International Journal of Computational Geometry
& Applications
Vol. 29, No. 3 (2019) 219–237

DOI: 10.1142/S0218195919500067

Fast Detection of Degenerate Predicates in Free Space Construction

Victor Milenkovic*

Department of Computer Science, University of Miami
Coral Gables, Florida 33124-4245, USA
vjm@cs.miami.edu

Elisha Sacks

Computer Science Department, Purdue University
West Lafayette, Indiana 47907-2066, USA
eps@cs.purdue.edu

Nabeel Butt

Facebook, 1 Hacker Way, Menlo Park
California 94025, USA
nfb@fb.com

Received 25 September 2016
Accepted 18 December 2018
Published 25 October 2019
Communicated by M. Lin

An implementation of a computational geometry algorithm is robust if the combinatorial output is correct for every input. Robustness is achieved by ensuring that the predicates in the algorithm are evaluated correctly. A predicate is the sign of an algebraic expression whose variables are input parameters. The hardest case is detecting *degenerate* predicates where the value of the expression equals zero. We encounter this case in constructing the free space of a polyhedron that rotates around a fixed axis and translates freely relative to a stationary polyhedron. Each predicate involved in the construction is expressible as the sign of a univariate polynomial f evaluated at a zero t of a univariate polynomial g, where the coefficients of f and g are polynomials in the coordinates of the polyhedron vertices. A predicate is degenerate when t is a zero of a common factor of f and g. We present an efficient degeneracy detection algorithm based on a one-time factoring of all the univariate polynomials over the ring of multivariate polynomials in the vertex coordinates. Our algorithm is 3500 times faster than the standard algorithm based on greatest common divisor computation. It reduces the share of degeneracy detection in our free space computations from 90% to 0.5% of the running time.

Keywords: Robust computational geometry; configuration spaces; multivariate polynomial factoring.

*Corresponding author.

1. Introduction

An implementation of a computational geometry algorithm is robust if the combinatorial output is correct for every input. Robustness is achieved by ensuring that the predicates in the algorithm are evaluated correctly. A predicate is the sign of an algebraic expression whose variables are input parameters. The predicate is degenerate when the value of the algebraic expression equals zero. Predicates must be evaluated using computer arithmetic. Nondegenerate predicates can be evaluated efficiently using floating point arithmetic (Sec. 2). Efficient evaluation of degenerate predicates is a research problem.

Prior research mainly addresses degeneracy due to input in special position, such as the signed area of a triangle with collinear vertices. Such degeneracy can be eliminated by input perturbation.[1] We address predicates that are degenerate for all input, which we call *identities*. One example is the signed area of a triangle pab with p the intersection point of line segments ab and cd. The sign is always zero because the algebraic expression for the area in the coordinates of a, b, c, and d is the zero polynomial. This identity can occur when constructing the convex hull of the intersection of two polygons in general position. Figure 1 shows polygons abc and $defgh$ that intersect at points $\{p_1, \ldots, p_6\}$. The convex hull algorithm encounters an identity when it evaluates the signed area of any three of $\{p_1, p_2, p_3, p_4\}$. Triangulating the intersection of two polygons involves similar identities.

Identities are common when an algorithm evaluates predicates on arguments that are derived by another algorithm, such as the signed areas evaluated on the derived p_i in our example. When the algebraic expressions are rational, the identities are amenable to polynomial identity detection.[2] We address irrational expressions involving zeros of univariate polynomials.

There are two prior approaches to identity detection (Sec. 2). One uses software arithmetic, computer algebra, and root separation bounds to detect all degenerate predicates, including identities.[3] The second adds identity detection logic to computational geometry algorithms. In the convex hull example, this logic checks if three points lie on a single input segment. The first approach can greatly increase the running time of the software and the second approach can greatly increase the software development time.

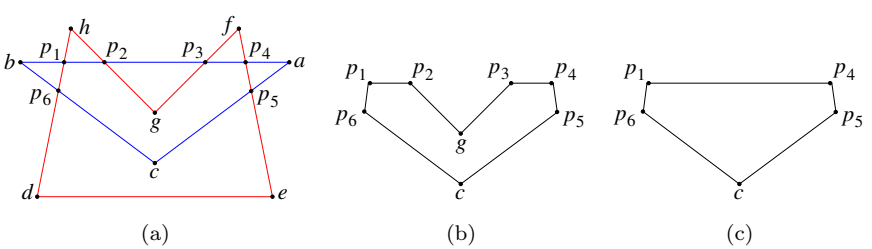


Fig. 1. (a) Polygons, (b) intersection and (c) convex hull.

We present a new approach to identity detection that avoids the high running time of numerical identity detection and the long development time of identity detection logic. The approach applies to a class of computational geometry algorithms with a common set of predicates. We write a program that enumerates and analyzes the predicates. The predicates are represented by algebraic expressions with canonical variables. The hull example requires 24 canonical variables for the coordinates of the 12 input points that define the three arguments of the signed area. When implementing the algorithms, we match their arguments against the canonical variables and use the stored analysis to detect identities.

We apply this approach in constructing the free space of a polyhedron R that rotates around a fixed axis and translates freely relative to a stationary polyhedron O (Secs. 3 and 8). For example, R models a drone helicopter and O models a warehouse. The configuration of R is specified by four parameters: the three coordinates of its position vector and a parameter t that represents the rotation angle. The configuration space is the four-dimensional space of configurations. The free space is the subset of the configuration space where the interiors of R and O are disjoint. Robust and efficient free space construction software would advance motion planning, part layout, assembly planning, and mechanical design.

The structure of the free space is determined by the configurations where R has contacts with O. A contact occurs between a feature of O and a feature of R that together have four vertices: a vertex that is on a facet or two edges that are tangent. The rotation parameter of a configuration with four contacts is a zero of a univariate polynomial of degree 6, which we call an *angle polynomial*. The coefficients of an angle polynomial are multivariate polynomials in the 48 coordinates of the 16 vertices of the four pairs of features in contact. Every predicate in free space construction is expressible as the sign of an angle polynomial f evaluated at a zero $t = t_0$ of an angle polynomial g (Sec. 5). A predicate is degenerate when t_0 is a zero of a common factor h of f and g. A predicate is an identity when h corresponds to a common factor of f and g considered as 49-variate polynomials in t and in the vertex coordinates.

Neither prior identity detection approach is practical. Detecting an identity as a zero of the greatest common divisor of f and g is slow (Sec. 9). Devising identity detection logic for every predicate is infeasible because there are over 450,000,000 predicates and 26,000 identities. We present an efficient identity detection algorithm (Sec. 6) based on a one-time analysis of the angle polynomials (Sec. 7).

For the one-time analysis, we enumerate the angle polynomials using canonical variables for the vertex coordinates. Working in the monomial basis is impractical because many of the angle polynomials have over 100,000 terms. Instead, we represent angle polynomials with a data structure, called an *a-poly*, that is a list of lists of vertices (Sec. 4). The enumeration yields 1,000,000 a-polys, which we reduce to the 30,000 representatives of an isomorphism that respects factorization. We construct a table of factors for the isomorphism class representatives in one CPU-day on a workstation.

We factor an angle polynomial by looking up the factors of its representative in the table and substituting its vertex coordinates for the canonical variables. We use the factoring algorithm to associate each zero $t = t_0$ of an angle polynomial g with an irreducible factor h. Before evaluating a predicate at t_0, we factor its angle polynomial f. The predicate is an identity if h is one of the factors. Our algorithm is 3500 times faster than computing greatest common divisors. It reduces the share of degeneracy detection in our free space computations from 90% to 0.5% of the running time (Sec. 9). We conclude the paper with guidelines for applying our identity detection strategy to other domains (Sec. 10).

2. Prior Work

Identity detection is the computational bottleneck in prior work by Hachenberger on computing the Minkowski sum of two polyhedra.[4] He partitions the polyhedra into convex components and forms the union of the Minkowski sums of the components. Neighboring components share common, collinear, or coplanar features, resulting in many identities in the union operations. Detecting the identities via the numerical approach (using CGAL) dominates the running time.

Mayer *et al.* partially compute the free space of a convex polyhedron that rotates around a fixed axis and translates freely relative to a convex obstacle.[5] They report no identity detection problems. Identities can be detected using one rule: all polynomials generated from a facet of one polyhedron and an edge of the other are the same up to sign and hence their zeros are identical. These polynomials correspond to our type I predicates for general polyhedra (Sec. 8).

We address identities in four prior works. We compute polyhedral Minkowski sums orders of magnitude faster than Hachenberger by using a convolution algorithm, which has fewer identities, and by detecting identities with special case logic.[6] We compute free spaces of planar parts bounded by circular arcs and line segments.[7] The number of identities is small, but the proof of completeness is lengthy. We compute free spaces of polyhedra where R translates in the xy plane and rotates around the z axis.[8] The identity detection logic is confirmed using our new approach. There are 816 isomorphism classes with 290 in the basis versus 30,564 and 15,306 for the 4D configuration space (Sec. 7).

Finally, we find placements for three polyhedra that translate in a box. The algorithm performs a sequence of ten Minkowski sums and Boolean operations, resulting in many identities.[9] One implementation handles the identities as special cases. A second implementation prevents identities with a polyhedron approximation algorithm that rounds and perturbs the output of each step. The former is twice as fast as the latter and is exact, but took months to develop and lacks a correctness proof.

Our prior work demonstrates efficient evaluation of nondegenerate predicates. We implement the algorithms robustly using our Adaptive Controlled Perturbation (ACP) strategy. The numeric input is a vector of the parameters of the input geometric objects. We construct a perturbed input that is in general position by

adding a random number in $[-\delta, \delta]$ to each parameter. When we evaluate a predicate, we first try floating point interval arithmetic. If the sign of its expression is ambiguous, we increase the precision until the sign is resolved. The output is correct for the perturbed input, which is δ-close to the input in the max norm. Using $\delta = 10^{-8}$, the predicate evaluation time is at most 20% of the running time in our implementations.

3. Free Space

We study free space construction for polyhedra R and O with triangular facets. Without loss of generality, we use the z axis as the axis of rotation. We represent the rotation angle using a rational parameterization of the unit circle. A configuration c of R is a rotation parameter t and a translation vector d, denoted $c = (t, d)$. The configuration c maps a point p to the point $c(p) = d + \Theta(t)p$ with

$$\Theta(t)p = \left(\frac{(1-t^2)p_x - 2tp_y}{1+t^2}, \frac{2tp_x + (1-t^2)p_y}{1+t^2}, p_z \right). \tag{1}$$

A point set P maps to $c(P) = \{c(p) \,|\, p \in P\}$. The free space is $\{c \,|\, O \cap c(R) = \emptyset\}$.

The boundary of the free space consists of *contact configurations* c at which the boundaries of $c(R)$ and O intersect but not the interiors. The generic contacts are a vertex $c(r_k)$ of $c(R)$ on a facet $o_h o_i o_j$ of O, a vertex o_h of O on a facet $c(r_i r_j r_k)$ of $c(R)$, and an edge $o_h o_i$ of O intersecting an edge $c(r_j r_k)$ of $c(R)$. The boundary has faces of dimension $k = 0, \ldots, 3$. A face of dimension k consists of configurations where $4 - k$ contacts occur but the interiors of O and $c(R)$ are disjoint.

The four vertices of the two features of a contact are coplanar, so their tetrahedron has zero volume. We obtain a rational expression in c by substituting the vertices into the formula $(q - p) \times (u - p) \cdot (v - p)$. The expressions are $n \cdot (c(r_k) - o_h)$ with $n = (o_i - o_j) \times (o_h - o_j)$, $n = \Theta(t)((r_i - r_k) \times (r_j - r_k))$, and $n = (o_h - o_i) \times \Theta(t)(r_k - r_j)$. The first expression is zero if $c(r_k)$ lies on the plane $o_h o_i o_j$ or equivalently $d \in \text{plane}(o_h o_i o_j) - \theta(t) r_k$, where $A - B = \{a - b \,|\, a \in A \text{ and } b \in B\}$. We use the notation $o_h o_i o_j - r_k$ to denote either the expression or its zero set. We refer to the expression as a *contact expression* and the zero set as a 1-*contact*. Similarly the expressions or 1-contacts $o_h - r_i r_j r_k$ and $o_h o_i - r_j r_k$.

Figure 2 depicts a common zero c of contact expressions $p_1 = o_0 o_1 o_2 - r_1$, $p_2 = o_1 o_2 o_3 - r_1$, $p_3 = o_0 - r_0 r_1 r_2$, and $p_4 = o_0 - r_0 r_1 r_3$. Testing if c is a contact configuration gives rise to a typical identity. For the interiors of O and $c(R)$ to be disjoint, $c(r_0)c(r_2)$ cannot pierce $o_0 o_1 o_2$. Testing this uses the signs of five other contact expressions, including $o_0 o_1 o_2 - r_0$. Because $p_1 = p_2 = 0$ at c, $c(r_1) \in \text{plane}(o_0 o_1 o_2) \cap \text{plane}(o_1 o_2 o_3) = \text{line}(o_1 o_2)$. Similarly, $p_3 = p_4 = 0$ implies $o_0 \in \text{line}(c(r_0)c(r_1))$. Since this line shares two points with $\text{plane}(o_0 o_1 o_2)$, they are coplanar, $c(r_0) \in \text{plane}(o_0 o_1 o_2)$, and so $o_0 o_1 o_2 - r_0$ is identically zero at c. This identity resembles the signed area identities in Sec. 1 in that a polynomial is evaluated on arguments that are derived from the input. However, we cannot apply

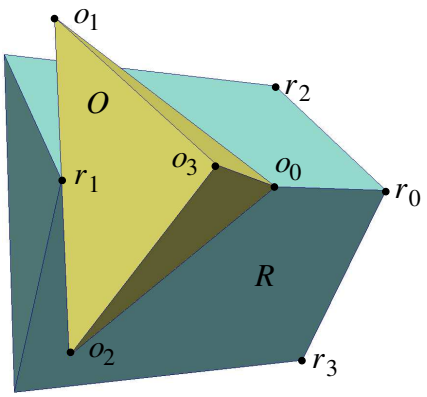


Fig. 2. Identity in piercing predicate: r_0 is in the plane of $o_0o_1o_2$.

polynomial identity detection because the arguments are zeros of polynomials, not rational functions of the input.[2]

4. Angle Polynomials

Every contact expression has the form $n \cdot d + m$ with n and m (vector and scalar) functions of t (Sec. 3). Four contact expressions $n_i \cdot d + m_i$ have a common zero (t, d) if the matrix

$$\begin{vmatrix} n_{1x} & n_{1y} & n_{1z} & m_1 \\ n_{2x} & n_{2y} & n_{2z} & m_2 \\ n_{3x} & n_{3y} & n_{3z} & m_3 \\ n_{4x} & n_{4y} & n_{4z} & m_4 \end{vmatrix}$$

has a zero determinant and has a 3-by-3 left minor with a nonzero determinant. We call the numerators of the determinants *angle polynomials*. The denominator is a power of $1 + t^2$. For each zero t of the matrix angle polynomial, we find a minor whose angle polynomial is nonzero at t using identity detection. If one is found, the three expressions corresponding to the minor have a unique zero d at that t.

The vectors n_i have the form a, $\Theta(t)a$, or $a \times \Theta(t)b$ and the scalars m_i are sums of terms of the form $a \cdot b$ or $a \cdot \Theta(t)b$ with a and b constant vectors. We expand the matrix determinant along the last column. The determinants of the minors are $n_1 \cdot (n_2 \times n_3)$ etc. Using the formula $u \times (v \times w) = (u \cdot w)v - (u \cdot v)w$, they reduce to sums of terms of the form $a \cdot b$, $a \cdot \Theta(t)b$, or $(a \cdot \Theta(t)b)(c \cdot \Theta(t)d)$. Applying Eq. (1) and clearing the denominator reduces the m_i to quadratics in t, the minors to quartics in t, and the matrix angle polynomial to degree 6.

We define 2-*contacts* $o_io_j - r_k = \{c \,|\, c(r_k) \in \text{line}(o_io_j)\}$ and $o_i - r_jr_k = \{c \,|\, o_i \in c(\text{line}(r_jr_k))\}$ and 3-*contact* $o_i - r_j = \{c \,|\, c(r_j) = o_i\}$. If we introduce dummy vertices, a 2-contact can be expressed as the intersection of two 1-contacts, e.g.

$o_io_j - r_k = (o_io_jo - r_k) \cap (o_io_jo' - r_k)$ for dummies o and o'. Similarly, a 3-contact can be expressed as the intersection of three 1-contacts.

We represent an angle polynomial with an *a-poly:* a list of elements of the form $L_O - L_R$ with L_O and L_R lists of vertices of O and of R in increasing index order. Elements are in order of increasing $|L_O| + |L_R|$, increasing $|L_O|$, then increasing vertex index (lexicographically). Each list has zero to three elements. The a-poly of a matrix polynomial is a list of k-contacts with a total k of 4. The a-poly of a minor of 1-contacts is a list of three elements that represent the vector n of its rows: $o_ho_io_j-$ for a row arising from a 1-contact $o_ho_io_j - r_k$, $-r_ir_jr_k$ for $o_h - r_ir_jr_k$, and $o_ho_i - r_jr_k$ for $o_ho_i - r_jr_k$. The first two notations drop a vertex that does not contribute to the minor. For minors involving 2-contacts and 3-contacts, the a-polys take two other forms. The first form is $(-r_{i_1}r_{j_1}, -r_{i_2}r_{j_2}, o_{i_3}o_{j_3}-)$ or $(-r_{i_1}r_{j_1}, o_{i_2}o_{j_2}-, o_{i_3}o_{j_3}-)$, equivalent to $(-r_{i_2}r_{j_2}r, -r_{i_2}r_{j_2}r', o_{i_3}o_{j_3} - r_{i_1}r_{j_1})$ or $(o_{i_2}o_{j_2}o-, o_{i_2}o_{j_2}o'-, o_{i_3}o_{j_3} - r_{i_1}r_{j_1})$ with dummies o, o', r, r'. The second form is $(o_{i_1}o_{j_1} - r_{i_2}r_{j_2}r_{k_2})$ or $(o_{i_1}o_{j_1}o_{k_1} - r_{i_2}r_{j_2})$, equivalent to $(-r_{i_2}r_{j_2}, -r_{i_2}r_{k_2}, o_{i_1}o_{j_1}-)$ or $(-r_{i_2}r_{j_2}, o_{i_1}o_{j_1}-, o_{i_1}o_{k_1}-)$.

We calculate the polynomials of a-polys involving 2-contacts or 3-contacts directly rather than introducing dummy vertices and calculating determinants. A matrix angle polynomial can by defined by a 2-contact and two 1-contacts, by two 2-contacts, or by a 3-contact and a 1-contact. We compute these angle polynomials as follows.

For a 2-contact $o_io_j - r_k$, $c(r_k) = d + \Theta(t)r_k \in \text{line}(o_io_j)$, so d is on the line through $o_i - \Theta(t)r_k$ and $o_j - \Theta(t)r_k$. We intersect this line with two 1-contacts. We express the line as $\lambda u + v$ with $u = o_j - o_i$ and $v = o_j - \Theta(t)r_k$, compute the values $\lambda_i = -(n_i \cdot v + m_i)/(n_i \cdot u)$ where the line intersects the two planes $n_i \cdot p + m_i = 0$, set $\lambda_1 = \lambda_2$, and cross multiply to obtain a quartic angle polynomial. Similarly 2-contact $o_i - r_jr_k$ corresponds to a line with $u = \Theta(t)(r_j - r_k)$ and $v = o_i - \Theta(t)r_j$. For two 2-contacts, the contact expression is the signed volume of the four points that define their lines, which yields a quartic angle polynomial. For a 3-contact $o_i - r_j$, $o_i = d + \Theta(t)r_j$. We substitute $d = o_i - \Theta(t)r_j$ into the contact expression to obtain a quadratic angle polynomial.

The angle polynomials of minors involving 2-contacts and 3-contacts are quadratics. The first form is zero when edges $c(r_{i_1}r_{j_1})$, $c(r_{i_2}r_{j_2})$, and $o_{i_3}o_{j_3}$ are parallel to a common plane; likewise edges $c(r_{i_1}r_{j_1})$, $o_{i_2}o_{j_2}$, and $o_{i_3}o_{j_3}$. The second form is zero when an edge $o_{i_1}o_{j_1}$ is parallel to a facet $c(r_{i_2}r_{j_2}r_{k_2})$; likewise an edge $c(r_{i_2}r_{j_2})$ and a facet $o_{i_1}o_{j_1}o_{k_1}$. The angle polynomials of both forms are obtained by applying Eq. (1) to $\Theta(t)((r_{j_1} - r_{i_1}) \times (r_{j_2} - r_{i_2})) \cdot (o_{j_3} - o_{i_3})$ and $\Theta(t)(r_{j_1} - r_{i_1}) \cdot ((o_{j_2} - o_{i_2}) \times (o_{j_3} - o_{i_3}))$ and clearing the denominator.

5. Predicates

Some predicates in free space construction are the sign of an angle polynomial evaluated at a zero of another angle polynomial. The rest are the sign of a contact expression p evaluated at a zero $c_0 = (t_0, d_0)$ of four contact expressions $\{p_1, p_2, p_3, p_4\}$.

We convert the second form to the first form as follows. Let $\{p_i, p_j, p_k\}$ have a nonzero left minor at t_0 and let f be the angle polynomial of $\{p_i, p_j, p_k, p\}$. We show that $f(t_0) = 0$ iff $p(c_0) = 0$. If $f(t_0) = 0$, $\{p_i, p_j, p_k, p\}$ are linearly dependent in d at $t = t_0$, so p is a linear combination of $\{p_i, p_j, p_k\}$ and $p(c_0) = 0$. If $p(c_0) = 0$, $\{p_i, p_j, p_k, p\}$ must be linearly dependent in d at $t = t_0$ because $p_i(c_0) = p_j(c_0) = p_k(c_0) = 0$, so $f(t_0) = 0$.

We see that every predicate is the sign of an angle polynomial f evaluated at a zero $t = t_0$ of an angle polynomial g. The predicate is an identity if f and g, considered as 49-variate polynomials in t and in the vertex coordinates, have a common factor h, and $h(t_0) = 0$ when the coordinates have their input values. We make identity detection fast by enumerating the polynomials, factoring them, and storing the results in a table, as we discuss next.

6. Factoring

We factor angle polynomials represented as a-polys. Two a-polys are *equivalent* if they denote the same polynomial up to sign. For example, an a-poly involving 2-contacts has an equivalent a-poly involving only 1-contacts. Two a-polys are *isomorphic* if a vertex bijection maps one to the other. For example, $o_0o_1o_2r_0r_1 \rightarrow o_0o_2o_1r_1r_0$ maps $(o_0o_2 - r_1, o_0o_1 - r_0r_1, o_0o_1o_2 - r_0)$ to $(o_0o_1 - r_0, o_0o_1 - r_0r_1, o_0o_1o_2 - r_1)$. The bijection maps a factorization of one a-poly to a factorization of the other. The factoring algorithm uses a table that contains the factorization of a representative a-poly from each isomorphism class. Factor a-polys are drawn from a minimal set of basis isomorphism classes. Since isomorphic a-polys can also be equivalent, a factor is represented as a maximal set of equivalent a-polys from a basis class. This representation is compact because almost all the sets are singletons. Table construction is explained in Sec. 7. The table is available in the web directory http://www.cs.miami.edu/home/vjm/robust/identity.

The factoring algorithm maps an input a-poly to a unique representative of its isomorphism class, obtains the factor sets of the representative from the table, and applies the inverse map to obtain sets of a-polys in the vertices of the input a-poly. It selects the lexicographical minimum from each set, using a vertex order that we indicate by the o and r indices. Selecting the minimum ensures that the algorithm generates a unique a-poly for each factor. Hence, the factorizations of angle polynomials with a common factor contain the unique a-poly of that factor.

6.1. *Mapping an a-poly to its representative*

The mapping algorithm generates the permutations of the input a-poly such that the elements remain increasing in $|L_O| + |L_R|$ then in $|L_O|$, but disregarding the lexicographical order of vertex indices. For each permutation, it assigns each vertex an indicator: a bit string in which a 1 in position k indicates that the vertex

appears in the kth element. It labels a permutation with its O vertex indicators in decreasing order followed by its R vertex indicators in decreasing order. It selects the permutation with the largest label, replaces the ith O vertex in indicator order by the canonical vertex o_i, and likewise for the R vertices and r_i. If two vertices have the same indicator, both orders yield the same output because the indices of an a-poly are placed in increasing order.

In the a-poly $(o_{27} - r_{22}r_{66}r_{86}, o_{51} - r_{22}r_{66}r_{86}, o_{27}o_{43} - r_{15}r_{86}, o_{27}o_{51} - r_{75}r_{86})$, o_{27} has indicator 1011 because it appears in the first, third and fourth element. The indicator list is 1011, 0101, 0010; 1111, 1100, 1100, 0010, 0001. The permutations swap the first two elements and/or the last two elements. Swapping the last two elements yields the maximal indicator list 1011, 0110, 0001; 1111, 1100, 1100, 0010, 0001, so the map is $o_{27}o_{51}o_{43}r_{86}r_{22}r_{66}r_{75}r_{15} \to o_0o_1o_2r_0r_1r_2r_3r_4$ and the representative is $(o_0 - r_0r_1r_2, o_1 - r_0r_1r_2, o_0o_1 - r_0r_3, o_0o_2 - r_0r_4)$. In the table, this representative has (singleton) factor sets $\{(o_0o_1 - r_0r_1r_2)\}$ and $\{(-o_0o_1o_2, o_0o_1 - r_0r_3, o_0o_2 - r_0r_4)\}$. The inverse vertex mapping yields the factors $(o_{27}o_{51} - r_{22}r_{66}r_{86})$ and $(-r_{22}r_{66}r_{86}, o_{27}o_{43} - r_{15}r_{86}, o_{27}o_{51} - r_{75}r_{86})$.

6.2. *Selecting a unique a-poly factor*

Figure 3 illustrates equivalent a-polys. The 1-contacts $o_0o_1 - r_0r_1$, $o_0o_1 - r_2r_3$, and $o_0o_1o_2 - r_4$ determine the rotation parameter t because they are invariant under translation parallel to o_0o_1. Translating R parallel to o_0o_1 until one element becomes a 2-contact yields an a-poly C_i that is zero at the same t values. If r_0r_1 hits o_0, $C_1 = (o_0 - r_0r_1, o_0o_1 - r_2r_3, o_0o_1o_2 - r_4)$ (Fig. 3(b)) and if r_2r_3 hits o_1, $C_2 = (o_1 - r_2r_3, o_0o_1 - r_0r_1, o_0o_1o_2 - r_4)$. These equivalent a-polys are isomorphic under the map from C_1 to C_2 : $o_o o_1o_2r_0r_1r_2r_3r_4 \to o_1o_0o_2r_2r_3r_0r_1r_4$. There are also C_3 and C_4 where r_0r_1 hits o_1 or r_2r_3 hits o_0 and $C_5 = (o_0o_2 - r_4, o_0o_1 - r_0r_1, o_0o_1 - r_2r_3)$ and $C_6 = (o_1o_2 - r_4, o_0o_1 - r_0r_1, o_0o_1 - r_2r_3)$ where r_4 hits o_0o_2 or o_1o_2. C_5 and C_6 are equivalent but not isomorphic to C_1, C_2, C_3, and C_4 because their first element has two O vertices, not one.

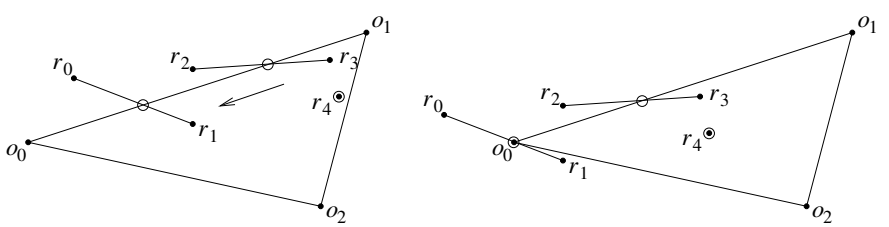


Fig. 3. Equivalent a-polys: (a) translation parallel to o_0o_1 preserves circled 1-contacts $o_0o_1 - r_0r_1$, $o_0o_1 - r_2r_3$, and $o_0o_1o_2 - r_4$; (b) a-poly $(o_0 - r_0r_1, o_0o_1 - r_2r_3, o_0o_1o_2 - r_4)$.

The a-poly $(o_1 - r_0r_1r_2, o_0o_1 - r_3r_4, o_0o_1 - r_5r_6, o_oo_1o_2 - r_7)$ maps to the representative $(o_0 - r_0r_1r_2, o_0o_1 - r_3r_4, o_0o_1 - r_5r_6, o_oo_1o_2 - r_7)$ with factor sets $\{(o_0o_1 - r_0r_1r_2)\}$ and

$$\{(o_0 - r_3r_4, o_0o_1 - r_5r_6, o_0o_1o_2 - r_7), (o_0 - r_5r_6, o_0o_1 - r_3r_4, o_0o_1o_2 - r_7),$$
$$(o_1 - r_3r_4, o_0o_1 - r_5r_6, o_0o_1o_2 - r_7), (o_1 - r_5r_6, o_0o_1 - r_3r_4, o_0o_1o_2 - r_7)\}$$

because the second factor is isomorphic to $C_1, \ldots, C_4$. Its factor set does not contain a-polys isomorphic to C_5 and C_6 because their class is not in the basis. The inverse map (in this case swapping o_0 and o_1) results in factor sets $\{(o_0o_1 - r_0r_1r_2)\}$ and

$$\{(o_1 - r_3r_4, o_0o_1 - r_5r_6, o_0o_1o_2 - r_7), (o_1 - r_5r_6, o_0o_1 - r_3r_4, o_0o_1o_2 - r_7),$$
$$(o_0 - r_3r_4, o_0o_1 - r_5r_6, o_0o_1o_2 - r_7), (o_0 - r_5r_6, o_0o_1 - r_3r_4, o_0o_1o_2 - r_7)\}.$$

The factors are $(o_0o_1 - r_0r_1r_2)$ and the (unique minimal) third element $(o_0 - r_3r_4, o_0o_1 - r_5r_6, o_0o_1o_2 - r_7)$ of the second set.

7. Constructing the Table of Factors

This section shows how we enumerate the a-poly isomorphism classes (Sec. 7.1), factor the class representatives and select a basis (Sec. 7.2), construct the table of factors (Sec. 7.3), and verify the table (Sec. 7.4).

7.1. *Isomorphism classes*

We enumerate the a-poly classes as follows. Let a_i, b_i, c_i, d_i, e_i, and f_i denote vertices. Let $s_i = \{a_i - d_i\}$, $t_i = \{a_i - d_ie_i, a_ib_i - d_i\}$, $u_i = \{a_i - d_ie_if_i, a_ib_i - d_ie_i, a_ib_ic_i - d_i\}$, $v_i = \{a_ib_i - c_id_ie_i, a_ib_ic_i - d_ie_i\}$, $w_i = \{-a_ib_i, a_ib_i-\}$, $x_i = \{-a_ib_ic_i, a_ib_i - c_id_i, a_ib_ic_i-\}$. We generate the determinant a-polys with the sets $s_1 \times u_2$ (a 3-contact and a 1-contact), $t_1 \times t_2$ (two 2-contacts), $t_1 \times u_2 \times u_3$ (a 2-contact and two 1-contacts), and $u_1 \times u_2 \times u_3 \times u_4$ (four 1-contacts). We generate the left minor a-polys with the sets v_1 (edge parallel to facet), $w_1 \times w_2 \times w_3$ (edges parallel to plane), and $x_1 \times x_2 \times x_3$ (minor). For each element of each set, we assign O vertices to the a_i, b_i, c_i in every possible manner. Starting from o_0, we assign increasing indices to the vertices of an edge or a facet. We assign R vertices to d_i, e_i, f_i likewise.

For example, $s_1 \times u_2 = \{(a_1 - d_1, a_2 - d_2e_2f_2), (a_1 - d_1, a_2b_2 - d_2e_2), (a_1 - d_1, a_2b_2c_2 - d_2)\}$. We must set $a_1 = o_0$. We can set $a_2 = o_0$ or $a_2 = o_1$ because a_2 is in a different feature, and then assign increasing indices to b_2 and c_2 because they are in the same feature. Similarly for d_1, d_2, e_2, and f_2. The results are

$$\{(o_0 - r_0, o_0 - r_0r_1r_2), (o_0 - r_0, o_1 - r_0r_1r_2), (o_0 - r_0, o_0 - r_1r_2r_3),$$
$$(o_0 - r_0, o_1 - r_1r_2r_3), (o_0 - r_0, o_0o_1 - r_0r_1), (o_0 - r_0, o_1o_2 - r_0r_1),$$

$$(o_0 - r_0, o_0o_1 - r_1r_2), (o_0 - r_0, o_1o_2 - r_1r_2), (o_0 - r_0, o_0o_1o_2 - r_0),$$
$$(o_0 - r_0, o_1o_2o_3 - r_0), (o_0 - r_0, o_0o_1o_2 - r_1), (o_0 - r_0, o_1o_2o_3 - r_1)\}.$$

The enumeration yields about one million a-polys. Generating their representatives (Sec. 6) and removing duplicates yields 30,564 isomorphism class representatives.

7.2. *Basis classes*

We factor the representatives probabilistically. We replace the canonical coordinates of $o_0, \ldots, o_{11}$ and $r_0, \ldots, r_{11}$ with random integers, construct the resulting univariate integer polynomials, and factor them with Mathematica. An irreducible univariate implies that the canonical polynomial is irreducible; the converse is true with high probability.

An a-poly depends on a vertex if its univariate changes when the coordinates of the vertex are assigned different random integers. For example, $(o_0o_1 - r_0, o_2 - r_1r_2r_3, o_2 - r_1r_2r_4)$ does not depend on r_3 and r_4 because $o_2 - r_1r_2r_3$ and $o_2 - r_1r_2r_4$ can be replaced by $o_2 - r_1r_2$: o_2 in contact with $r_1r_2r_3$ and $r_1r_2r_4$ is equivalent to o_2 in contact with r_1r_2. An a-poly is *complete* if it depends on all of its vertices.

We select a set B of complete and irreducible class representatives, define their classes as the basis, and construct a map I from the univariate of each element of B and its permutations to the set of equivalent isomorphic a-polys. Starting with empty B and I, we visit each representative ρ. If ρ is complete, its univariate p is irreducible, and $I(p) = \emptyset$, we add ρ to B, permute its vertices in every way, calculate the univariate u for each resulting a-poly a, and add a to the set $I(u)$.

In the Sec. 6.2 example, C_1 is in B. Permutations C_2, C_3, and C_4 generate the same univariate p as C_1, so $I(p) = \{C_1, C_2, C_3, C_4\}$. C_5 and C_6 also generate p, but their representative $\rho = (o_0o_1 - r_0, o_0o_2 - r_1r_2, o_0o_2 - r_3r_4)$ generates the same polynomial q as C_1 permuted by the permutation that takes C_5 to ρ. Since $I(q)$ is not empty when ρ is considered, ρ is not added to B. Thus the basis does not contain more than one class that generates the same polynomial and $I(p)$ does not contain C_5 and C_6.

7.3. *Factor table*

The factor table provides a list of factor sets for each representative. For $\rho \in B$ with univariate p, the list is $\langle I(p) \rangle$. If $\rho \notin B$, we process each factor f of p as follows.

(1) Determine which vertices f depends on. Assign the coordinates of a vertex of ρ different random integers, generate the new univariate, and factor it. If f is not a factor, it depends on the vertex.
(2) Rename the vertices of ρ to obtain a ρ' for which the factor f' that corresponds to f depends on $o_0, o_1, \ldots, o_m$ and $r_0, r_1, \ldots, r_n$ for some m and n. Let f

depend on $o_{i_0}, o_{i_1}, \ldots, o_{i_m}$ and $r_{j_0}, r_{j_1}, \ldots, r_{j_n}$ but not on $o_{i_{m+1}}, o_{i_{m+2}}, \ldots, o_{i_m}$ and $r_{j_{n+1}}, r_{j_{n+2}}, \ldots, r_{j_n}$. Substitute $o_{i_k} \to o_k$ for $k = 1, \ldots, m$ and $r_{i_k} \to r_k$ for $k = 1, \ldots, n$.

(3) Find the factor f' of ρ' that depends on $o_0, \ldots, o_m$ and $r_0, \ldots, r_n$, and has the same degree as f. (This factor is unique; otherwise, we would consider the matching factors as a group.)

(4) Look up $I(f')$ and invert the vertex substitution to obtain a factor set.

For example, the univariate of $\rho = (o_0o_1 - r_0r_1, o_2o_3o_4 - r_2, o_2o_3o_4 - r_3, o_5o_6o_7 - r_4)$ has a factor f that depends on all its vertices and a quadratic factor g that depends on o_2, o_3, o_4, r_2, r_3. The factor set of f is $I(f) = \{(o_2o_3o_4-, o_5o_6o_7-, o_0o_1 - r_0r_1)\}$. To obtain the factor set of g, substitute $o_2, o_3, o_4, o_0, o_1, r_2, r_3, r_0, r_1 \to o_0, o_1, o_2, o_3, o_4, r_0, r_1, r_2, r_3$. The quadratic factor g' of $\rho' = (o_3o_4 - r_2r_3, o_0o_1o_2 - r_0, o_0o_1o_2 - r_1, o_5o_6o_7 - r_4)$ depends on o_0, o_1, o_2, r_0, r_1, and $I(g') = \{(o_0o_1o_2 - r_0r_1)\}$. Inverting the vertex substitution yields the second factor set of ρ: $\{(o_2o_3o_4 - r_2r_3)\}$.

To save space, if an element of B has a univariate of degree six, we do not store the polynomials of its permutations in I. We do not need them for factoring because they cannot be proper factors. To test if representative ρ with an irreducible degree-6 univariate p is in the basis, we generate the permutations of ρ and their univariates. If none has an entry in I, we add ρ to B, and we add an entry for p to I. If the univariate p' of a permutation has an entry in I, the sole factor set of ρ is the result of applying the inverse permutation to $I(p')$.

7.4. *Analysis*

Out of 30,564 representatives, 15,306 are basis, 991 are constant, 3840 are irreducible but non basis, 8263 have two factors (including 260 squares), and 2164 have three factors (including 6 cubes). Since a predicate is an a-poly evaluated on a zero of a basis a-poly, we list $450,000,000 \approx 30564 \cdot 15306$ predicates in the introduction. The number of identities is $26,000 \approx 1 \cdot 3840 + 2 \cdot 8263 - 1 \cdot 260 + 3 \cdot 2164 - 2 \cdot 6$ ways of evaluating a non basis a-poly on a zero of a basis a-poly. Of the 15306+3840 irreducible representative polynomials, 363 have two basis a-polys, 50 have three, 194 have four, and 194 have six.

The algorithm depends on the assumption that the a-poly representation is complete under factorization. The correctness of the factorization depends on probabilistic assumptions, for example that an irreducible a-poly has an irreducible univariate polynomial for the chosen vertex coordinates. We verify that the table is correct probabilistically. A factorization $f_1f_2\cdots f_m \,|\, f$ is equivalent to a polynomial identity $f - af_1f_2\cdots f_m = 0$ for some constant a. We verify the identities using Schwartz's lemma.[2] We use random 20-bit values modulo a prime p. The first substitution determines a and the rest verify the identity. We reduce the probability of an error to below 10^{-1000} with 10 minutes of verification. The completeness assumption is verified because an exception would imply an empty factor set.

The running time for factor table construction is one CPU day, including 23 hours to generate the permutations of the degree-six irreducible a-polys to test if they are in the basis. The worst case is four contacts between O vertices and R facets or vice versa, which have about 11.5 billion permutations. The tests all succeed, so perhaps we could have avoided this cost by proving a theorem.

8. Contact Set Subdivision

We continue our discussion of free space with an algorithm for constructing the faces of its boundary. The *contact set* of a 1-contact is the subset for which the two features are in contact. We construct the subdivision of a contact set induced by the other contact sets. Each predicate used in the construction has the same zero set as an a-poly, so identity detection applies. The remaining (and substantial) step in free space construction is to stitch the faces into a boundary representation of free space.

Contact sets. The contact set of $o_h o_i o_j - r_k$ is the configurations (t, d) such that $d + \Theta(t) r_k$ lies on $o_h o_i o_j$. Hence, d lies on a parametric triangle $o_h - \Theta(t) r_k$, $o_i - \Theta(t) r_k, o_j - \Theta(t) r_k$. Similarly, the contact set of $o_h - r_i r_j r_k$ is the parametric triangle $o_h - \Theta(t) r_i, o_h - \Theta(t) r_j, o_h - \Theta(t) r_k$, and the contact set of $o_h o_i - r_j r_k$, is the parametric parallelogram $o_h - \Theta(t) r_j, o_h - \Theta(t) r_k, o_i - \Theta(t) r_j, o_i - \Theta(t) r_k$.

Contact facets. We are only interested in the portion of a contact set that is on the boundary of the free space. A necessary condition is that the interiors of O and $c(R)$ are disjoint in a neighborhood of their point of contact. A *contact facet* is the restriction of a contact set to the intervals of t values where the disjointness condition holds. We express the condition in terms of the signs of a-polys (ignoring the vertex index order rule, which might change the sign), so the intervals are bounded by zeros of a-polys. For $o_h o_i o_j - r_k$, $(o_h o_i o_j - r_k r_l)$ must be positive for every edge $r_k r_l$; likewise for $o_h - r_i r_j r_k$. For $o_h o_i - r_j r_k$, let $o_h o_i$ be incident on the triangles $o_h o_i m_1$ and $o_i o_h m_2$, let $r_j r_k$ be incident on the triangles $r_j r_k n_1$ and $r_k r_j n_2$, and let s_x and t_x be the signs of $(o_h o_i m_x - r_j r_k)$ and $(o_h o_i - r_j r_k n_x)$ for $x = 1, 2$. The interiors are locally disjoint if $s_1 = s_2 = -t_1 = -t_2$.

The parametric edges of a contact facet are called contact facet edges and are in the zero sets of 2-contacts. The parametric vertices are called contact facet vertices and are in the zero sets of 3-contacts.

Contact facet subdivision. At a fixed t, each contact facet (triangle or parallelogram) is intersected and subdivided by other contact facets. The intersection of two facets is called an FF-edge, and the intersection of a facet edge and a facet is called an EF-vertex. The endpoints of an FF-edges are facet vertices or EF-vertices. Two FF-edges intersect at an FFF-vertex, the intersection of three facets.

Structure changes. The subdivision of a contact facet by other contact facets is continuous in t, except for four types of structure changes. I. Facets appear or

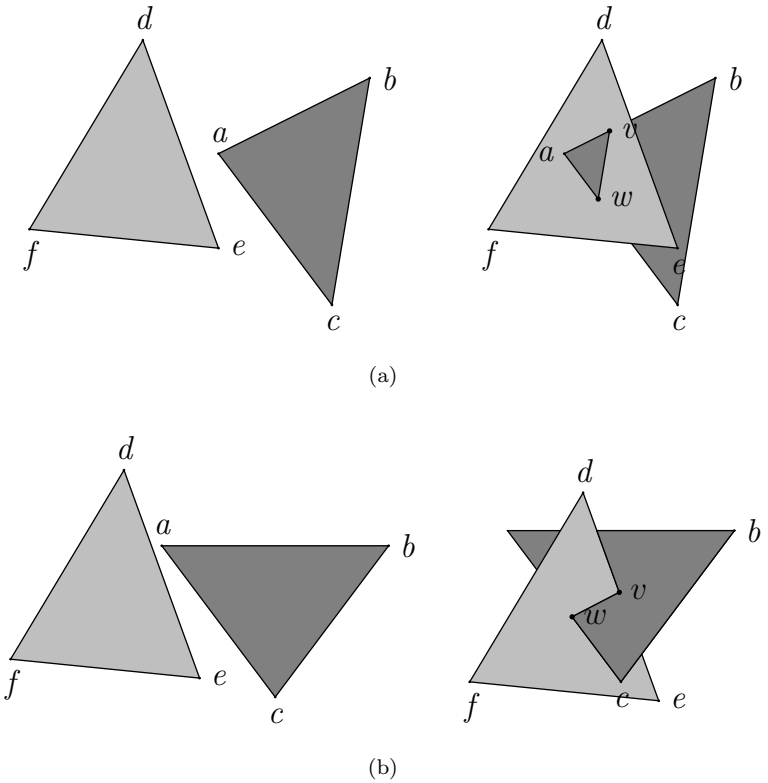


Fig. 4. Type II structure changes: (a) vertex–facet and (b) edge–edge.

disappear at the bounds of their intervals. II. FF-edges appear or disappear when a facet vertex hits a facet or when two facet edges intersect. In Fig. 4, the FF-edge vw appears when the facet vertex a hits the facet def or when the facet edges ac and de intersect. III. FFF-vertices appear or disappear when a facet edge e hits an FF-edge, and two EF-vertices swap position on e. In Fig. 5, the EF-vertices j

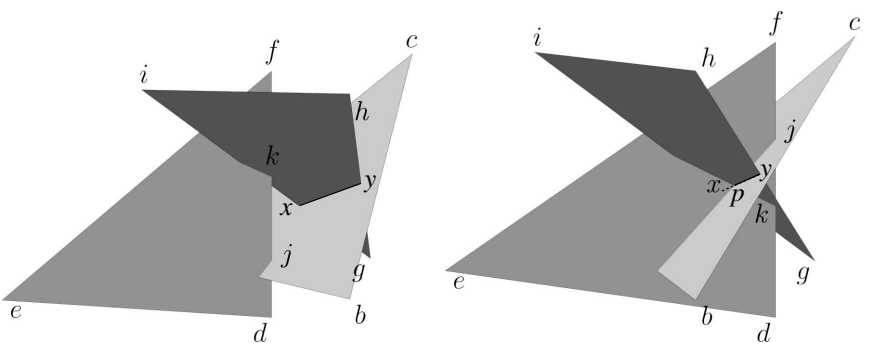


Fig. 5. Type III structure change.

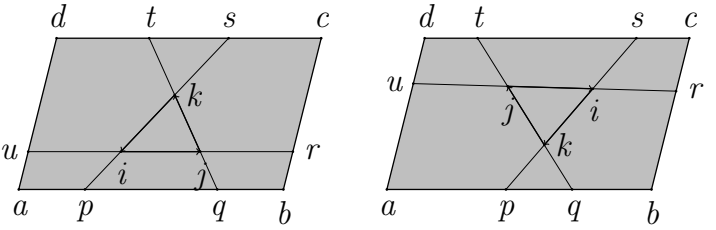


Fig. 6. Type IV structure change.

IIa (a) zero distance from a facet vertex to the plane of a facet.
(b) 3-contact of vertex, 1-contact of facet.
IIb (a) zero volume of the tetrahedron defined by two facet edges.
(b) 2-contact of each edge.
III (a) zero distance from an EF-vertex to the plane of a facet.
(b) 2-contact of EF-vertex edge, 1-contact of EF-vertex facet, 1-contact of facet.
IV (a) zero distance between two FFF-vertices along an FF-edge.
(b) four 1-contacts of the facets incident on the FFF-vertices.

Fig. 7. Structure change conditions (a) and a-polys (b).

and k swap on the facet edge df when it hits the FF-edge xy, and the FFF-vertex p appears. IV. The FFF-vertices of four facets swap position along their FF-edges when the facets intersect at a vertex. In Fig. 6, three facets intersect $abcd$ in FF-edges ps, qt, and ur, FFF-vertices i and j swap on ur, j and k swap on qt, and i and k swap on ps.

Structure change a-polys. Structure changes occur at zeros of a-polys. The Type I facet interval a-polys are discussed above. Figure 7 describes the a-polys of the other types.

Sweep algorithm. We construct the subdivision of a contact set by constructing the subdivision of each contact facet at its initial t value, sweeping along its t interval, computing the t values where the subdivision undergoes structure changes, and updating the structure. The sweep state is the ordered list of EF-vertices along each contact edge, the set of interior EF-vertices, the set of FF-edges, and the ordered list of FFF-vertices along each FF-edge. The sweep events are the angles where (1) EF-vertices and (2) FF-edges appear or disappear (at a Type I or II structure change) (3) an internal EF-vertex hits an FF-edge (III), (4) two EF-vertices swap on a contact edge (III), or (5) two FFF vertices swap on an FF-edge (IV). FFF-vertices can appear and disappear at events 2, 3 or 4. Events 1 and 2 are calculated before the sweep, event 3 is calculated at a 1 or 2 event, and events 4 and 5 are calculated when two vertices become adjacent on an edge. To calculate event

3, we compute the zeros of its a-poly (Fig. 7III). When sweeping through each zero, we check if the FF-edge and EF-vertex exist at that angle and evaluate predicates at that angle to determine if the EF-vertex hits the line of the FF-edge between its current endpoints, and if so, where in the list of FFF-vertices the new one should be inserted. Similarly for 4 and 5.

Handling identities. Each predicate has the same zero set as an a-poly. For example, testing if a contact edge pierces a contact facet involves testing an (endpoint) contact vertex and the contact facet. The associated a-poly contains the vertex 3-contact and the facet 1-contact. Before evaluating a predicate at an event angle, we check for identity. An identity results from evaluating the predicate at a parameter t that is a zero of a $k \geq 1$ repeated irreducible factor of the a-poly of the predicate. We replace the sign of the predicate with the sign of its kth derivative, evaluated using automatic differentiation. The derivative gives the sign of the predicate value immediately after the event, as required by the sweep algorithm.

9. Results

We measure the running time of predicate evaluation using the factoring algorithm (Sec. 6) for identity detection and compare it to using the greatest common divisor (GCD) for degeneracy detection.

We compute the GCD with Euclid's algorithm. The main step is polynomial division. We compute the degree of a remainder by finding its nonzero coefficient of highest degree. If the sign of a coefficient is ambiguous in interval arithmetic, we evaluate modulo several 32-bit primes. By the Chinese Remainder Theorem (CRT), if it is zero modulo a sufficient number of primes, it is zero. The a-polys have degree at most 9 in the input coordinates (Sec. 4). The leading coefficient is at most degree 9 in the polynomial coefficients. Hence, CRT requires $\lceil 9 \cdot 9 \cdot 53/32 \rceil = 135$ modulo evaluations. This analysis assumes that all inputs have the same exponent field and can be replaced by their 53-bit integer mantissas.

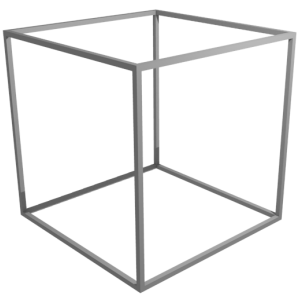
In our first set of tests, we selected 100,000 representatives at random and instantiated them on a pool of 12 O vertices and 12 R vertices with random coordinates in $(-1, 1)$. We factored the univariates of these a-polys, isolated the zeros of the factors, and stored them in a red-black tree.

Let t_h be the ith largest zero of h, a factor of a random a-poly g. When inserting t_h into the tree, it must be compared to prior zeros, such as the jth zero t_e of e, a factor of f. If $e = h$ and $i = j$, then $t_h = t_e$. To measure the running time of the identity detection algorithm, we add an unnecessary test if $g(t_e)$ is an identity. This ensures identity tests on random polynomials with both possible answers. Adding these 9,660,759 identity tests (221,252 positive) increases the running time from 6.6 seconds to 18.1 seconds, giving an average identity detection time of 1.2 microseconds.

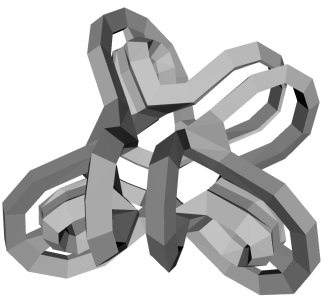
To test the GCD approach, we drop factoring and the equality test, and replace each polynomial f with its square-free form $f/\mathrm{GCD}(f, f')$. If the comparison of zeros

t_f of f and t_g of g is ambiguous in double precision, we run the following degeneracy test on $g(t_f)$. Set $h \leftarrow \mathrm{GCD}(f, g)$ and $e = f/h$. If $e(t_f)$ is unambiguously nonzero, $g(t_f)$ must be zero. If $e(t_f)$ and $g(t_f)$ are both ambiguous, redo these steps with more precision. The additional time was 376 seconds for 81264 degeneracy tests, for an average time of 4627 microseconds. To be sure that $t_f = t_g$ and not some other zero of g, we must check that its comparison with other zeros is unambiguous, so the true cost of the GCD method is even higher.

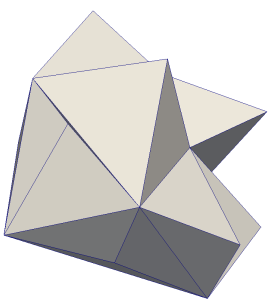
In the second set of tests, we ran our sweep algorithm (Sec. 8) on the polyhedra shown in Fig. 8. Table 1 shows the average running times for sweeping a facet using factor-based identity detection and GCD degeneracy detection. We sweep all the



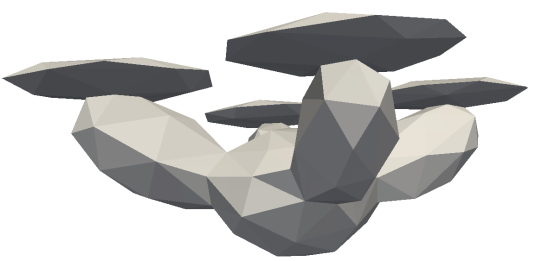
frame: 40 vertices and 96 facets



knot: 480 vertices and 992 facets



simple: 18 vertices and 32 facets



drone: 189 vertices and 374 facets

Fig. 8. Sweep Inputs. Actual frame and knot are large enough for simple and drone to fly through.

Table 1. Sweep algorithm: f total number of contact facets, t_{fac} and t_{GCD} average running time in seconds for sweeping a facet with our identity detection and GCD-based degeneracy detection.

test	O	R	f	t_{fac}	t_{GCD}	$t_{\mathrm{GCD}}/t_{\mathrm{fac}}$
1	frame	simple	2196	0.021	0.533	25.33
2	frame	drone	18812	0.148	2.093	14.13
3	knot	simple	23419	0.023	0.651	27.24
4	knot	drone	235789	0.037	0.857	23.26

facets (tests 1-3) or a large random sample over all angles (test 4). The first tests indicate that factor-based identity detection is 3500 times faster than GCD-based degeneracy detection. The sweep tests show that this speedup reduces the sweep time by a factor of 14 or more. Factor-based identity detection uses less that 0.5% of the overall running time, versus 90% for GCD-based degeneracy detection.

10. Discussion

We have shown that looking up the factors of an a-poly is much faster than polynomial algebra for zero detection. As an additional advantage, factorization provides a unique representation of each algebraic number as the ith zero of an irreducible polynomial.

In future work, we will extend the sweep algorithm to completely construct the subdivision of a contact set. We are missing the surfaces that bound the cells and their nesting order. We will construct a connected component of the free space boundary by visiting the neighboring contact facets, computing their subdivisions, and so on. All the predicates are angle polynomials evaluated at zeros of angle polynomials.

For efficient free space boundary construction, we must eliminate irrelevant contact facets from consideration and must eliminate irrelevant sweep angles for relevant contact facets. One strategy is to construct a polyhedral inner and outer approximation of R as it sweeps through a small angle. For that angle range, the boundary of the rotational free space lies between the boundaries of the translational free spaces of the approximations. We use our fast polyhedral Minkowski sum software to generate approximations of the rotational free space boundary for a set of angle intervals covering the unit circle.[6,10] We see no reason that sweeping a relevant facet should have fewer identities than sweeping an irrelevant facet, and so fast identity detection should provide the same speedup as observed in Sec. 9.

We conclude that our identity detection algorithm is useful for the drone–warehouse problem. But does the technique generalize to other domains? We discuss three challenges.

Factor table construction (Sec. 7) depends on the property that the set of polynomials is closed under factorization. What if this is not true for some alternate class of polynomials? We would realize that something was missing when factor table construction failed due to unmatched univariates. We would then analyze the failure to uncover the missing polynomials. (This is how we discovered the first alternate form for a minor a-poly.) The new polynomials might also have had unmatched factors, but the process of adding missing factor polynomials must converge because factoring reduces degree.

Representative generation (Sec. 6) depends strongly on the a-poly representation. However, the approach should generalize. A predicate has symmetries on its inputs that leave it alone or flip its sign. Two invocations of a predicate function (with repeated inputs) are isomorphic if one can get from one to the other by

applying those symmetries and reindexing inputs. A representative is the isomorphism class member that is lexicographically minimal.

Generalizing the matching of factors to prior classes for table generation would greatly increase the running time because it uses enumeration of permutations. In free space construction for R with d degrees of freedom, the computational complexity is $(3d!)d^4$, which would balloon to 80,000 years for $d = 6$ (unconstrained rotation and translation). We could use the "Birthday paradox" to match two polynomials by generating the square root of the number of permutations for each one and finding a collision, reducing the running time by a factor of $\sqrt{3d!}$ to less than a day. Moreover, these enumerations can be tested in parallel.

Greater complexity might also increase the time required to construct a representative (Sec. 6) for identity detection, but this cost can be reduced by using a subset of the symmetry group and increasing the size of the lookup table. For example, if we stored all the canonical a-polys instead of just the class representatives, the table would have 972,806 entries, which is unproblematic for current computers.

Acknowledgments

Victor Milenkovic was supported by NSF 1526335. Elisha Sacks and Nabeel Butt were supported by NSF 1524455.

References

1. D. Halperin, Controlled perturbation for certified geometric computing with fixed-precision arithmetic, in *ICMS* (2010), pp. 92–95.
2. J. T. Schwartz, Fast probabilistic algorithms for verification of polynomial identities, *J. ACM* **27** (1980) 701.
3. Cgal, Computational Geometry Algorithms Library, http://www.cgal.org.
4. P. Hachenberger, Exact Minkowski sums of polyhedra and exact and efficient decomposition of polyhedra into convex pieces, *Algorithmica* **55** (2009) 329.
5. N. Mayer, E. Fogel and D. Halperin, Fast and robust retrieval of Minkowski sums of rotating convex polyhedra in 3-space, *Computer-Aided Design* **43** (2011) 1258.
6. M.-H. Kyung, E. Sacks and V. Milenkovic, Robust polyhedral Minkowski sums with GPU implementation, *Computer-Aided Design* **67–68** (2015) 48.
7. V. Milenkovic, E. Sacks and S. Trac, Robust free space computation for curved planar bodies, *IEEE Trans. Automation Sci. Engin.* **10** (2013) 875.
8. E. Sacks, N. Butt and V. Milenkovic, Robust free space construction for a polyhedron with planar motion, *Computer-Aided Design* **90C** (2017) 18.
9. E. Sacks and V. Milenkovic, Robust cascading of operations on polyhedra, *Computer-Aided Design* **46** (2014) 216.
10. 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 30th Canadian Conference on Computational Geometry* (*CCCG 2018*) (2018), pp. 223–229.