Efficient Predicate Evaluation using Statistical Degeneracy Detection

Victor Milenkovic[*] Elisha Sacks[†]

Abstract

Computational geometry algorithms branch on the signs of predicates. Evaluating degenerate (zero sign) predicates is costly. Degeneracy is common for predicates whose arguments have common antecedents. Prior degeneracy detection techniques are slow, especially on predicates that involve algebraic numbers. We present statistical degeneracy detection (SDD) algorithms. Rational predicates are evaluated modulo randomly selected primes. Algebraic predicates are evaluated on randomly perturbed inputs. We analyze the failure rates under statistical assumptions. The algorithms are incorporated into an exact geometric computation library. Extensive testing shows that the library is reliable and fast. We also give an algorithm that reduces algebraic degeneracy detection to rational degeneracy detection without perturbation. This algorithm is much slower than the perturbation algorithm yet is far faster than prior work even when rational predicates are evaluated deterministically.

1 Introduction

We present research on the implementation of computational geometry algorithms. Implementations employ floating point arithmetic, whereas algorithms are expressed using real arithmetic. Although floating point is very accurate, even a tiny numerical error can cause a logical operator to return an incorrect Boolean value, which can cause a program crash or an unbounded error in the output. We follow the exact geometric computation (EGC) [25] strategy of ensuring accurate output by implementing logical operators that are correct despite numerical error.

A CG algorithm takes as input points or other geometric objects with rational parameters (coordinates or coefficients), expressed as ratios of integers or floating point numbers. The algorithm branches on the signs, > 0, $= 0$, or < 0 of polynomial functions of parameters, called predicates. For example, the sign of

$$\text{turn}(a,b,c) = (a-b)\times(b-c) \text{ with } u\times v = u_x v_y - u_y v_x$$

[*]Department of Computer Science, University of Miami, Coral Gables, FL 33124-4245, USA, vjm@cs.miami.edu. Supported by NSF CCF-1526335.

[†]Computer Science Department, Purdue University, West Lafayette, IN 47907-2066, USA, eps@purdue.edu. Supported by NSF CCF-1524455.

determine if abc turns left, goes straight, or turns right at b. Algorithms also generate new parameters using rational functions on (antecedent) parameters or the zeros of polynomials whose coefficients are rational functions of parameters. The former is rational, such as the coordinates of the intersection p of lines ab and cd, and the latter is algebraic, such as the intersections of the circle through b with center a with the line cd. Parameters and the predicates on them are rational if all functions in their derivation are rational; otherwise, they are algebraic.

In general, it is inexpensive to determine the sign of a nonzero predicate. For example, double precision floating point interval arithmetic usually results in an interval that does not contain zero, and hence the sign is known. Occasionally, additional precision is needed, such as using the MPFR library (`mpfr.org`). However, we find the additional cost is modest, up to 20%.

The situation for degenerate (zero) predicates is much direr. For rational predicates, it is necessary to use exact rational arithmetic using a library such as GMP (`gmplib.org`), and this can be much more expensive than double precision interval arithmetic. The general technique for algebraic predicates is root separation bounds, and these are very pessimistic, requiring many bits of precision. Exact rational arithmetic can sometimes be practical, but root separation bounds are almost never practical.

Degeneracy resulting from input in special position, such as collinear a, b, c can be eliminated by input perturbation: adding a small random quantity to each input parameter [12]. However, the cost of rational arithmetic for input degeneracies is not too high, and special position rarely results in algebraic degeneracy.

Derived parameters that are related by shared antecedents can also cause predicates to be degenerate. For example, consider line segments a_1b_1, a_2b_2, a_3b_3, and cd whose endpoint coordinates are input parameters. If cd intersects the other segments at p_1, p_2, and p_3, these points are collinear, $\text{turn}(p_1, p_2, p_3) = 0$, and this degeneracy is impervious to input perturbation. The coordinates of c and d are common antecedents of p_1, p_2, and p_3 in a manner that makes $\text{turn}(p_1, p_2, p_3)$ identically zero as a rational function of the coordinates of the eight input points. We call this type of degenerate predicate an *identity* because it is identically zero on an open set in the input parameter space, whereas a special position is zero on a measure zero set.

For a derivation depth of one or perhaps two, it is possible to analyze the identities and detect them by logic. For example, we know the p_1, p_2, p_3 in the above example will be collinear without exact rational evaluation. However, when multiple operations are cascaded, the number of types of identities rises exponentially with the derivation depth, and logic analysis becomes impractical. Unfortunately, the bit-complexity hence the cost of exact rational arithmetic or root separation also grows much greater.

We believe this effect often prevents the practical use of CG. Individual operations are efficient, but identities cause multiple consecutive operations to be very inefficient.

1.1 Contribution

We propose statistical degeneracy detection (SDD) to detect degenerate predicates without using exact rational arithmetic or separation bounds. We present three SDD algorithms and provide a library implementation at `https://github.com/Robust-Geometric-Computation`. Each algorithm outputs an estimated failure (false degeneracy) probability based on statistical assumptions. This approach allows efficient EGC on cascaded geometric operations. The estimated probabilities can be set so low as to make failure impossible in practice.

We provide the first probabilistic algorithms for degeneracy detection for both the rational and algebraic case, and we introduce the concept of statistical degeneracy detection. Prior work [10] uses a statistical assumption but does not provide an estimate of the probability of failure.

The first algorithm (Sec. 2) detects degenerate rational predicates by evaluating ambiguous (interval arithmetic interval contains zero) predicates modulo k random primes. We prove a worst-case bound on the probability of failure. However, this *probabilistic* bound requires having a bound b on the bit-complexity, but cancellation (of common factors of the numerators and denominators) greatly and unpredictably reduces b. Even given b, the probability bound is very pessimistic. We estimate the probability using a statistical assumption: nonzero predicates are zero modulo a random prime at the same rate as all nonzero expressions. In our tests, the estimated failure probability is always negligible for $k = 2$.

The second algorithm (Sec. 3) uses polynomial quotient rings to reduce an algebraic predicate to multiple rational predicates without the use of exact arithmetic or root separation bounds. The rational predicates can be evaluated using the first algorithm. It is much more efficient than root-separation-based methods but is limited to a small number of arguments.

The third algorithm (Sec. 4) uses the observation that identities remain zero after input perturbation, but all other expressions are likely to change their values. It perturbs its input to eliminate input (special position) degeneracies with high probability. It applies a second perturbation (provisionally) to predicates that remain ambiguous at h bits of precision. If an expression remains ambiguous, it reports an identity. It uses a measure on nonzero, nonidentity expressions to report a probability of failure, under the statistical assumption that this measure is the same for nonzero predicates. In our tests, the estimated failure probability is always negligible for $h = 265$.

1.2 Prior work

EGC comprises exact geometry kernels and number types. An exact geometry kernel supports a set of predicates for a class of objects, possibly with a limited capacity for defining new objects and predicates. The canonical examples are the CGAL [8] and Leda [15] kernels for rational operations on points. CGAL also provides an algebraic kernel for zeros of univariate polynomials. Two zeros can be ordered, but other predicates involving zeros are not supported. An exact number type supports a set of operations on a subset of the real numbers. The canonical example is the Leda real type for general expressions involving rational operations, radicals, and zeros of polynomials [7]. A number type is more flexible and easier to use than a kernel. But a kernel can model entire objects, rather than their parameters, and can exploit domain-specific algorithms.

Geometry kernels and number types are built from a common set of tools: interval arithmetic, arbitrary size integer arithmetic, arbitrary precision floating point arithmetic, and separation bounds.

Interval arithmetic [19] uses floating point arithmetic to compute an interval of floating point numbers that contains the value of an expression. EGC uses interval arithmetic in floating point filtering [5]: the sign of a predicate is determined when its interval excludes zero.

Arbitrary size integer arithmetic libraries, such as GMP (`gmplib.org`), are used to evaluate rational predicates. The complexity of a b-bit operation is $O(b \log b)$. The bit complexity of a rational predicate can be exponential in its derivation depth but is much lower in practice because common factors are canceled. Adaptive precision evaluation [22] is faster than GMP style arithmetic but is restricted to predicates in input parameters.

Rational predicates can also be evaluated using modular arithmetic. A predicate of bit complexity b requires b/k k-bit moduli. Degeneracy can be determined by verifying that all the residues are zero, at a cost of $\mathrm{O}(b)$. Computing the sign requires Chinese remaindering at a worst-case cost of $\mathrm{O}(b^2)$ and an expected cost of $\mathrm{O}(b)$ [6]. This paper proposes a probabilistic algorithm, but it depends on on a lemma [10] (Lemma 3.1) that makes the statistical assumption that the value of a polynomial modulo an integer is uniformly distributed. Despite the log factor savings for modular arithmetic in theory, arbitrary size integer arithmetic is most used in practice.

Degenerate algebraic predicates can be detected using separation bounds. A separation bound β for a predicate e is a positive number such that $e \neq 0$ implies $|e| > \beta$. Li, Pion, and Yap [13] survey separation bound computation and Emiris, Mourrain, and Tsigaridas [9] present the state of the art. The LEDA exact number type [7] evaluates a predicate with interval arithmetic and increases the floating point precision until the interval excludes zero or the interval width is less than a separation bound. The separation bound technique is rarely practical because the bounds shrink rapidly as the number and degree of the algebraic numbers increase.

Some special cases are handled without separation bounds. Berberich et al [3] present an arrangement algorithm for plane algebraic curves using only symbolic methods. It includes univariate and bivariate polynomial support that is faster than the CGAL algebraic kernel. Masterjohn et al [14] present an arrangement algorithm that uses our fixed δ perturbation framework (below) to avoid degeneracies. Neither is subject to identities. Blomer [4] provides a probabilistic algorithm for rational expression whose leaves are roots of integers.

Halperin [12] pioneered input perturbation for preventing special position degeneracy. Each input parameter of an algorithm is perturbed by a value chosen uniformly in $[-\delta, \delta]$. The algorithm is run on the perturbed input. The δ is chosen so that floating point filtering succeeds with high probability. If every instance succeeds for a run of the algorithm, the output is correct for the perturbed input, hence is correct with backward error δ for the original input. Otherwise, the algorithm is rerun with a different perturbation. This approach does not address identities. Moreover, it can require values of δ that exceed the error bounds of applications.

We [18] developed a perturbation algorithm that uses a fixed, user-specified δ. We evaluate predicates using arbitrary precision interval arithmetic [11] and increase the precision until the interval excludes zero. We abort the algorithm when the precision reaches a threshold that is high enough that only identities reach it. When this happens, we devise ad hoc code for that identity.

We [17] present an identity detection algorithm for all predicates involving contacts between polyhedrons with four degrees of freedom. These predicates can be expressed as $g(r)$ where r if a zero of f, and both f and g are univariate polynomials whose coefficients are multivariate polynomials in the input parameters. The algorithm uses a precomputed table of all polynomials f and g, up to isomorphism, and their multivariate factorizations.

2 SDD algorithm for rational predicates

The probabilistic degeneracy detection algorithm for an ambiguous rational predicate evaluates it modulo a random 32-bit prime p. Each input parameter is converted from a double to the form $a \times 2^b$, with a and b integers, then this expression is evaluated modulo p. Modular addition and multiplication are 64-bit machine operations followed by evaluation modulo p. Modular division uses the extended Euclidean algorithm. If any divisor is divisible by p, the test is rerun with a new random prime. We repeat the test k times and declare the predicate degenerate if the residue is zero each time.

We bound the probability that the algorithm will report a false degeneracy for a predicate e whose bit complexity is bounded by b. Let n denote the number of 32-bit primes. At most $b/31$ primes divide e because they are larger than 2^{31}, so the probability that a given prime divides e is at most $t = b/(31n)$. The probability that k primes divide e is at most t^k and so a maximum failure probability of r is ensured by setting $k = \log r / \log t$. Unfortunately, there is no general method to determine a tight bound on b.

The SDD algorithm estimates the actual false degeneracy rate under the assumption that an ambiguous predicate and its subexpressions are equally likely to fail the test. To get this estimate, we set t equal to $1/k$ times the fraction of the unambiguous (hence nonzero) subexpressions that are zero modulo one of the k primes.

3 Extension to algebraic predicates using quotient rings

We extend rational degeneracy detection to algebraic predicates. Consider a predicate polynomial $e(x)$ evaluated at a simple zero r of $f(x)$, where e and f have rational coefficients. We compute $g = \gcd(e, f)$, using rational degeneracy detection to detect if the leading coefficient of a remainder is zero and hence the remainder has lower degree than the generic case. Since $f(r) = 0$, $x - r \mid f$ and so $e(r) = 0$ if and only if $g(r) = 0$. Let $h = f/g$. If $g(r) = 0$, $x - r \mid g$, so $x - r \nmid h$, since r is a simple zero of f, and $h(r) \neq 0$. If $h(r) = 0$, $x - r \mid h$, so $x - r \nmid g$ and $g(r) \neq 0$. Hence, either $g(r) = 0$ and $h(r) \neq 0$ or vice versa. We evaluate both expressions in interval arithmetic and increase the precision until one interval excludes zero. If $h(r) \neq 0$, $e(r)$ is degenerate. This technique detects an algebraic zero without the use of separation bounds or exact arithmetic.

We use quotient rings to extend this algorithm to predicates that have multiple algebraic parameters. Consider a predicate $e(x_1, x_2)$ evaluated on simple zeros r_1 and r_2 of $f_1(x_1)$ and $f_2(x_2)$. Let R_1 denote the quotient ring $\mathbb{Q}[x_1]/f_1$ of polynomials in x_1 modulo f_1. Convert e to $R_1[x_2]$ by expressing it as $e = \sum_k p_k(x_1) x_2^k$ then replacing each p_k by its remainder when divided by f_1. Apply the above algorithm to e and f_2, which is trivially in $R_1[x_2]$, with $g = \gcd(e, f_2)$ and $h = f_2/g$ in $R_1[x_2]$. Either $h(r_1, r_2) = 0$ or $g(r_1, r_2) = 0$, and $e(r_1, r_2)$ is degenerate if $h(r_1, r_2) \neq 0$. In the general case, $e(x_1, \ldots, x_m)$ is evaluated on simple zeros $r_1, \ldots, r_m$ of $f_1(x_1), \ldots, f_m(x_m)$. Let $R_0 = \mathbb{Q}$ and $R_i = R_{i-1}[x_i]/f_i(x_i)$ for $i > 0$. Convert e and f_m to elements of $R_{m-1}[x_m]$ and apply the above algorithm, with $e(r_1, \ldots, r_m)$ degenerate if $h(r_1, \ldots, r_m) \neq 0$.

Implementing this algorithm requires zero detection and multiplicative inverse computation in R_i. These use rational degeneracy detection (Sec. 2) for $R_0 = \mathbb{Q}$. For $i > 0$, an element $a(x_i) \in R_i$ is represented by the remainder of a when divided by f_i in $R_{i-1}[x_i]$. Zero leading coefficients are detected in R_{i-1}, and a is zero when all the coefficients are zero. The inverse of a is computed with the extended Euclidean algorithm on $R_{i-1}[x_i]$. We detect a zero divisor (that has no inverse) when $g = \gcd(a, f_i)$ has nonzero degree. In that case, we calculate $h = f_i/g$, determine whether g or h is nonzero on $r_1, \ldots, r_i$, replace f_i with the other, and restart the degeneracy detection algorithm for e.

The number of restarts is at most total degree of f_i minus total degree of r_i, $i = 1, \ldots, m$. To prove correctness, we need to show that one of g and h is nonzero on $r_1, \ldots, r_i$ and the other is zero. Consider $f_i(r_1, \ldots, r_{i-1})(x_i) \in \mathbb{R}[x_i]$ that is the current $f_i(x_i)$ with $r_1, \ldots, r_{i-1}$ substituted in its coefficients. The invariant is that $f_i(r_1, \ldots, r_{i-1})(x_i)$ is divisible by $(x - r_i)$ but not $(x - r_i)^2$. This is true for the initial f_i. If $f_i(x_i) = g(x_i)h(x_i)$, then $f_i(r_1, \ldots, r_{i-1})(x_i) = g(r_1, \ldots, r_{i-1})(x_i)h(r_1, \ldots, r_{i-1})(x_i)$, one factor is divisible by $x - r_i$, and neither factor is divisible by $(x - r_i)^2$ due to the uniqueness of factorization. The factor that is not divisible by $x - r_i$ is nonzero on r_i.

The algorithm is impractical for large values of m because it requires d^{2m} operations for $f_1, \ldots, f_m$ of total degree d. It is difficult to interface with computational geometry algorithms because predicates must be expressed as polynomials in the m algebraic parameters.

4 Perturbation-based SDD algorithm for algebraic predicates

The exponential complexity of the quotient-ring-based SDD algorithm leads us to prefer a perturbation-based algorithm. This algorithm cannot detect special position degeneracy, so we prevent it with an input perturbation. We classify an ambiguous predicate as an identity if it remains ambiguous when the precision of the interval arithmetic is increased and when it is evaluated on a second perturbed input. A nondegenerate predicate is unlikely to remain ambiguous in either case.

The user selects the perturbation size δ and the identity detection control parameter h with default values $\delta = 2^{-27} \approx 10^{-8}$ and $h = 212$. The former is chosen based on the accuracy needed for the application, and the latter can be increased if the SDD failure probability estimate is too high. The algorithm employs two internal parameters: the perturbation precision b and the secondary perturbation size s, with initial values $b = 26$ and $s = \delta$. Each input parameter is perturbed by a b-bit number that is uniformly distributed in $[-\delta, \delta]$. For the initial b and the default δ, perturbing an input is equivalent to randomizing the lower half of its mantissa. Smaller δ or larger b would require expressing each input as a sum of doubles.

The algorithm runs the geometric computation on the perturbed input p. If an algebraic predicate $e(p)$ is ambiguous in floating point interval arithmetic, it is reevaluated in h-bit interval arithmetic. If it is still ambiguous, the algorithm selects a second perturbation $q = p + rv$. Each coordinate of the vector v is drawn uniformly from the set of b-bit numbers in $[-1, 1]$. The scalar r is initialized to s then is divided by ten until the isolating intervals of the algebraic numbers, which were computed at p, are also isolating at q. The algorithm reports an identity when $e(q)$ is ambiguous in h-bit interval arithmetic. Otherwise, it computes the sign of $e(p)$ using l-bit interval arithmetic, starting with $l = 2h$ and doubling l until the interval excludes zero.

This algorithm requires that the initial b-bit perturbation eliminates all degenerate non-identity predicates; otherwise, l would increase without bound. In practice, we put an upper bound $m = 424$ on l to prevent an infinite loop. If that bound were ever reached, we would restart with a different perturbation. If the bound were reached yet again, we would double the default values of b and m.

The algorithm cannot assign a nonzero sign to an identity. We bound the probability of a false identity under the assumption that $e(t)$ is analytic on $[0, r]$. This assumption can be guaranteed when the algebraic numbers in e are zeros of polynomials whose coefficients are rational parameters: when applying the Descartes rule of signs, verify that these coefficients have constant signs on $[0, r]$. We know of no practical test for polynomials with algebraic coefficients. We discuss this issue further in Sec. 6.

Suppose $e(p)$ is reported as an identity. Since $e(p)$ and $e(q)$ are ambiguous, $\Delta = e(q) - e(p)$ is ambiguous. A sufficient condition for ambiguity is $|\Delta| < w$ with w the width of the interval value of Δ in h-bit interval arithmetic. We call $|\Delta|/w$ the perturbation ratio of e. We approximate $e(q)$ by its linear Taylor series $e(p) + rg \cdot v$ with g the gradient of e with respect to p. The approximate perturbation ratio is $r|g \cdot v|/w$. Its maximum for a predicate that depends on d input parameters is $m = r||g||\sqrt{d}/w$ because the components of v are bounded by one.

Lemma 1 *The probability that $r|g \cdot v|/w < 1$ is less than $2\sqrt{2d}/m$.*

Proof. If $|g \cdot v| < w/r$, v lies in a slab of $(-1, 1)^d$ of thickness at most $2w/(r||g||)$. The maximum area of a cross section is $\sqrt{2}$ by Ball's theorem. Hence, the volume of the slab is at most $2\sqrt{2}w/(r||g||) = 2\sqrt{2d}/m$. □

We estimate an upper bound on $2\sqrt{2d}/m$ using the maximum d and using the minimum perturbation ratio of all ambiguous predicates that are nondegenerate as an estimate for the lower bound of m. This false identity probability estimate neglects the truncation error of the linear Taylor series. One can estimate this error by comparing $e(p + rv)$ to $e(p + rv/2)$ and can control it by shrinking r.

5 Results

We tested SDD on eight computational geometry algorithms on polyhedrons. Table 1 lists the input to the tests, and subsequent tables list the results. The inputs are displayed in the papers cited below. We provide the software at `https://github.com/Robust-Geometric-Computation`.

5.1 Rational predicates

The first five tests use rational predicates. 1) We pack three polyhedrons into a box with an algorithm [2] that composes ten Minkowski sums and Boolean operations. 2) We compute a constrained Delaunay triangulation of a polyhedron with an algorithm [23] that places Steiner points on edges. 3) We apply the same triangulation algorithm to the Minkowski sum of two polyhedrons. 4) We compute a constrained Delaunay mesh of a polyhedron with an algorithm [24] that places Steiner points on edges and facets, and at the circumcenters of tetrahedrons. 5) We repeat a test of our algorithm [1] for approximating the free space of a four degree of freedom (4DOF) polyhedron that translates freely and rotates around its z axis.

The best prior degeneracy detection algorithm is exact rational evaluation with floating point filtering. Adaptive precision evaluation [22] is inapplicable because most of the predicates have derived parameters. For test 1, our prior work [21] eliminates all degeneracies by perturbing the vertices of the polyhedron output of each step. Topology changes are allowed. As indicated in Sec. 1.2, this approach is efficient yet lacks an error bound. For test 5, our prior work [1] uses a preliminary version of rational SDD.

Table 2 shows the test results using $k = 2$ primes. Columns p through c refer to SDD. The percentage of ambiguous predicates a ranges from 0% to 22% of which the degenerate percentage d is a large majority. The predicate evaluation time t is between 30% and 60% of the total CPU time c. At least 60% of t is for floating point interval arithmetic f, at most 25% is for modular arithmetic m, and at most 38% is for arbitrary precision interval arithmetic e. Using $k = 5$ primes increases m by median and maximum factors of 1.7 and 2.9. The next two columns compare SDD to exact evaluation of ambiguous predicates using GMP. The predicate evaluation time increases by a factor $\times t$ of up to 216 and the CPU time increases by a factor $\times c$ of up to 131. The last column lists the maximum bit complexity b of the predicates in the test: the total number of bits in the numerator after cancellation. This number is obtained as a byproduct of exact rational evaluation. It ranges from about a thousand to almost 2 million.

The degeneracy detection failure probability bound (Sec. 2) with $k = 2$ is at most 4×10^{-7} because b is at most 1810577 (for test 4b), and $n \approx 9.3 \times 10^7$ by the prime number theorem. For $k = 5$, the bound is 10^{-19}. We never see a zero residue for an unambiguous subexpression of an ambiguous predicate, and so the estimated failure rate is zero. We ran test 4b 250 times with k=1000. Each run had 44 million unambiguous subexpressions and 200,000 nonzero ambiguous predicates. The latter is derived as $p(a/100)(1-d/100) \approx 0.2$ million. The number of zero residues for each type was 6404 and 29. This implies a zero residue rate of 1 in 1.7 billion for both populations. Thus the rate for the former appears to be a good proxy for the latter. Furthermore, both are close to $1/q$ for a random 32-bit prime q, 1 in 3 billion, corresponding to a uniform distribution of residue values. In contrast, the provable bound for $k = 1$ and $b = 1810577$ is 1 in 1700. So high bit complexity has some effect, but not nearly as much as the bound indicates. Using the measured zero residue rate, the estimated degeneracy detection failure rate for $k = 2$ is 3×10^{-19}, similar to the provable bound for $k = 5$.

We can only derive the bound for $k = 5$ by evaluating in exact rational arithmetic to determine the bit-complexity b after cancellation. In contrast, the estimated rate requires negligible overhead to compute since the SDD algorithm already calculates the residues for the unambiguous subexpressions of ambiguous predicates. To estimate the probability that the geometric construction failed, the individual predicate probability $3 \cdot 10^{-19}$ should be multiplied times the number of nonzero ambiguous predicate, such as the 200,000 for test4b. The largest of these is 56 million for test 5b.

5.2 Algebraic predicates

The last three tests use both algebraic and rational predicates. 6) We repeat a test from our prior work on 4DOF motion planning [17]: sort 100000 angles at which four randomly generated pairs of robot and ob-

Table 1: Test inputs.

#	Algorithm	Shape 1	Facets	Shape 2	Facets	Shape 3	Facets
1a	packing	cube	12	glacier	32	sphere	760
1b	packing	glacier	32	glacier	32	glacier	32
1c	packing	glacier	32	sphere	760	sphere	760
1e	packing	sphere	760	sphere	760	sphere	760
2a	CDT	bull	12400				
2b	CDT	ear	32236				
2c	CDT	horse	39694				
3a	CDT of Minkowski sum	bull	12400	glacier	32		
3b	CDT of Minkowski sum	ear	32236	glacier	32		
3c	CDT of Minkowski sum	horse	39694	glacier	32		
4a	mesh	bull	12400				
4b	mesh	ear	32236				
4c	mesh	horse	39694				
5a	4DOF free space	frustum	12	tworooms	122		
5b	4DOF free space	plus	44	lattice-room	204		
6	4DOF rotations	not applicable					
7a	4DOF path	frustum	12	tworooms	122		
7b	4DOF path	plus	44	lattice-room	204		
8a	3DOF free space	r1	4	o1	736		
8b	3DOF free space	r1	4	o2	2640		
8c	3DOF free space	r1	4	o3	4628		
8d	3DOF free space	r2	14	o4	8068		

Table 2: Rational predicates: # test, p predicates in millions, a percent of p that are ambiguous, d percent of a that are degenerate, t predicate evaluation time in seconds, f, m, e percent of t for floating point, modular, and arbitrary precision arithmetic, c total CPU time in seconds, $\times t$ and $\times c$ multipliers of t and of c for exact evaluation, and b maximum bit complexity.

#	p	a	d	t	f	m	e	c	$\times t$	$\times c$	b
1a	38	9	97	11	80	13	7	30	10	4	5307
1b	35	9	93	11	72	15	13	23	13	7	13925
1c	49	9	76	19	60	19	21	49	4	2	3484
1d	902	8	79	274	66	18	16	666	6	3	9362
2a	3	1	98	0.6	75	25	0	1.8	1	1	1064
2b	6	0	98	1.2	96	4	0	4.0	1	1	942
2c	8	0	99	1.4	97	3	0	5.2	1	1	921
3a	29	2	77	11	63	9	28	37	10	4	4362
3b	54	1	51	25	53	9	38	74	5	2	4373
3c	109	1	69	33	70	7	23	113	6	3	4429
4a	45	3	92	12	83	6	11	35	4	2	600364
4b	194	2	95	51	90	4	6	167	38	12	1810577
4c	90	1	88	23	90	3	7	90	1	1	65680
5a	1113	5	91	252	79	10	11	420	216	131	292091
5b	1026	22	75	704	60	23	17	1114	100	63	115587

stacle features can have simultaneous contacts. 7) We generate a path in the test 5 approximate free space. 8) We mesh the free space of a 3DOF polyhedron that translates and rotates in a plane, which we compute with our prior algorithm [20].

In test 6, the robot and obstacle features are each generated from a pool of 12 vertices with random coordinates. Four robot/obstacle feature pairs define a rational angle polynomial. Angle parameters s and t are zeros of angle polynomials f and g and yield unit vectors

$$u = \left(\frac{1-s^2}{1+s^2}, \frac{2s}{1+s^2}\right) \qquad \text{and} \qquad v = \left(\frac{1-t^2}{1+t^2}, \frac{2t}{1+t^2}\right).$$

Table 3: Algebraic predicates: # test, p predicates in millions, a percent of p that are ambiguous, l and i percent of a that are algebraic and identities, t predicate evaluation time in seconds, f, m, e percent of t for floating point, modular, and arbitrary precision arithmetic, c total CPU time in seconds, and 10^{-r} error probability.

#	p	a	l	i	t	f	m	e	c	r
6a	35	0.2	96	96	51	5	2	93	57	*
6b	35	0.2	96	96	46	5	1	94	53	*
7a	0.2	13	45	1	1	8	13	79	1	28
7b	0.1	5	21	0	0	9	12	79	0	36
8a	1	5	48	37	1	33	14	53	2	20
8b	4	7	49	20	2	34	16	50	6	30
8c	11	6	44	22	6	35	14	51	15	16
8d	105	5	26	10	40	42	10	48	93	18

Vectors with $\text{sign}(u_y) = \text{sign}(v_y)$ (equivalent to $\text{sign}(s) = \text{sign}(t)$) are ordered by $\text{sign}(u \times v)$ (in 2D $u \times v = u_x v_y - u_y v_x$). Test 6a tests for identity by determining if $f(t) = 0$. It is also required to check that $s' - t \neq 0$ for every zero $s' \neq s$ of g. Test 6b checks if $u \times v = 0$ directly. In test 7, the path consists of rotations plus shortest paths on the polyhedron boundary of the approximate free space for a fixed angle. In test 8, we approximate the boundary patches with triangles that conform at patch boundaries, compute the arrangement of the triangles, and return the union of the cells with positive winding numbers.

There is no practical prior general degeneracy detection algorithm. The only option is separation bounds and these are impossibly small in every test. For test 6, our prior degeneracy detection algorithm is a table lookup of the factorizations of f and g, which is much faster than SDD but requires much specialized work to categorize all the angle polynomials for the domain. For test 7, our prior work prevents degeneracy by repeated geometric rounding [16], which is extremely slow (Sec. 1.2).

Table 3 shows the test results. We set $h = 265$ in the perturbation algorithm. We obtained this value by setting $h = 106$ (two times double precision) and increasing it by increments of 53 until the error estimate became tiny for tests 7 and 8. The error cannot be estimated in test 6 because every ambiguous predicate is an identity. The predicate values are the same for $h = 212$ and $h = 265$, which shows that the error estimate is conservative. Empirically, ambiguous at h bits is equivalent to identity, so algebraic predicates never require more than h bits and there are no restarts. As noted for the rational case, the error rate should be multiplied by the number of nonzero ambiguous expressions to obtain the probability that the computation failed. The error rate is very conservative, since this rate is zero.

Table 4 compares the perturbation-based (Sec. 4) and quotient-ring-based (Sec. 3) algorithms on predicates from tests 6 and 8. Test 7 is unsuitable for the residue method because the predicates required to construct a shortest path traversing m faces of a polyhedron are degree m in m square roots of rational expressions. The test 6 predicates are a) $f(t)$ or b) $(1-s^2)(2t)-(2s)(1-t^2)$ where s and t are zeros of f and g, respectively. The test 8 predicates are low-degree polynomials in three or four coordinates of points. The coordinates of a point are rational expressions in a zero of a polynomial of degree 2 or 4. The residue algorithm always returns the same result as the perturbation algorithm, which provides further evidence that the latter is correct. We compare the running times on predicates where floating point filtering fails, hence degeneracy detection is required. The residue algorithm with rational SDD ($k = 2$) is faster than the perturbation algorithm on predicates with $n = 1$ algebraic numbers but is 4 times slower with $n = 2$, 400 times slower with $n = 3$, and 1150 times slower with $n = 4$. The residue algorithm using exact rational evaluation for ambiguous predicates is 2 to 24 times slower than using SDD.

Table 4: Predicate evaluation: # test, n algebraic arguments, p predicates, t perturbation algorithm time in microseconds, $\times p$ and $\times e$ multipliers for residue algorithm with SDD and exact evaluation.

#	n	p	t	$\times p$	$\times e$
6a	1	84000	560	0.2	2.4
6b	2	84000	509	3.5	84
8	3	32500	25	400	1040
8	4	22000	64	1150	2600

6 Discussion

The tests confirm the claims in the introduction. 1) Identities are common in algorithms that construct objects. Tests 1, 5, 7, and 8 have many constructions and 5%–10% of the predicates are identities, whereas the other tests have few constructions and under 2% identities. 2) Identities are a computational bottleneck for prior degeneracy detection algorithms. For rational predicates, exact evaluation is median 8 and maximum 216 times slower than SDD on tests 1, 3, 4, and 5, which have high expression depth. For algebraic predicates, separation bounds are useless for all the tests. 3) SDD is reliable and fast. For parameter settings with a minuscule estimated error rate ($k = 2$ and $h = 256$), the predicate evaluation time is at most 60% of the CPU time. Hence, no alternate algorithm could reduce the overall running time by more than a factor of two. 4) The running time grows slowly with the parameters k and h, so there is no need for fine-tuning. For example, $h = 265$ is larger than necessary for most of the tests, but the additional cost is insignificant. 5) For algebraic predicates, the perturbation-based algorithm far outperforms the quotient-ring-based algorithm.

The error estimate for the perturbation-based SDD algorithm depends on the assumption that the predicate is analytic on $[0, r]$. In practice, a value of h that yields a small estimate also ensures that ambiguity at h is equivalent to identity. This implies not only that a) a nondegenerate yet ambiguous $e(p)$ is unlikely but also that b) an identity $e(p)$ rarely becomes a non-identity $e(q)$. Since identity is more restrictive than non-identity, c) going from non-identity to identity is even less likely. The probability of an undetected analytic failure is the product of (a) and (b). Although we do not have an analysis of these probabilities as for an analytic false identity, we feel it is safe to disregard their product in comparison to the analytic false identity probability.

References

[1] C. Arluck, V. Milenkovic, and E. Sacks. Approximate free space construction and maximum clearance path planning for a four degree of freedom robot. In *Proceedings of the Canadian Conference on Computational Geometry*, 2018.

[2] F. Avnaim and J.-D. Boissonnat. Simultaneous containment of several polygons. In *Symposium on Computational Geometry*, pages 242–247, 1987.

[3] E. Berberich, P. Emeliyanenko, A. Kobel, and M. Sagraloff. Exact symbolic-numeric computation of planar algebraic curves. *Theoretical Computer Science*, 491(C):1–32, 2013.

[4] J. Blömer. A probabilistic zero-test for expressions involving roots of rational numbers. In G. Bilardi, G. F. Italiano, A. Pietracaprina, and G. Pucci, editors, *Algorithms — ESA' 98*, pages 151–162. Springer Berlin Heidelberg, 1998.

[5] H. Bronnimann, C. Burnikel, and S. Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Applied Mathematics*, 109(1-2):25–47, 2001.

[6] H. Brönnimann, I. Z. Emiris, V. Y. Y. Pan, and S. Pion. Sign determination in residue number systems. *Theoretical Computer Science*, 210:173–197, 1999.

[7] C. Burnikel, S. Funke, K. Mehlhorn, S. Schirra, and S. Schmitt. A separation bound for real algebraic expressions. *Algorithmica*, 55(1):14–28, 2009.

[8] Cgal, Computational Geometry Algorithms Library. http://www.cgal.org.

[9] I. Emiris, B. Mourrain, and E. Tsigaridas. Separation bounds for polynomial systems. *Journal of Symbolic Computation*, page 128–151, 2020.

[10] I. Z. Emiris. A Complete Implementation for Computing General Dimensional Convex Hulls. Technical Report RR-2551, INRIA, May 1995.

[11] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple precision binary floating point library with correct rounding. *ACM Transactions on Mathematical Software*, 33:13, 2007.

[12] D. Halperin. Controlled perturbation for certified geometric computing with fixed-precision arithmetic. In *ICMS*, pages 92–95, 2010.

[13] C. Li, S. Pion, and C. Yap. Recent progress in exact geometric computation. *Journal of Logic and Algebraic Programming*, 64:85–111, 2005.

[14] J. Masterjohn, V. Milenkovic, and E. Sacks. Finding intersections of algebraic curves in a convex region using encasement. In *Proceedings of the Canadian Conference on Computational Geometry*, pages 91–97, 2018.

[15] K. Melhorn and S. Näher. *The LEDA platform for combinatorial and geometric computing.* Cambridge University Press, 1999.

[16] V. Milenkovic and E. Sacks. Geometric rounding and feature separation in meshes. *Computer-Aided Design*, 108:12–18, 2019.

[17] V. Milenkovic, E. Sacks, and N. Butt. Fast detection of degenerate predicates in free space construction. *International Journal of Computational Geometry & Applications*, 29(03):219–237, 2019.

[18] V. Milenkovic, E. Sacks, and S. Trac. Robust free space computation for curved planar bodies. *IEEE Transactions on Automation Science and Engineering*, 10(4):875–883, 2013.

[19] R. E. Moore. *Methods and Applications of Interval Analysis.* SIAM Studies in Applied Mathematics. SIAM, Philadelphia, 1979.

[20] E. Sacks, N. Butt, and V. Milenkovic. Robust free space construction for a polyhedron with planar motion. *Computer-Aided Design*, 90C:18–26, 2017.

[21] E. Sacks and V. Milenkovic. Robust cascading of operations on polyhedra. *Computer-Aided Design*, 46:216–220, Jan. 2014.

[22] J. R. Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete and Computational Geometry*, 18:305–363, 1997.

[23] H. Si and K. Gartner. 3D boundary recovery by constrained delaunay tetrahedralization. *International Journal for Numerical Methods in Engineering*, 85:1341–1364, 2011.

[24] H. Si and J. R. Shewchuk. Incrementally constructing and updating constrained Delaunay tetrahedralizations with finite-precision coordinates. *Engineering with Computers*, 30:253–269, 2014.

[25] C. Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of discrete and computational geometry*, chapter 41, pages 927–952. CRC Press, Boca Raton, FL, second edition, 2004.