Chapter 1

RELIABLE GEOMETRIC COMPUTATIONS WITH ALGEBRAIC PRIMITIVES AND PREDICATES

Mark Foskey
Dinesh Manocha
Tim Culver
John Keyser
and Shankar Krishnan

1. Introduction

The problem of accurate and robust implementation of geometric algorithms has received considerable attention for more than a decade. Despite much progress in computational geometry and geometric modeling, practical implementations of geometric algorithms are prone to error. Much of the difficulty arises from the fact that reasoning about geometry most naturally occurs in the domain of the real numbers, which can only be represented approximately on a digital computer. Many times, the correctness of geometric algorithms depends on correctly evaluating the signs of arithmetic expressions, and errors due to rounding or imprecise inputs can lead to grossly incorrect results or failure to run to completion.

The proposed solutions to this problem can be classified into *inexact* and *exact* approaches [29]. The former approach accepts the inaccuracy of the machine representation, and attempts to modify the algorithms, given that constraint, so that they reliably produce acceptable output. The notion of acceptability is dependent on the application. Algorithms developed in this way have been shown to work in specific cases. On the other hand, *exact geometric computation* (EGC) requires that every predicate evaluation be correct [35]. The exact computation paradigm eliminates numerical error in geometric computations entirely. Unfortunately, exact implementations are often far too slow, especially when we are dealing with nonlinear primitives. Karasick et al. [22] noted that naive implementations can take several orders of magnitude longer than an equivalent floating-point implementation, an observation that is consis-

tent with our experience. The goal has been to find techniques that reduce the performance penalty to an acceptable level.

As work on exact geometric computation has proceeded, it has become clear that the performance problems can be greatly alleviated. One area that has received less attention is the issue of reliability when dealing with nonlinear algebraic or curved primitives. This area provides numerous interesting challenges for EGC and we address some of them in this paper.

Exact Computation as a Practical Approach. There is no question that EGC is slower than computation relying solely on machine precision arithmetic. The question is whether the slowdown is worth the gain in precision. Indeed, in many scientific or engineering applications the input data is inexact, and the question arises whether an exact result is even meaningful. But the main reason for using EGC is not exactness in itself, but rather *reliability*. A common cause of program failure is that rounding errors lead to inconsistent combinatorial decisions, e.g. about where a point lies with regard to a surface. By making a single interpretation of the data and performing calculations that are consistent with that interpretation, we can avoid this source of failure. Solving the problems of accuracy and consistency is the first step towards a general solution to the robustness problem, which also involves handling degeneracies and special cases.

Organization. The rest of this paper is organized as follows. In Section 2 we briefly review the relevant literature. In Section 3 we present some underlying geometric problems involving curved primitives. We discuss our general approach to EGC for curved primitives in Section 4, including methods we use to achieve reasonable speeds for these computations. In Section 5 we present some results for boundary evaluation and Voronoi computations, and we conclude in Section 6.

2. Literature Survey

The issue of robust and accurate computations in geometric applications has been addressed in numerous places, with surveys by Hoffmann [19] and Fortune [16] giving an indication of the variety of work. Yap [34] described the concept of exact geometric computation, and Li and Yap [28] have presented a more recent survey. Some of the earlier inexact approaches were based on geometric tolerances [32] and interval arithmetic [10].

One of the key ideas in accelerating exact computation is the use of *floating point filters*, in which predicates are first evaluated using fast floating point methods and then tested for reliability, by analyzing the size of the possible floating point error. If a predicate is unreliable, exact computation is performed. A good example of this method is the work of Fortune and van Wyk [17]. As

a preprocess, they perform an analysis of the calculations that will be needed by a geometric algorithm, so that the accuracy of these computations can be checked quickly at run time.

A number of researchers have used exact computation for boundary evaluation. Benouamer, Michelucci, and Peroche [2] implement a solid modeler using a filtered approach that differs from that of Fortune and van Wyk. Benouamer et al. express each sequence of calculations as an *expression dag*, that is, a directed acyclic graph with operations at internal nodes and constants at the leaves. Calculations are initially performed using interval arithmetic, and if the result is not sufficiently precise, then exact rational arithmetic is used.

Fortune [15] also used exact arithmetic to implement a polyhedral solid modeler. That work sets an upper limit on the bit-length of accepted input, so that all geometric predicates can be evaluated using arithmetic at some fixed precision.

Yap and Dubé [35] introduced a general approach they call "precision-driven computation." Like Benouamer et al., they also use expression dags, but as a tool to determine in advance the amount of precision needed (that is, the number of digits in a floating point representation). Precision-driven computation is noteworthy because it is fundamentally distinct from filtered approaches. However, it is only applicable for closed-form calculations.

Boundary evaluation in solid modeling has been a well studied research topic in the area of polyhedral models. In addition to the results on the subject mentioned above, we note work of Hoffmann [20] and Requicha and Voelcker [31]. Some algorithms have also been proposed for quadrics or higher degree algebraic primitives. Casale et al. [4] use trimmed parametric surfaces to generate boundary representations of sculptured solids. Their algorithm uses subdivision methods to evaluate surface intersections, and represents the trimming boundary with piecewise linear segments. Krishnan and Manocha presented algorithms and a system called BOOLE based on the algebraic formulation of the problem [26]. BOOLE is based on lower dimensional algorithms for computing the intersections of parametric surfaces and uses a combination of symbolic and numeric algorithms [25]. It uses 64-bit IEEE floating point arithmetic.

One area where reliability is particularly challenging and has received relatively little study is computation of the medial axis of a polyhedron. There have been a number of approximate approaches, however. Vleugels and Overmars [33] and Etzion and Rappoport [14] both use recursive subdivision of space to create an arbitrarily close approximation, while Hoff et al. [21] compute an approximation at a fixed resolution using graphics hardware. Other authors do not rely on an approximation. Of these, Milenkovic [30] was the first to propose an algorithm for computing the medial axis as a 3D geometric object by tracing the seams between the curved faces of the structure.

3. Nonlinear Geometric Problems

In this section we discuss a number of problems involving curved geometric primitives that arise in geometric applications.

Polynomial Root Isolation. The isolation of complex roots of polynomials is in a sense a geometric problem in the complex plane. Also, the other problems we discuss will depend in an essential way on localizing the real roots of polynomials.

Curve Arrangements. Given a number of algebraic curves in a bounded region of the plane, the goal in the curve arrangement problem is to compute the connected subregions that have no curve passing through them. Each subregion, called a *face*, is defined by piecewise algebraic curves that enclose its boundary. For each of the polynomials defining a face, all points in the face will have the same sign with regard to that polynomial. The output of the curve arrangement algorithm is the explicit topological description of each cell.

Boundary Evaluation. In computational solid geometry (CSG) [20], objects are constructed from solid primitives by the boolean operations of union, intersection, and set difference. A complicated object will be represented as a tree, with geometric primitives at the leaves, and boolean operations at the internal nodes. The *boundary evaluation problem* is the problem of taking a CSG model and constructing from it a representation of its boundary as a set of possibly curved two dimensional surface primitives with adjacency information.

The Medial Axis Transformation. The medial axis of a polyhedron is the locus of points that are the centers of spheres contained in the polyhedron and touching the boundary at two or more points. In 3 dimensions, the medial axis is made up of portions of quadric surfaces intersecting along curves. All of the known practical algorithms for computing the exact medial axis—explicitly constructing all of its surfaces and curves—rely on tracing these curves, starting at the vertices of the polyhedron [5]. Combinatorial errors at early stages can cause incorrect curves to be generated, typically resulting in a program failure. The probability for such a failure increases rapidly with the complexity of the polyhedron.

4. Exact Geometric Computation for Curves and Surfaces

The fundamental challenge of EGC in the curvilinear domain is that the coordinates of points determined by intersecting polynomial surfaces will typically be irrational and thus not representable exactly by a rational package. We represent such a point by retaining the polynomials defining it, along with an

axis-aligned box with rational coordinates known to contain only that intersection. There are mechanisms to shrink the box when necessary, to isolate the roots and make comparisons between the points. As a result, the algorithm uses minimal precision to accurately perform the tests. This technique is an example of the distinction between exact arithmetic and exact computation—the only explicit representation of a point is inexact, but all comparisons are made exactly. The library MAPC [24] has been designed to embody these representations for points and curves in two dimensions.

Methods used to isolate points in 2D and 3D are detailed in [23] and [5] respectively. We will briefly discuss them here to indicate the kind of calculations needed. The fundamental tool we use for reliably localizing algebraic points is the *Sturm sequence*. For a univariate polynomial f, the Sturm sequence of f is the polynomial remainder sequence of f and f', with the signs changed according to a simple convention. For any given real number x, the number of *sign permanencies* $\mathrm{PERM}(f, x)$ is the number of times the sign remains the same when successive polynomials in the Sturm sequence are evaluated at x. For two real numbers $x < y$, $\mathrm{PERM}(f, y) - \mathrm{PERM}(f, x)$ is the number of real roots of f between x and y.

There are generalizations of the Sturm sequence concept for sets of polynomials in two and three dimensions, but they are much slower. In the computation of the medial axis, there are times when a point must be localized in 3D using the trivariate Sturm sequence [5]. However, in most cases it is possible to localize a 2D point using only univariate Sturm sequences [23]. The idea is to use an alternate method to find candidate boxes that may contain one or more points, and then to reduce the problem to a sequence of root determinations on the boundaries of these boxes.

4.1 Improving Efficiency

To ensure accuracy, Sturm sequence calculations are usually done with exact arithmetic. Since bit lengths arising in a Sturm sequence calculation can be exponential in the degree of the polynomial, these calculations can be quite slow. To improve running times, we have made extensive use of floating point filters. Unfortunately, there is a large gap between the 53 mantissa bits of a machine double on current hardware and the hundreds of bits that can arise in a Sturm sequence calculation. It is useful to have an arithmetic that is faster than rational arithmetic but that can be flexible in the amount of precision it allows. Aberth and Schaefer [1] have proposed a solution to this problem in what they call *range arithmetic*, which combines conservative interval arithmetic with variable precision floating point computation. Each number in their arithmetic is represented by a single floating point number of arbitrary length, with an associated single-precision radius R. The radius R indicates the width of the

associated interval, on the scale of the least significant machine word in the representation of the floating point number.

To guarantee the precision of their results, Aberth and Schaefer perform a calculation at a specified initial precision, keeping track of the loss of accuracy. If the accuracy of the result is insufficient, they increase the precision of their representation and recompute the results with increased precision. We find that this approach of *iterative revision* is useful for many geometric problems, where precision-driven computation using expression dags may not be effective.

5. Some Results

In this section, we give examples of how the techniques we have described can be applied to a number of problems.

5.1 Polynomial Root Isolation

To isolate complex roots of polynomials, we have used Aberth and Schaefer's Range library [1] to apply a variant, described in [27], of the Durand-Kerner algorithm [11, 9] to a number of polynomials. A similar algorithm has also been proposed by Bini [3]. We tested this algorithm on a benchmark set of polynomials from the PoSSo project, available at the following site:

```
http://www-sop.inria.fr/saga/POL
```

Each class of polynomials is known to be troublesome for many root finding algorithms. We compared our results with two other packages, Maple and MuPAD. The results are given in Table 1.1. Maple and MuPAD allow the user to specify the number of digits retained throughout a calculation, but not the (smaller) number of digits that will be reliable in the output. For the purposes of comparison, we specified that calculations be performed with the same number of digits as the maximum used in our calculations with the Range library. Because the Range library performs later calculations with fewer digits than the maximum required, it has a speed advantage when intermediate calculations must be performed at a much higher precision than needed in the final result. The particular root-isolation method used also contributes to the speedup. There is, however, an overhead in maintaining the error interval, which becomes more apparent when less precision is needed. These results were previously reported in [27].

We used the following polynomials:

- **Poly1:** $\Sigma_{i=0}^{n} \frac{x^i}{i!}, n = 50.$
- **Poly2:** $(x - 3c^2)^2 + icx^7, 0 < c \ll 1, c = 10^{-20}.$
- **Poly3:** $(c^2x^2 - 3)^2 + c^2x^9, c = 10^{20}.$

Case	Root Precision	MuPAD	Maple	Range
Poly1	10	78.77	4.79	13.17
Poly2	120	5.15	41.54	13.06
Poly3	80	3.32	8.539	6.32
Poly4	30	4.569	0.792	0.82
Poly5	30	32.21	36.31	5.67
Poly6	30	14.01	34.31	0.65
Poly7	30	6.517	13.53	1.23
Poly8	30	12.21	18.717	5.26
Poly9	30	71.215	1.26	1.44

Table 1.1. Univariate root finding algorithm applied to nine polynomials from the PoSSo benchmark suite. The second column indicates the number of required significant digits, specified in advance. The last three columns indicate running times taken by Maple, MuPAD and our algorithm. Times are in seconds and measured on an SGI Origin 400 MHz R12000 processor running Irix6.5.

- **Poly4:** $x^{20} + cx^{14} + x^5 + 1$, $c = 10^{12}$.
- **Poly5:** $\prod_{i=1}^{n}(x - i), n = 40$.
- **Poly6:** $(0.01x^{10} + (x - 10)^2)\prod_{i=1}^{20}(x - i)$.
- **Poly7:** $\prod_{i=1}^{20}(x - i)(x - 20)^2$.
- **Poly8:** $x^{14} + 2cx^{11} + c^2x^8 + 4x^7 - 4cx^4 + 4$, $c = 10^{24}$.
- **Poly9:** $x^n - a, n = 50, a = 1$.

5.2 Determinant Sign

The problem of efficiently computing the sign of the determinant of large rational matrices has not been extensively studied, but there are situations where it can be useful [7]. One straightforward approach, using the Range library, is simply to use Gaussian elimination with partial pivoting, repeating the process with progressively higher precision until the sign can be determined reliably. Figure 1.1 indicates that this method often performs much better than exact methods that use modular arithmetic, when applied to nonsingular matrices. For general matrices, Culver et al. [6] proposed a filter for computing determinant signs exactly. The singular value decomposition, computed at machine precision, can be used to indicate whether the matrix is likely to be singular, what the likely sign of the determinant is, and whether the matrix is well-conditioned enough to ensure that the sign estimate is correct. If the sign estimate is correct, it is used. If the matrix is ill-conditioned and likely to be singular, modular

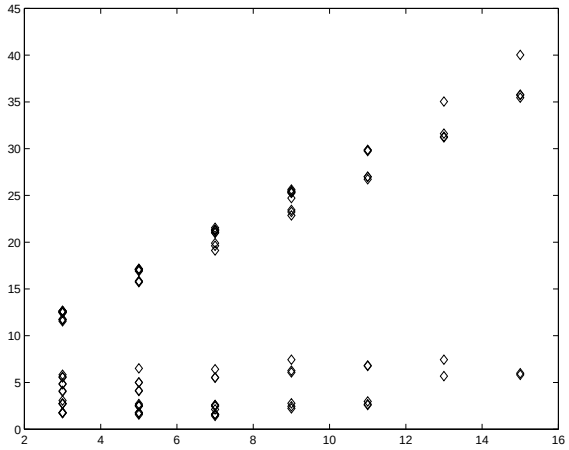

Figure 1.1. **Determinant sign speedup.** The speedup factor for the Range library, for various matrices. The horizontal axis gives the order of the matrix, while the vertical axis gives the speedup factor in comparison to an exact modular arithmetic algorithm. The set of matrices includes randomly generated matrices and others arising in geometric problems. The matrices are all nonsingular.

arithmetic is used. Otherwise, Gaussian elimination using the Range library is used.

5.3 MAPC

The MAPC library [24], mentioned in Section 4, provides tools for exact manipulation of algebraic points and curves. It uses exact arithmetic based on LiDIA [18] to accurately compute the required Sturm sequences. For high degree polynomials, exact computation of Sturm sequences can be prohibitively slow because of the very large growth in the bit lengths of the coefficients. Some boundary evaluation computations on algebraic primitives can require Sturm sequence computation for polynomials of degree greater than 80.

MAPC uses a number of techniques to deal with this problem. Of key importance is reducing the number of Sturm sequence computations that must be performed. For instance, floating point methods can be used to estimate the locations of polynomial roots, which are then confirmed by a Sturm sequence computation. This is much more efficient than repeated bisection using Sturm sequences.

When a Sturm sequence computation is necessary, Aberth and Schaefer's Range library [1] is used. As with the sign of determinant calculations discussed above, the process of generating the Sturm sequence and computing polynomial signs is performed at successively increasing precision until all the signs can be

Case	1	2	3	4
Number of Curves	3	3	6	7
Coefficient Bit size	25	22	25	62
Number of Faces	9	11	31	63
Total time without Range	5.2	9.0	62.8	122.8
Total time with Range	1.8	4.0	11.0	9.3

Table 1.2. Arrangement of planar algebraic curves. The application finds all subregions, the segments of curves bounding each subregion, and the connectivity between subregions. The table shows the maximum bit length needed to express the coefficients of the curves, the number of faces generated by the arrangement, the time taken using the original (exact rational based) code, and the time taken using Range. The curves have maximum degree 4. Times are in seconds on a 300 MHz R12000 MIPS processor.

evaluated exactly. Once a fixed precision is reached without success (e.g. 40 decimal digits), exact rationals are used instead.

We have used MAPC as a tool in approaching each of the following three geometric problems. We give performance information for each one, with attention to the benefits of arbitrary-precision error bounded arithmetic. Again, there is a speed advantage that grows with the complexity of the problem, but a small overhead that becomes apparent with simpler calculations.

Curve Arrangements. The computation of curve arrangements is a useful test case for the MAPC library. In Table 1.2 we indicate performance results for some example arrangement computations.

Boundary Generation. A motivating application for the MAPC library is the boundary evaluation of (low degree) algebraic solids. MAPC is a core library in the ESOLID system [23] performing such boundary evaluations. In general, computing the boundary representation for a CSG model leads to problems of accuracy and robustness. These problems are exacerbated when the underlying primitives have curved boundaries. To alleviate the reliability problems, the ESOLID system performs performs all geometric tests exactly, using layered filters to make the exact computation more efficient.

We have tested ESOLID on portions of a real-world model, the Bradley Fighting Vehicle provided courtesy of the Army Research Laboratory. Some example output B-reps are shown in Figure 1.2, with comparative timings given in Table 1.3.

Figure 1.3 gives an example of a calculation that can be hard for machine-precision methods. The intersection curve between the two cylinders is not self-crossing, although it appears so at the scale shown. This distinction can be hard for non-exact methods to resolve.

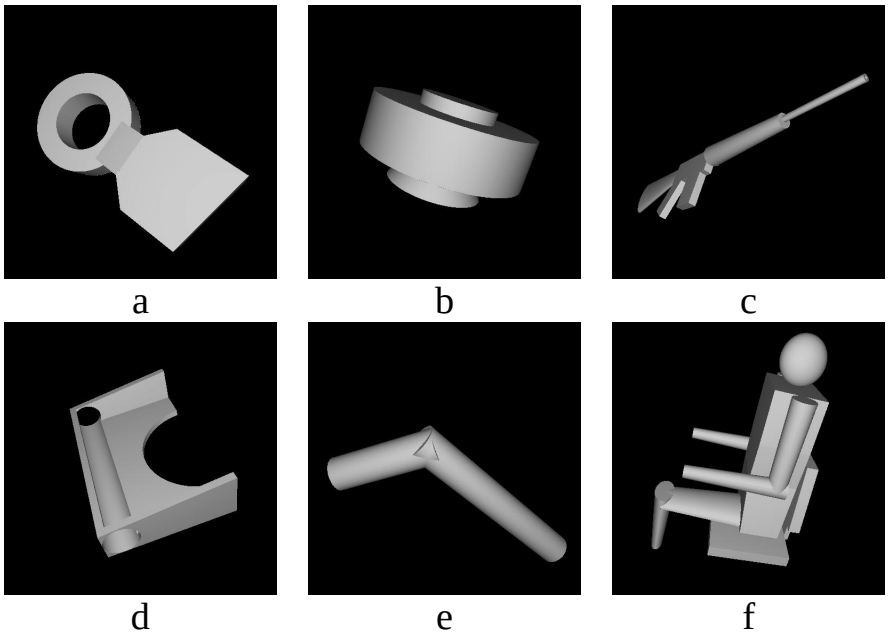

Figure 1.2. **Boundary computations.** Boundary representations of six selected portions of the Bradley Fighting Vehicle, computed by ESOLID. Computations were done exactly, and then output as trimmed NURBS patches for rendering. Computation times ranged from about 10 seconds to 633 seconds; further performance details are given in Table 1.3.

Example Number	without Range Total Time	without Range Sturm Time	with Range Total Time	with Range Sturm Time
a	10.23	0.51	10.95	1.62
b	12.57	0.24	12.69	1.44
c	633.42	597.33	42.99	6.93
d	63.15	8.34	61.26	6.36
e	250.74	190.62	73.86	15.36
f	26.37	1.29	28.14	3.63

Table 1.3. Timings for the examples from Figure 1.2, with and without the incorporation of the Range library. Range is used to improve the efficiency of Sturm sequence calculations. The total time and the time spent in Sturm computations is shown.

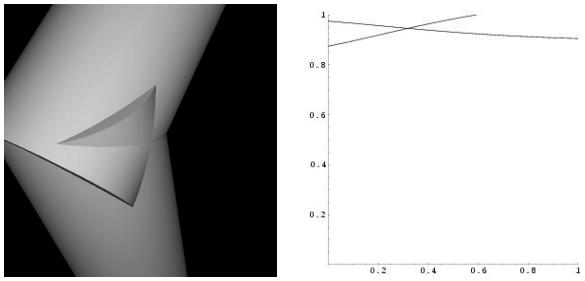

Figure 1.3. An example of a case that can be difficult for machine-precision methods. The plot on the right shows a portion of the intersection curve of the two cylinders, in the parametric domain of one of the cylinders. The curve is very nearly singular, but in fact has two distinct components.

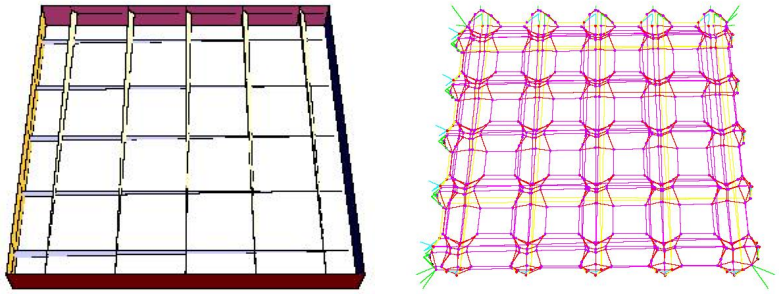

Figure 1.4. The "iron maiden pizza box" and a schematic of its medial axis. The top and bottom of the box are removed to show the spikes inside. The model has 56 faces, and the computation took 23 minutes.

Medial Axis. Computing the medial axis of a polyhedron is a challenging problem because it inherently requires analysis of intersecting curved surfaces. Culver et al. [8] have implemented an exact algorithm for medial axis evaluation that relies both on the MAPC library and on the sign of determinant filter described in Section 5.2, both of which incorporate the Range library. The program has been used to compute the medial axis of complicated polyhedra with as many as 250 faces [7]. Examples of the output are given in Figures 1.4 and 1.5. In both examples, seams are depicted as straight lines.

6. Conclusions and Future Work

We have found that exact geometric computation can be a practical tool to alleviate reliability problems in geometric computations with algebraic curves

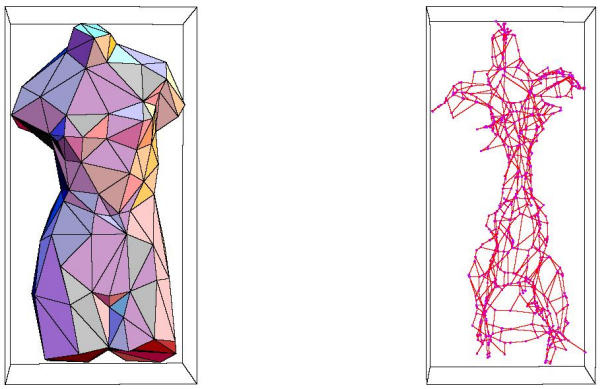

Figure 1.5. The Venus de Milo and a schematic of its medial axis. Seams touching vertices of the polyhedron are omitted for clarity. The polyhedron in this example has 250 faces, and took 5.6 hours to compute.

and surfaces. Some of the initial results related to root finding, curve arrangements, boundary and Voronoi computations are promising and there are many areas for future research.

One important problem is that of dealing with *degeneracies*, such as the intersection of a line with a polygon only at one vertex, or along an edge. Degeneracies can be a source of non-robustness on the one hand, or of serious implementation difficulties on the other. For simplicity, algorithms often assume that primitives are arranged so that there are no degeneracies (i.e., they are in *general position*). In practice, however, primitives often are not in general position, causing implementations of the algorithms to fail. Recasting an algorithm to handle degeneracies tends to result in a situation in which most of the code is to handle special cases. Edelsbrunner and Mücke have presented a nice overview of the problem [12].

A number of authors have proposed the idea of *symbolic perturbation* to solve these problems [12, 13]. Unfortunately, these algorithms depend on the existence of a decision tree in which each decision rests on the evaluation of an expression that is some known polynomial in the input values. In the non-linear problems we have discussed, the calculation cannot be formulated in this way. A general approach to the problem of degeneracies, perhaps in the spirit of symbolic perturbation, would be a significant contribution.

Finally, another goal is to perform reliable computations with higher degree primitives (e.g., bicubic rational parametric patches that are widely used in geometric modeling). Currently, we have handled primitives of algebraic degree four for boundary evaluation and the medial axis computation results in quadric surfaces.

7. Acknowledgments

We would like to thank the BRL-CAD group at the Army Research Lab for the use of the Bradley Fighting Vehicle model. This work was supported in part by an ARO Contract DAAD19-99-1-0162, NSF award 9876914, DOE ASCI grant and ONR Young Investigator Award.

References

[1] Oliver Aberth and Mark J. Schaefer. Precise computation using range arithmetic, via C++. *ACM Transaction on Mathematical Software*, 18(4):481–491, 1992.

[2] M. O. Benouamer, D. Michelucci, and B. Peroche. Error-free boundary evaluation based on a lazy rational arithmetic: A detailed implementation. *Computer Aided Design*, 26(6):403–416, June 1994.

[3] D. Bini. Numerical computation of polynomial zeros by means of Aberth's method. *Numerical Mathematics*, 13:179–200, 1996.

[4] M. S. Casale and J. E. Bobrow. A set operation algorithm for sculptured solids modeled with trimmed patches. *Computer Aided Geometric Design*, 6:235–247, 1989.

[5] T. Culver, J. Keyser, and D. Manocha. Accurate computation of the medial axis of a polyhedron. In *Proc. Symposium on Solid Modeling and Applications*, pages 179–190, 1999.

[6] T. Culver, J. Keyser, D. Manocha, and S. Krishnan. A hybrid approach for evaluating signs of moderately sized matrices. Technical Report TR00-020, Department of Computer Science, University of North Carolina, 2000.

[7] Tim Culver. *Computing the Medial Axis of a Polyhedron Reliably and Efficiently*. Ph.D. thesis, University of North Carolina at Chapel Hill, Chapel Hill, North Carolina, USA, 2000.

[8] Tim Culver, John Keyser, and Dinesh Manocha. Accurate computation of the medial axis of a polyhedron. In *Proc. Symposium on Solid Modeling and Applications*, pages 179–190, 1999.

[9] K. Dochev. *Physical and Mathematical Journal of the Bulgarian Academy of Sciences*, 5:136–139, 1962.

[10] Tom Duff. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 131–138, July 1992.

[11] E. Durand. Solutions numeriques des equations algebriques. *Tome I, Masson, Paris*, 1960.

[12] Herbert Edelsbrunner and Ernst Peter Mucke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990.

[13] Ioannis Z. Emiris and John F. Canny. A general approach to removing degeneracies. In *Proceedings of the 32nd IEEE Symposium on the Foundations of Computer Science*, pages 405–413, 1994.

[14] Michal Etzion and Ari Rappoport. Computing the Voronoi diagram of a 3-D polyhedron by separate computation of its symbolic and geometric parts. In *Proc. Symposium on Solid Modeling and Applications*, pages 167–178, 1999.

[15] S. Fortune. Polyhedral modeling with exact arithmetic. *Proc. Symposium on Solid Modeling and Applications*, pages 225–234, 1995.

[16] S. Fortune. Robustness issues in geometric algorithms. In M. C. Lin and D. Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes Comput. Sci.*, pages 9–14. Springer-Verlag, 1996.

[17] Steven Fortune and Christopher J. van Wyk. Static analysis yields efficient exact integer arithmetic for computational geometry. *ACM Transactions on Graphics*, 15(3):223–248, July 1996.

[18] LiDIA Group. A library for computational number theory. Technical report, TH Darmstadt, Fachbereich Informatik, Institut fur Theoretische Informatik, 1997.

[19] C. M. Hoffmann. The problems of accuracy and robustness in geometric computation. *IEEE Computer*, 22(3):31–41, March 1989.

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

[21] Kenneth E. Hoff III, Tim Culver, John Keyser, Ming Lin, and Dinesh Manocha. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Computer Graphics Annual Conference Series (SIGGRAPH '99)*, pages 277–286, 1999.

[22] M. Karasick, D. Lieber, and L. R. Nackman. Efficient Delaunay triangulations using rational arithmetic. *ACM Trans. Graph.*, 10(1):71–91, January 1991.

[23] J. Keyser, S. Krishnan, and D. Manocha. Efficient and accurate B-rep generation of low degree sculptured solids using exact arithmetic I: Representations and II: Computation. *Computer Aided Geometric Design*, 16(9):841–882, October 1999.

[24] John Keyser, Tim Culver, Dinesh Manocha, and Shankar Krishnan. MAPC: A library for efficient and exact manipulation of algebraic points

and curves. In *Proc. 15th Annual ACM Symposium on Computational Geometry*, pages 360–369, 1999.

[25] S. Krishnan and D. Manocha. An efficient surface intersection algorithm based on the lower dimensional formulation. *ACM Transactions on Graphics*, 16(1):74–106, 1997.

[26] S. Krishnan, D. Manocha, M. Gopi, T. Culver, and J. Keyser. BOOLE: A boundary evaluation system for boolean combinations of sculptured solids. *International Journal of Computational Geometry and Applications*, 11(1):105–144, 2001.

[27] Shankar Krishnan, Mark Foskey, John Keyser, Tim Culver, and Dinesh Manocha. PRECISE: Efficient multiprecision evaluation of algebraic roots and predicates for reliable geometric computation. Technical Report TR00-008, University of North Carolina-Chapel Hill, 2000.

[28] C. Li and C. Yap. A new constructive root bound for algebraic expressions. In *Proceedings of the Symposium on Discrete Algorithms*, 2001.

[29] Chen Li and Chee Yap. Recent progress in exact geometric computation. http://www.cs.nyu.edu/exact/doc/dimacs.ps.gz, 2001.

[30] V. Milenkovic. Robust construction of the Voronoi diagram of a polyhedron. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 473–478, 1993.

[31] A.A.G. Requicha and H.B. Voelcker. Boolean operations in solid modeling: boundary evaluation and merging algorithms. *Proceedings of the IEEE*, 73(1), 1985.

[32] M. Segal. Using tolerances to guarantee valid polyhedral modeling results. In *Proceedings of ACM Siggraph*, pages 105–114, 1990.

[33] Jules Vleugels and Mark Overmars. Approximating generalized Voronoi diagrams in any dimension. Technical Report UU-CS-1995-14, Department of Computer Science, Utrecht University, 1995.

[34] C. K. Yap. Towards exact geometric computation. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 405–419, 1993.

[35] Chee Yap and Thomas Dubé. The exact computation paradigm. In D. Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry*, pages 452–492. World Scientific Press, Singapore, 1995.