Degree-Driven Algorithm Design for Computing Volumes of CSG Models

David L. Millman Jack Snoeyink
Department of Computer Science, University of North Carolina Chapel Hill *

Abstract

We describe a framework for computing volumes of CSG models. The framework was designed using Liotta, Preparata, and Tamassia's degree-driven algorithm design paradigm, which suggests that the algorithm designer balances minimizing the time, space, and precision used by an algorithm. The framework serves as an example of how degree-driven algorithm design can be used in practice to produce a fast and accurate implementation.

1 Introduction

Consider the query on a multi-component constructive solid geometry (CSG) model: *What is the volume enclosed by each component within some tolerance?*

Let the modeling primitives be point sets satisfying the inequality

$$f(x,y,z) \leq A_1x^2 + A_2y^2 + A_3z^2 + A_4xy + A_5xz + A_6yz + A_7x + A_8y + A_9z + A_{10}.$$

Each component C_i is defined by a formula F_i of regularized set operations on primitives. Moreover, all components have disjoint interiors, and are contained in a bounding box B. We can test if a point q is in C_i by plugging the coordinates of q into the inequalities defining C_i and evaluating F_i.

Multi-component CSG models are used by simulation codes, such as MC21 [5], that solve the neutron transport equation. Volumes are computed with a Monte Carlo algorithm that samples N points inside B, counts the number of points h_i landing in each C_i, and approximates the volume of each C_i as $\texttt{Vol}(B) * h_i/N$. In practice, N is very large, over 1.4 billion samples for a 10^{-4} relative error, causing this algorithm to be slow.

With co-authors, in [4], we proposed a framework for computing volumes that was over 500x faster and two orders of magnitude more accurate than the Monte Carlo algorithm. Liotta, Preparata, and Tamassia's degree-driven algorithm design paradigm [3] guided many of our decisions. While the decisions were not discussed in [4], we believe that they were important because the framework is an example of how one can, in practice, use degree-driven design to arrive at a fast and accurate implementation even without sophisticated software libraries.

* [dave,snoeyink]@cs.unc.edu

2 Precision of Computing Volumes

When designing a geometric algorithm, it is common to assume the Real-RAM model of computation, which assumes that arithmetic operations are exact and take unit time. In this setting, the *predicates*, which control branching in an algorithm by deciding geometric relationship between objects, are exact.

When implementing a geometric algorithm, predicates are evaluated numerically. Often a computer trades a small amount of error in the numeric calculation so that it can be done quickly in hardware. The small errors can cause a predicate to decide an incorrect relationship between two objects. As a result, an algorithm may branch incorrectly, possible breaking the invariants used to prove the correctness of an algorithm.

Liotta, Preparata, and Tamassia [3] suggested that in addition to analyzing the time and space used by a geometric algorithm one could also analyze the precision as well. Their suggestion, called *degree-driven algorithm design*, was to analyze the precision of an algorithm by its predicates. More formally, consider a geometric problem where the input is defined by coordinates (perhaps with some combinatorial relationships), that can be scaled to w-bit integers. A *predicate*, tests the sign of a multivariate polynomial whose variables are from the input coordinates. We define the *degree of a predicate* as the degree of its corresponding polynomial and the *degree of an algorithm* as the highest degree of the predicates used by the algorithm.

Let's see an example where we wish to test if point q is inside a primitive s, defined in the previous section, which we call the $\texttt{isInside}(s,q)$ predicate. The 13 input coordinates are the x-, y-, and z-coordinates of q and the coefficients $s_1, \ldots, s_{10}$ defining s scaled to w-bit integers. The combinatorial relationships are that $q = (q_x, q_y, q_z)$ and $s = (s_1, \ldots, s_{10})$. To test if q is inside of s check the sign of the polynomial $P(s,q) = s_1x^2 + s_2y^2 + \ldots + s_{10}$. Observe that P is degree 3 and therefore `isInside` is degree 3. Moreover, any algorithm using `isInside` is at least degree 3. Sometimes the degree d of a polynomial is more important than the polynomial itself. In such a case, we write $\textcircled{d}$, for example $P(s,q) = \textcircled{3}$.

To begin, let's consider algorithms for computing volumes. First, consider the Monte Carlo algorithm described in the previous section. As its only predicate is `isInside`, which is degree 3, the algorithm is degree 3. To our knowledge, this is the lowest degree algorithm, but, as mentioned earlier, many samples are needed for an accurate volume.

Second, consider constructing the boundary representation of each C_i by extending Berberich *et al.*[1] and using the surface patches to set up the integration domains. Having a boundary representation could achieve a very accurate volume calculation, but the precision required for building the boundary, in particular intersecting some (but not all) pairs of quadrics, is high.

The framework described in [4] explores an octree, computing in each cell the representation of the model restricted to the cell. Once the representation restricted to the cell is simple enough or the cell is small enough, the framework computes volumes. For example, it computes a portion of the boundary representation in cells containing edges formed by the intersection of a plane and a quadric, but uses Monte Carlo in small cells where many curved surfaces intersect. Exploring the octree depends on the predicate `Classify`, and the constructions `Simplify` and `Integrate` which we detail below.

The predicate `Classify`(s, b) takes an axis aligned box b, with degree 1 corners, and a primitive s, and returns if b is completely inside, outside, or intersects s. The implementation of this predicate, described in [4], determines the type of the conical intersection curve of a plane and a quadric. The conic can be written as

$$f(x,y) = \begin{pmatrix} x & y & 1 \end{pmatrix} \begin{pmatrix} ① & ① & ② \\ ① & ① & ② \\ ② & ② & ③ \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

The matrix in the above equation is called the *discriminant*. The suggestion of [4] was to determine the type of the conic with Levin's classification [2], which uses a high precision calculation of computing the eigenvalues of the discriminant. Since, we have observed that it is sufficient to check if the type of the conic is an ellipse, which simplifies to a degree 5 test of the sign of the determinant of the discriminant.

Theorem 1 *Given an axis aligned box b with degree 1 corners and a quadric s defined by degree 1 coefficients. The* `Classify`(s, b) *predicate returns the side of s containing b or if b intersects s in degree 5.*

Note that many quadrics have special forms, for example a plane or an axis aligned cylinder, in such cases, `Classify` factors and has lower degree.

The construction `Simplify`(C, b) takes a set of components C, and an axis aligned box b and computes the representation of each component restricted to b. For each C_i the construction: classifies each surface defining C_i with respect to b, replaces any surface in which b is inside or outside with a true or a false, and uses Boolean logic to simplify each formula. The only numerical test of `Simplify` is `Classify`, the rest is logic and data structuring.

Theorem 2 *Given an axis aligned bounding box b and a set of components C, the construction* `Simplify`(C, b) *returns the restriction of each component of C to b with degree 5.*

The construction `Integrate`(C, b, ϵ, δ) takes set of components C with representations restricted to box b, and error ϵ and confidence δ and returns the volume of the intersection of each component's intersection with b to error within ϵ and confidence δ or a flag if it cannot. Three example integrators are: the box integrator, which returns volumes when b is completely inside or outside of components; the cylinder bundle integrator, which returns volumes when the forumulæ restricted to b represent non-intersecting cylinders (which is common when the model contains bundles of pipes); and the Monte Carlo integrator mentioned earlier.

The framework we described in [4] keeps track of error bounds on computed volumes so that the specified tolerance is met. We have observed that the error bounds are too loose, resulting in unnecessary work. For example, when only three decimal places of accuracy are requested we get five decimal places correct. Currently, we are investigating how to better maintain error bounds from the integrators in order to produce volumes that are still accurate but not past the point of benefit.

References

[1] E. Berberich, M. Hemmer, L. Kettner, E. Schömer, and N. Wolpert. An exact, complete and efficient implementation for computing planar maps of quadric intersection curves. In *Proc. of the* 21^{st} *Annu. Symp. Comput. Geom.*, pages 99–106. ACM, 2005.

[2] J. Levin. A parametric algorithm for drawing pictures of solid objects composed of quadric surfaces. *Commun. ACM*, 19(10):555–563, 1976.

[3] G. Liotta, F. P. Preparata, and R. Tamassia. Robust proximity queries: An illustration of degree-driven algorithm design. *SIAM J. Comput.*, 28(3):864–889, 1999.

[4] D. L. Millman, D. P. Griesheimer, B. R. Nease, and J. Snoeyink. Robust volume calculations for constructive solid geometry (CSG) components in Monte Carlo transport calculations. In *PHYSOR: Advances in Reactor Physics*. Amer. Nucl. Soc., 2012. electron. proc.

[5] T. M. Sutton, T. J. Donovan, T. H. Trumbull, P. S. Dobreff, E. Caro, D. P. Griesheimer, L. J. Tyburski, D. C. Carpenter, and H. Joo. The MC21 Monte Carlo transport code. In *Joint international topical meeting M&C + SNA 2007*. Amer. Nucl. Soc., 2007. electron. proc.