezoidation of simple polygons, partitioning a simple polygon using non-intersecting polygonal chains, computing the intersection of two planar polygons and finding the intersection curve of two parametric surfaces. Some of the algorithms used are not necessarily the most optimal in terms of time complexity. In such cases, we have traded implementation simplicity for efficiency. This section briefly discusses these algorithms.

We use Seidel's algorithm [Sei91] to construct the horizontal decomposition of a simple polygon. This enables us to answer point location queries in $O(\log n)$ time and find one point inside the polygon in constant time. It is an incremental randomized algorithm whose expected complexity is $O(n \log^* n)$. In practice, it is almost linear time for a simple polygon having n vertices.

A frequently occuring problem in a single CSG operation involves trimming out a parametric surface (surface patch) using a set of intersection curves. This reduces to partitioning a simple polygon with a collection of non-intersecting polygonal chains. An $O(n \log^2 n)$ algorithm

*Supported in part by Sloan Foundation, university research council grant, NSF grant CCR-9319957, ONR contract N00014-94-1-0738, ARPA contract DABT63-93-C-0048, NSF/ARPA Science and Technology Center for Computer Graphics & Scientific Visualization and NSF Prime contract No. 8920219

is used for this purpose, where n is the total number of vertices of the polygon and the chains. We find all the intersections between the chain segments and the polygon edges and label them in the order they occur with respect to the chain and the polygon. This is followed by a stack-based traversal of the polygon boundary and partitions are generated when proper combinations of labelled intersections are found [KNM94].

Computing the intersection curve between two parametrically defined surfaces forms an important ingredient of the CSG algorithm. We use the algorithm in [KM94] which computes all the components of the intersection curve between a pair of patches in piecewise linear form. The accuracy of the intersection curve is user-controlled.

Moreover, we use tests based on bounding boxes and linear programming to prune out non–intersecting surfaces.

3 Boolean Operations between Solids

In this section, we briefly highlight the important steps in computing the result of Boolean operation of two solids (A and B). Initially, we compute the intersection curve between all pairs of intersecting spline–surfaces. The intersection algorithm finds the intersection curve between a pair of non–trimmed patches. However, if the patches are trimmed, we need to retain only those portions of the intersection curve which are common to trimming regions of both the patches. This is accomplished by marching along the piecewise linear chain in each domain and performing in–out tests all along. The resulting curve partitions the surface of each solid into components. Each such partition is tested for membership in the final solid by performing ray–shooting queries. We make use of connectivity graphs to minimize the number of ray–surface intersection queries. Membership decisions are based on the following rules:

- *Union:* A component of A is part of the new solid if it lies **outside** B. A component of B is part of the new solid if it lies **outside** A.
- *Intersection:* A component of A is part of the new solid if it lies **inside** B. A component of B is part of the new solid if it lies **inside** A.
- *Difference:* A component of A is part of the new solid if it lies **outside** B. A component of B is part of the new solid if it lies **inside** A.

4 Implementation and Performance

The system has been implemented on a high-end SGI-Onyx workstation with a single processor configuration. The system consists of two main parts: the surface intersection code and the supporting geometric routines for CSG.

The input to the intersection code are two parametric patches, and the output is a piecewise linear approximation of their intersection curve. The accuracy of this output directly depends on the *stepsize* and *tolerance* used by the numerical tracing algorithm. Our implementation uses EISPACK routines (in Fortran) for various matrix computations. The intersection code is invoked at every level in the CSG tree. We found that the choice of stepsize was critical in preventing excessive accumulation of floating point error and data proliferation while performing the set operations on multi-level CSG trees. Large errors could result in inconsistentcies in the topology of the final solid. In our implementation, we use conservative stepsizes (0.02–0.05) to circumvent this problem.

Geometric algorithms implemented on fixed precision arithmetic introduce different types of computational errors. In general, performing robust and efficient computations on non–linear models is an open problem. Therefore different comparison tests have to be made using tolerances. Due to the wide range of input values encountered, setting a fixed tolerance is not sufficient. To reduce this problem, our system normalizes the patches (scaled down) before applying the intersection algorithm. Degenerate cases are handled as special cases by reducing the problem to intersection of polygons in $2D$. The detection of such degenerate overlaps is sensitive to tolerance values, and larger tolerance values seem to work better in these cases.

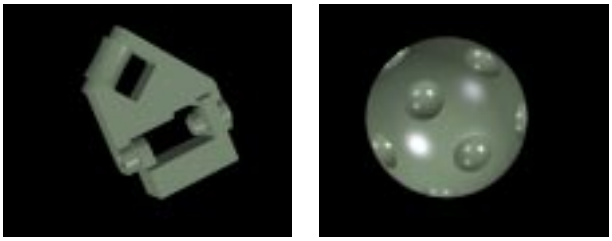

Figure 1: (a) Roller model (b) Cratered sphere

Fig. 1 shows two of the models generated by our system.

- *Roller model*: The complete roller model is generated from a 30 level CSG tree. It consists of 137 trimmed patches (some of whose degree is as high as 6×2). The total time for model generation is close to 3.5 minutes.
- *Cratered sphere*: This model is generated from a 15 level CSG tree composed of difference operations between a big sphere and a number of smaller spheres. The resulting solid consists of 96 trimmed patches each of degree 2×2 and took 135 seconds to generate.

The complete details of the algorithm and its performance are presented in [KNM94].

Acknowledgements

We thank Electric Boat for letting us use the CSG representation of the roller model and the UNC Walkthru group for their support.

References

[Hof89] C.M. Hoffmann. *Geometric and Solid Modeling.* Morgan Kaufmann, San Mateo, California, 1989.

[KM94] S. Krishnan and D. Manocha. An efficient surface intersection algorithm based on the lower dimensional formulation. Technical Report TR94-062, Department of Computer Science, University of North Carolina, 1994.

[KNM94] S. Krishnan, A. Narkhede, and D. Manocha. Boole: A system to compute boolean combinations of sculptured solids. Technical Report TR95-008, Department of Computer Science, University of North Carolina, 1994.

[Sei90] R. Seidel. Linear programming and convex hulls made easy. In *Proc. 6th Ann. ACM Conf. on Computational Geometry*, pages 211–215, Berkeley, California, 1990.

[Sei91] R. Seidel. A simple and fast randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry Theory & Applications*, 1(1):51–64, 1991.