Purdue University
Purdue e-Pubs

Department of Computer Science Faculty Publications

Department of Computer Science

2011

Controlled linear perturbation

Elisha P. Sacks
Purdue University, eps@cs.purdue.edu

Victor Milenkovic
University of Miami

Min-Ho Kyung
Ajou University

Follow this and additional works at: http://docs.lib.purdue.edu/cspubs

 Part of the Computer Sciences Commons

Repository Citation

Sacks, Elisha P.; Milenkovic, Victor; and Kyung, Min-Ho, "Controlled linear perturbation" (2011). *Department of Computer Science Faculty Publications.* Paper 1.
http://docs.lib.purdue.edu/cspubs/1

Comments

Paper published as: Sacks E., et al. Controlled linear perturbation. Computer-Aided Design (2011), http://dx.doi.org/10.1016/j.cad.2011.06.015

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

Controlled Linear Perturbation

Elisha Sacks

Computer Science Department, Purdue University, West Lafayette, IN 47907-2066, USA

Victor Milenkovic

Department of Computer Science, University of Miami, Coral Gables, FL 33124-4245, USA

Min-Ho Kyung

Department of Digital Media, Ajou University, Suwon, South Korea

Abstract

We present a new approach to the robustness problem in computational geometry, called controlled linear perturbation, and demonstrate it on Minkowski sums of polyhedra. The robustness problem is how to implement real RAM algorithms accurately and efficiently using computer arithmetic. Approximate computation in floating point arithmetic is efficient but can assign incorrect signs to geometric predicates, which can cause combinatorial errors in the algorithm output. We make approximate computation accurate by performing small input perturbations, which we compute using differential calculus. This strategy supports fast, accurate Minkowski sum computation. The only prior robust implementation uses a less efficient algorithm, requires exact algebraic computation, and is far slower based on our extensive testing.

Keywords: robust computational geometry, perturbation methods

1. Introduction

We present a new approach to the robustness problem in computational geometry and demonstrate it on Minkowski sums of polyhedra. Geometric algorithms reason about the combinatorial properties of geometric primitives that are represented using real parameters. Properties of primitives and relations among primitives are represented as the signs of polynomials, called *predicates,* in their parameters. Algorithms are formulated in the real RAM model: a hypothetical computer that performs arithmetic operations on real numbers in unit time. The *robustness problem* is how to implement the algorithms accurately and efficiently on actual computers.

We adopt the scientific computing paradigm of approximate numerical computation in floating point arithmetic. The resulting robustness problem is that even a tiny

Email addresses: eps@cs.purdue.edu (Elisha Sacks), vjm@cs.miami.edu (Victor Milenkovic), kyung@ajou.ac.kr (Min-Ho Kyung)

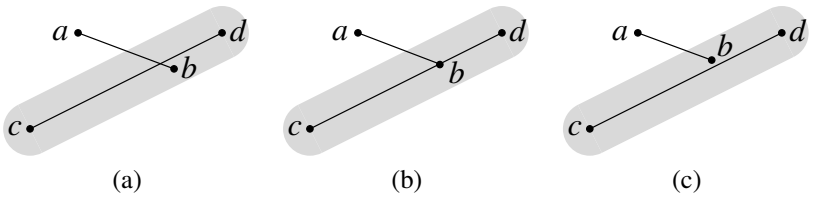


Figure 1: Unsafe input i_u (a), degenerate input i_d (b), and perturbed input i_p (c) for $f(b, c, d)$.

numerical error can cause a predicate to be assigned the wrong sign, which can create a combinatorial error in the algorithm output. For this to occur, the predicate must be *unsafe,* meaning that the magnitude of its value is less than the maximum numerical error. In particular, a *degenerate* predicate, one whose value is zero, is unsafe. For example, plane point p is left of the line from c to d when $f(p, c, d) = (d - c) \times (p - c) > 0$ with $s \times t = s_x t_y - s_y t_x$ (the 2D cross product). In Fig. 1a, $f(a, c, d)$ is safe and $f(b, c, d)$ is unsafe, as indicated by the shaded unsafe zone; in Fig. 1b, $f(b, c, d)$ is both degenerate and unsafe because b is on cd.

We define the error of a geometric algorithm to be the minimum distance from the input, i, to a perturbed input, i_p, such that the (possibly incorrect) predicate signs that were computed for i are correct for i_p. Fig. 1c shows an (unsafe) i_p for $f(b, c, d)$ positive. If an algorithm is implemented using approximate numerical computation, its error can be large even when the numerical error is small. Worse yet, the implementation can calculate predicate signs that violate Euclidean geometry.

There are several prior robustness strategies, but no complete solution to the robustness problem (Sec. 2). Our strategy, called controlled linear perturbation, is to compute a small input perturbation that makes all the predicates safe (Sec. 3). We use differential calculus to compute the perturbation efficiently. Its size is an upper bound on the error of the geometric algorithm, which is the minimal size of a perturbation that realizes the computed predicate signs.

We demonstrate controlled linear perturbation on Minkowski sum computation (Sec. 4). Minkowski sums are a core geometric concept with applications in packing, path planning, and assembly. Although an efficient algorithm has been known for 18 years [1], it has not been implemented because of the robustness problem. We present the first robust implementation and demonstrate that it is fast and accurate. The only prior robust implementation uses a less efficient algorithm, requires exact algebraic computation, and is far slower based on our extensive testing.

2. Prior work

The most common robustness strategy in industry is an empirical set of rules, such as treating nearly equal points as equal. When a new robustness problem arises, a rule is added and the prior rules are revisited to ensure consistency. This strategy cannot lead to reliable general-purpose software. We discuss the algorithmic alternatives.

2.1. Exact computational geometry

The mainstream robustness research strategy is exact algebraic computation [2]. The CGAL computational geometry library [3] implements exact arithmetic on rational numbers and on algebraic numbers. Any algorithm in the library can be executed with exact arithmetic or with floating point arithmetic. There is also extensive research on arrangements of algebraic curves [4].

Exact computation increases bit complexity, hence running time. For example, the product of two b-bit integers has $2b$-bits, as does the sum of two fractions. Prior work reduces the expected running time via floating point filtering and other heuristics [5]. The increase in complexity from input to output leads to unbounded complexity in sequences of computations. One option is to simplify the output while preserving its essential properties [6, 7], but a general simplification strategy is unknown and prior work is limited to a few simple domains.

Degenerate predicates pose a third problem. Computational geometry algorithms are developed under general position assumptions that preclude degeneracy. A robust algorithm must handle these cases, for example by splitting cd at b in Fig. 1b. Explicit handling is tedious for planar problems and is daunting in higher dimensions. The main alternative for exact computational geometry is symbolic perturbation [8, 9], which yields predicate signs that are correct for an arbitrarily small input perturbation.

The complexity, simplification, and degeneracy problems lead us to use approximate computation with error bounds. We are also influenced by the consensus within the scientific computing community that exact solutions to problems in continuous mathematics are pointless because the problems are approximate models of physical laws with approximate parameter values. On the other hand, exact computation can prove theorems, whereas approximate computation can only suggest that they hold.

2.2. Inconsistency sensitive algorithms

One robustness strategy is to modify real RAM algorithms to enforce consistency constraints on the data structures that represent geometric objects. For example, a direct floating point implementation of a real RAM algorithm could correctly calculate that a is above cd in Fig. 2a and that the intersection point of ab and cd is left of b, yet could inconsistently calculate that b is above cd. In Fig. 2b, the inconsistency is removed by splitting cd into four segments. Consistency enforcement increases the computational complexity by a factor that is polynomial in the input size and is linear in the number of inconsistencies. The output error is linear in the computation error and in the number of inconsistencies. We developed inconsistency sensitive algorithms for arrangements of semi-algebraic plane curves [10], manipulation of semi-algebraic plane regions [11], and Minkowski sums of plane regions bounded by line segments and by circular arcs [12, 13]. We tested the algorithms on inputs with many unsafe predicates.

One problem with inconsistency sensitivity is that each algorithm requires its own modification and error analysis, both of which are difficult. Another problem is that the error bounds involve perturbations whose algebraic degree and combinatorial complexity exceed those of the input. In our example, cd is perturbed into four segments. If the degree of cd were greater than one, these segments would have even higher degree.

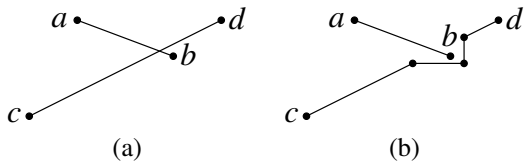


Figure 2: Inconsistency (a) resolved by splitting (b).

2.3. Numerical perturbation

Numerical perturbation is a robustness strategy in which safety is tested using floating point arithmetic and unsafe predicates are made safe by modifying the input. A predicate, $f(x)$, is safe at $x = a$ if $|f(a)| > \epsilon$ with ϵ a function of f, a, and the arithmetic precision [14]. In *controlled perturbation,* the input parameters are perturbed uniformly in δ intervals around their values. The δ is chosen so that the predicates are safe with a high probability. This strategy has been applied to arrangements of polyhedral surfaces [15], arrangements of circles on a sphere [16] and in the plane [17], convex hulls [18], and Delaunay triangulation [18].

Controlled perturbation can change the signs of predicates that are safe for the true input, whereas symbolic perturbation [9] changes no signs. The greater fidelity of symbolic perturbation comes at the price of a sharp increase in the already high cost of exact computational geometry.

There are two problems with controlled perturbation. The δ is chosen *a priori* based on the worst possible input, whereas a much smaller value usually suffices. The δ for a singular predicate (zero value and zero gradient) of degree d is $O(\sqrt[d]{\epsilon})$. This error is unacceptable with the reported ϵ values and is marginal even with ϵ equal to the rounding unit, $\mu \approx 10^{-16}$. For example, the 3D triple product, $a \cdot (b \times c)$, has $d = 3$ and is singular when the points are identical, so $\delta > \sqrt[3]{\mu} \approx 10^{-5}$.

3. Controlled linear perturbation

Controlled linear perturbation (CLP) is a numerical perturbation algorithm that addresses the problems of controlled perturbation. Instead of using an *a priori* δ, CLP computes a minimal δ for each input to the geometric algorithm. It picks a random perturbation direction, v, sets $\delta = 0$, and increases δ each time it encounters a predicate that is unsafe with the current δ. For efficiency, it uses linear estimates of the perturbed parameters and predicates. Safe predicates do not change sign, except in the rare case of a restart (explained below).

Instead of increasing δ to make singular predicates safe, we identify the cases where predicates can be singular and replace them by regular predicates in defined parameters. The parameter definitions are polynomials in the new and current parameters. This strategy reduces the error from $O(\sqrt[d]{\epsilon})$ to $O(\epsilon)$. For example, we achieve $\delta < 10^{-10}$ in Minkowski sum computation by replacing triple products of input vectors with inner products of defined unit vectors (Sec. 4.2).

CLP also improves upon the other prior robustness strategies. Unlike exact computation, there is no extra computational complexity in predicate evaluation and the

Input: $f, a, v, \epsilon, \delta_m, \delta_M$.
1. Set $p = a + \delta_m v$, $s = \text{sign}(f(p))$, and $w = \nabla f \cdot v$.
2. If $sf(p) > \epsilon$
   a. If $sw < 0$, set $\delta_M = \min(\delta_M, \delta_m + (s\epsilon - f(p))/w)$.
   else
   b. Set $s = \text{sign}(w)$ and $\delta_m = \delta_m + (s\epsilon - f(p))/w$.
3. If $\delta_M \leq \delta_m$
   Set $\delta_m = 2\delta_m, \delta_M = \infty$, and restart.
Output: s, δ_m, δ_M.

Figure 3: Predicate evaluation algorithm.

bit complexity is constant. Degeneracy handling involves simple, efficient numerical perturbation instead of complex symbolic perturbation. Singular predicates are replaced with regular predicates instead of triggering higher order expansions. Unlike consistency sensitive algorithms, the real RAM algorithm is not modified, there is no algorithm specific error analysis, and the perturbation has the same algebraic degree and combinatorial complexity as the input.

A random perturbation direction is inappropriate for dependent parameters, such as points on a circle. We propose a treatment of dependent parameters in Sec. 5. Controlled perturbation and symbolic perturbation do not handle dependent parameters.

3.1. Predicate evaluation

CLP assigns signs to a sequence of predicates and computes an interval of δ values for which every predicate is safe. The initial safety interval is $(0, \infty)$. Each predicate is made safe by shrinking the interval, so the prior predicates stay safe. If the interval becomes empty, a restart occurs.

Fig. 3 shows the algorithm. The predicate is f; the input parameter values are a; their perturbation direction, v, is random with v_i uniform in $[-1, 1]$; ϵ is the safety threshold; and (δ_m, δ_M) is the current safety interval. The outputs are the predicate sign, $s = \pm 1$, and the updated safety interval. Step 1 computes the perturbed input, $p = a + \delta_m v$, the default sign, $s = \text{sign}(f(p))$, and the directional derivative, $w = \nabla f \cdot v$ with ∇f the gradient at p. Step 2a handles a safe $f(p)$. If $sw < 0$, $f(p)$ moves toward zero as δ_m increases, so $sf(p) \geq \epsilon$ is necessary for f to stay safe. CLP updates δ_M based on the linear part of this inequality. Step 2b handles an unsafe $f(p)$ by setting $s = \text{sign}(w)$, so $sf(p)$ increases with δ_m, and increasing δ_m to satisfy $sf(p) \geq \epsilon$. Step 3 tests if the safety interval is empty and if so updates it and restarts the parent program.

We illustrate CLP on sorting x_1, x_2, and x_3. The predicate for $x_i < x_j$ is $x_j - x_i$. We set $\epsilon = 2\mu$, $a = (\mu, 0, 1)$, and $v = (-0.5, 0.5, -1)$ for illustration purposes. First, CLP evaluates $x_2 - x_1$ with safety interval $(0, \infty)$. Step 1 sets $f(p) = -\mu$, $s = -1$, and $w = 1$. The predicate is unsafe because $sf(p) = \mu$ and $\epsilon = 2\mu$. Step 2b sets $s = 1$ and $\delta_m = 3\mu$. Second, CLP evaluates $x_3 - x_1$. Step 1 sets $f(p) = 1 - 5\mu/2$, $s = 1$, and $w = -1/2$. Since the predicate is safe and $sw < 0$, step 2a sets $\delta_M = 2 - 6\mu$. Third, CLP evaluates $x_3 - x_2$. Step 1 sets $f(p) = 1 - 3\mu$, $s = 1$, and $w = -3/2$, f is safe, and step 2a sets $\delta_M = 2/3 - 4\mu/3$. The order is $x_1 < x_2 < x_3$ with error bound $\delta_m = 3\mu$.

3.2. Parameter definitions

CLP supports parameter definitions of the form $y = g(x)$ with g a polynomial, x_i^{-1}, or $\sqrt{x_i}$. The parameter y is added to x with perturbation direction $u = \nabla g \cdot v$. Unlike the perturbation direction of an input parameter, u is not restricted to $[-1, 1]$, is large for x_i^{-1} and $\sqrt{x_i}$ with x_i near zero, and can create large w values during predicate evaluation. As a result, a safe predicate with $sw < 0$ (step 2a) can decrease δ_M enough to cause a restart. After the restart, this predicate has the opposite sign, hence cannot cause another restart. A large w benefits step 2b by reducing the δ_m increment.

3.3. Singular unsafe predicates

A singular unsafe predicate is bad for all approaches to robustness. In CLP, the expected value of $w = \nabla f \cdot v$ is of order $\|\nabla f\|$ because v is random. If f is nearly singular, meaning ∇f is near zero, $|w|$ is small, so step 2b sets δ_m to a large value, which greatly increases the output error and often causes a restart.

We classify singularities as artifacts, coincidences, and special cases. Artifacts occur in algorithms that impose extra structure on the input, such as the order along the sweep axis in a sweep algorithm. They can be avoided by randomization. Coincidences occur when combinatorially distinct elements are numerically equal. We handle them by judicious parameter definitions. Special cases occur when the parameters of a predicate are related. We handle them by rewriting the predicate.

3.4. Error analysis

CLP is subject to truncation error due to the linear estimation in predicate and parameter evaluation. The error is $O(\delta_m^2)$ with the constant factor a linear function of the magnitudes of the second derivatives of the predicate or parameter. The constant is small for predicates, since their coefficients are bounded, but can be large for defined parameters. We remove the error by performing an extra restart with δ_m slightly larger than its final value and verifying that δ_m does not increase. There is no error because there is no linear estimation. The restart at most doubles the running time and is only required when a parameter has a large derivative using a conservative threshold.

CLP is also subject to rounding error. The relative rounding error in one arithmetic operation or square root is bounded by μ. The error in a sequence of k operations is exponential in k in the worst case, but is essentially constant in practice. We employ a safety threshold of $\epsilon = 100\mu$ in Minkowski sum computation, which is conservative by numerical analysis standards given that $k < 50$ in that algorithm.

4. Minkowski sums

The Minkowski sum of point sets A and B is $A \oplus B = \{a + b \,|\, a \in A, b \in B\}$. For polyhedra A and B, $A \oplus B$ is a polyhedral region whose boundary polygons are subsets of sum polygons. There are two types of sum polygons: 1) $v \oplus g$ with v a vertex of A and g a face of B or *vice versa* and 2) $e \oplus f$ with e an edge of A and f an edge of B.

Figure 4 shows an example. The Minkowski sum boundary polygons are drawn in white, including $g = \beta\gamma\epsilon$. Polygon g is a subset of the sum polygon $\alpha\beta\gamma\delta$, which is

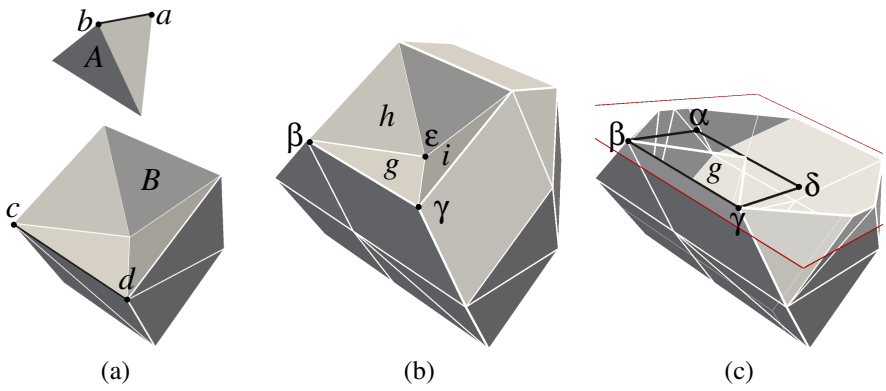


Figure 4: Polyhedra A and B (a), Minkowski sum $A \oplus B$ (b), and cutaway (c).

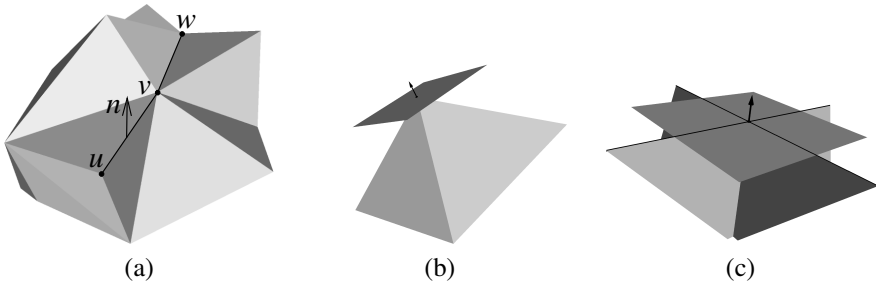


Figure 5: Compatibility: (a) exterior normals; (b) vertex/face, (c) edge/edge.

the sum of edges ab and cd. Edges $\epsilon\beta$ and $\gamma\epsilon$ are the intersections of g with the sum polygons that generate h and i. Vertex ϵ is the common point of g, h, and i.

The efficient Minkowski sum algorithm [1] exploits the fact that the Minkowski sum boundary is a subset of the union of the sum polygons of features with a common exterior normal, called compatible features. A vector, n, at a point, p, on a polyhedron, A, is an exterior normal if p is contained in an open set, O, such that $A \cap O$ is disjoint from the open half space defined by p and n. Figure 5 illustrates. Vertex u and edge uv have exterior normal n, but v, w, and vw have no exterior normals (a). The vertex/face (b) and edge/edge (c) pairs are compatible due to the displayed common normals.

The algorithm is summarized in Fig. 6 and is described in detail by Kaul [1] and by us [13]. The dominant computational cost is polygon intersection. After reviewing prior work (Sec. 4.1), we describe a CLP implementation (Sec. 4.2) and demonstrate that it is accurate and fast (Sec. 4.3).

4.1. Prior work

The only prior robust Minkowski sum implementation [19] reduces the general case to sums of convex polyhedra, which have an exact implementation [20]. It decomposes the input polyhedra into convex components, computes the component sums, and forms their union. The algorithm is slower than the efficient algorithm on polyhedra with

Input: polyhedra A and B.
1. Form a sum polygon for each pair of compatible features.
2. Compute the intersection edges of the sum polygons.
3. Compute the intersection points of the intersection edges.
4. Split each sum polygon along its intersection edges.
5. Group the new polygons into surfaces.
6. Identify the surfaces that form the Minkowski sum boundary.
Output: Minkowski sum boundary.

Figure 6: Efficient Minkowski sum algorithm.

many convex components. A polyhedron with r reflex edges can have $\Omega(r^2)$ convex components [21]. This type of input is typical in mechanical design (e.g. gears), in containment problems (e.g. the interior of a convex container), and in other applications. Not only are there more pairs of convex features than of compatible features, the former tend to have more intersections than the latter. Finally, computing the union of the component sums is a time and memory bottleneck [19]. These costs plus the overhead of exact arithmetic make the prior implementation prohibitively slow, as shown by Campen [22] and by us in Sec. 4.3.

One alternative to the prior implementation is to approximate the union using a volumetric grid [23]. A GPU implementation that computes the outer boundary of the Minkowski sum is very fast [24]. The accuracy is limited by the volumetric resolution: the reported results have a resolution of 1024^3, which yields a 0.1% voxelization error. Increasing the resolution incurs a cubic running time penalty and is limited by the GPU memory size. Another alternative is to compute sample points on the Minkowski sum boundary, but not the boundary topology [25].

Campen [22] computes the outer boundary of the Minkowski sum from the compatible pairs. Although sufficient for some applications, the outer boundary is insufficient for solving containment problems. Points are represented as triples of planes and polyhedra as binary spatial partitions. These ideas lead to low degree predicates for which exact evaluation is practical. Degeneracies are handled explicitly.

4.2. CLP implementation

The CLP implementation consists of definitions and predicates involving points whose coordinates are CLP parameters. A definition is a vector operation: scaling, addition, inner product, cross product, or Euclidean norm, which we write as ka, $a + b$, $a \cdot b$, $a \times b$, and $\|a\|$ with k a parameter and with a and b points. Each definition stands for a short sequence of CLP parameter definitions. For example $k = \|a\|$ stands for $d = a_x^2 + a_y^2 + a_z^2$ and $k = \sqrt{d}$. We use the notation $\hat{a} = a/\|a\|$.

We discuss the interesting definitions and predicates; the rest are standard and singularity free. There are no artifact singularities. We handle coincidences by replacing vectors with unit vectors, that is a by $\hat{a}$. We handle special case singularities by exploiting the parameter relationships to derive equivalent regular predicates.

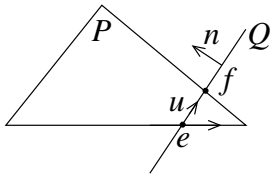


Figure 7: Intersection of polygons P and Q (top view of P).

Polyhedra. The input and output are polyhedra with triangular faces. A polyhedron is represented by its vertices, edges, and faces, and their incidence relations. The CLP input parameters are the coordinates of the vertices. An edge is an open line segment, ab, whose endpoints are vertices. Its tangent is $\widehat{b-a}$. A face is a triangle, abc, with edges ab, bc, and ca. The edges have the face on the left when viewed from outside the polyhedron. The exterior normal is $\widehat{u \times v}$ with u and v the tangents of ab and bc.

Step 1. The compatibility tests are Boolean formulas whose predicates are inner products and triple products of edge tangents, unit vectors $\widehat{u \times v}$ with u and v edge tangents, and face normals. The inner products are regular because their arguments are unit vectors. The triple products are regular because two of the arguments are orthogonal to the third. A vertex, v, and a face, abc, yield the sum polygon $v+a, v+b, v+c$ with the same tangents and normal as abc. Edges ab with tangent u and cd with tangent v yield the sum polygon $a+c, b+c, b+d, a+d$ with normal $n = \widehat{u \times v}$ and with tangents $u, v, -u, -v$. If the common exterior normal is $-n$, switch the edges in the definition.

Step 2. Test every pair of sum polygons, P and Q with normals m and n, for intersection and compute the intersection edges. Compute the points where an edge of one polygon intersects the other polygon. If there are two points, e and f, the intersection edge is ef (Fig. 7); otherwise there are zero points and the polygons are disjoint. Edge bc with tangent u intersects a polygon with point a and normal n if it intersects the plane of the polygon and the intersection point is in the polygon. The edge intersects the plane if $n \cdot (b-a)$ and $n \cdot (c-a)$ have opposite signs. The intersection point is $b+ku$ with $k = n \cdot (a-b)/n \cdot u$. The point in polygon test occurs in a 2D coordinate system obtained by parallel projection along the coordinate axis along which n has the largest magnitude. The test for point p and edge bc with tangent u is $u' \times (p'-b') > 0$ with u', p', and b' the projections of u, p, and b, and with $s \times t$ the 2D cross product.

We rewrite the test to avoid two special case singularities. Polygons that share an edge do not intersect. An edge and a polygon that share a vertex intersect there.

The tangent of the intersection edge, called the intersection tangent of P and Q, is $u = \widehat{m \times n}$. A special case occurs when P has an edge with tangent a and Q has an edge with tangent $\pm a$. In that case, we define $u = \text{sign}(b \cdot n)a$ with b the tangent of the next P boundary edge. These definitions ensure that the projection of n onto the P plane points to the left when the edge is traversed in the u direction (Fig. 7).

Step 3. Test every pair of intersection edges, ab with tangent u and cd with tangent v, for intersection and compute the intersection points. The test uses the 2D projection of

the sum polygon. The edges intersect when each projected edge intersects the line of the other. Projected edge $c'd'$ intersects the line of $a'b'$ when $u' \times (c' - a')$ and $u' \times (d' - a')$ have opposite signs. The edge intersection point is $a + ku$ with $k = (c' - a') \times v' / u' \times v'$.

The predicate $u' \times (c' - a')$ has three special cases; the other predicates are analogous. If a and c lie on the same boundary edge, the predicate is negative or positive when their order is ac or ca. If a, b, and c are boundary vertices, the predicate is negative or positive when their order is acb or abc. The third case is when c is generated by a sum edge, pq, with tangent w, ab is generated by a sum polygon, f, with normal m, and p is on the boundary of f. Since p is on the f plane, c is on the side into which w points, so the predicate is $w \cdot m$.

Step 4. Process each polygon independently. For each boundary edge, sort its intersection points with other polygons. For each intersection edge, sort its intersection points with other intersection edges. For each edge, form a new edge between each consecutive pair of its sorted points. For each new vertex, sort the incident new edges in clockwise order around the face normal. Form new polygons.

The order of points along an edge is defined as follows. Let edge cd with tangent u have intersection points p and q with two polygons whose normals are m and n and that contain points a and b. We have $p = c + \alpha u$ with $\alpha = (a - c) \cdot m / (u \cdot m)$ and $q = c + \beta u$ with $\beta = (b - c) \cdot n / (u \cdot n)$. Point p precedes q along cd if $\beta - \alpha > 0$.

A special case occurs when the polygons share a vertex. Pick $a = b$ and compute

$$\beta - \alpha = \frac{(u \cdot m)((a - c) \cdot n) - (u \cdot n)((a - c) \cdot m)}{(u \cdot m)(u \cdot n)} = \frac{(u \times (a - c)) \cdot (m \times n)}{(u \cdot m)(u \cdot n)}$$

using the identity

$$(A \cdot C)(B \cdot D) - (A \cdot D)(B \cdot C) = (A \times B) \cdot (C \times D).$$

Rewrite the numerator as $(u \times (a - c)) \cdot v = (a - c) \cdot (v \times u)$ with v the intersection tangent of the first plane with respect to the second. Assign the numerator the sign of $(a - c) \cdot w$ with $w = \widehat{v \times u}$. Return the product of this sign and the signs of the denominator terms.

The order of edges around a vertex is defined as follows. If ab is a boundary edge, it precedes every other edge and ba follows every other edge. An intersection edge whose other face is f precedes an intersection edge whose other face is g when $n \cdot u < 0$ with n the face normal and u the intersection tangent of f with respect to g.

Steps 5–6. All predicates are standard without singularities.

4.3. Results

We tested the Minkowski sum algorithm on nine input shapes: 1) cube, 2) glacier, 3) sphere, 4) torus, 5) helix, 6) dragon, 7) knot, 8) small grate, and 9) large grate. Fig. 8 shows sums involving shapes 1–5. Fig. 9 shows shapes 6–7, which are from Lien [25], and shapes 8–9, which are from Varadhan [23]. Table 1 characterizes the input. We computed every sum involving shapes 1–7 and the sum $8 \oplus 9$. Table 2 summarizes the algorithm performance. The complexity of the sum polygons is the total number of vertices, edges, and faces; likewise for the disjoint polygons from step 4, and for

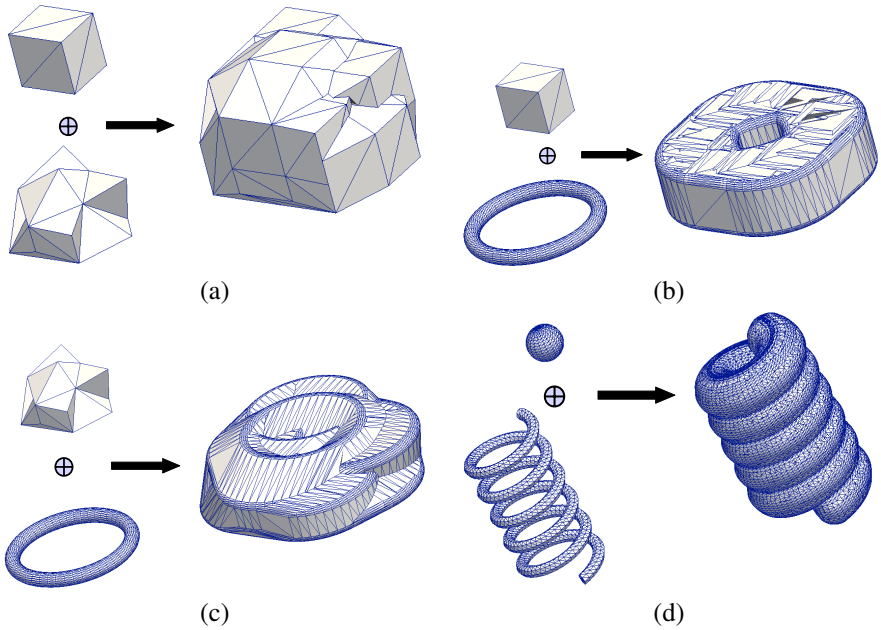


Figure 8: Minkowski sums: (a) $1 \oplus 2$, (b) $1 \oplus 4$, (c) $2 \oplus 4$, (d) $3 \oplus 5$.

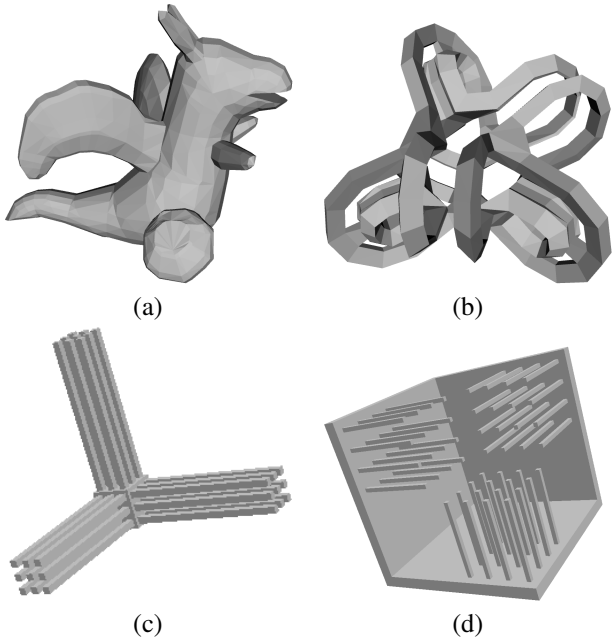


Figure 9: Shapes 6 (a), 7 (b), 8 (c), and 9 (d).

Table 1: Input shapes: i index, v vertices, e edges, f faces, v_c convex vertices, e_c convex edges.

i	v	e	f	v_c	e_c
1	8	18	12	8	16
2	18	48	32	15	36
3	382	1140	760	382	959
4	1000	3000	2000	550	2001
5	2008	6018	4012	1261	3017
6	1166	3492	2328	749	2511
7	480	1488	992	272	924
8	272	810	540	150	480
9	473	1413	942	240	850

Table 2: Results: i input, c, a, and s complexity of sum polygons, disjoint polygons from step 4, and Minkowski sum, t running time in seconds, p predicates, u unsafe ones, and δ_m perturbation size.

i	c	a	s	t	p	u	δ_m
$1 \oplus 1$	195	256	174	0.0	3546	702	8e-13
$1 \oplus 2$	327	493	322	0.0	6472	278	9e-13
$1 \oplus 3$	2718	2718	2718	0.0	88946	2212	2e-12
$1 \oplus 4$	7841	13921	6774	0.1	226803	7943	5e-11
$1 \oplus 5$	19055	65618	10286	0.2	615603	4084	9e-12
$1 \oplus 6$	10492	22988	5778	0.1	328095	1554	5e-12
$1 \oplus 7$	7012	25747	3486	0.1	239907	3942	4e-11
$2 \oplus 2$	685	1900	615	0.0	21495	704	8e-12
$2 \oplus 3$	4644	5640	4628	0.0	138992	526	8e-13
$2 \oplus 4$	11358	16846	8918	0.1	307974	1799	5e-12
$2 \oplus 5$	33547	125688	12940	0.6	1985429	879	1e-12
$2 \oplus 6$	17434	69510	9630	0.3	1159609	347	8e-13
$2 \oplus 7$	12231	68677	4564	0.2	620145	1182	6e-12
$3 \oplus 3$	11331	16386	9804	0.1	585104	17018	7e-12
$3 \oplus 4$	18828	26513	19450	0.2	1033088	3378	3e-12
$3 \oplus 5$	95737	163965	56056	0.8	3293501	3861	1e-13
$3 \oplus 6$	49415	98363	17422	0.5	2296007	4341	1e-13
$3 \oplus 7$	50661	95492	32116	0.5	1619037	4386	6e-12
$4 \oplus 4$	53834	807455	32948	2.2	8305130	29932	2e-08
$4 \oplus 5$	191653	573089	51592	2.2	7007129	35271	2e-11
$4 \oplus 6$	113555	244405	36426	1.2	4240657	11538	1e-12
$4 \oplus 7$	131960	458844	34254	1.9	5842178	18024	1e-11
$5 \oplus 5$	958424	226382422	1076538	733	1233971336	32690	2e-7
$5 \oplus 6$	435734	1239017	88670	5.8	16790484	471	3e-12
$5 \oplus 7$	455818	5349661	47032	23	64594450	35460	7e-12
$6 \oplus 6$	268583	1640537	107954	6.6	18098011	41706	2e-08
$6 \oplus 7$	236143	1249970	62600	7.4	21691347	9047	4e-11
$7 \oplus 7$	214728	10444033	30353	24	65595980	15540	3e-07
$8 \oplus 9$	273947	105894699	130788	130	720398926	50102	1e-5

Minkowski sums. The running time, t, is for one core of an Intel Core 2 Duo with 4 GB RAM.

Our program is very accurate. The error is below 10^{-10}, except for $4 \oplus 4$, $5 \oplus 5$, $6 \oplus 6$, $7 \oplus 7$, and $8 \oplus 9$. The only restarts occur in these tests. Adding a part to itself is hard because every sum polygon is duplicated, which creates $O(n)$ degeneracies. Computing $8 \oplus 9$ is harder because each part has $O(n)$ axis-aligned faces, which creates $O(n^2)$ degenerate sum polygons.

We ran the CGAL [3] version of the prior robust implementation [19] on our tests, using the filtered GMP rational number type. It was 1000–2000 times slower on tests $1 \oplus 1$ to $1 \oplus 7$, 6000 times slower on $2 \oplus 2$, 43,000 times slower on $2 \oplus 3$, and failed on the other tests. Failure means that it aborted or was still running after several hours. Campen [22] and the CGAL manual report similar performance for the CGAL software. Neither reference specifies the number type. We are 15 times faster than Varadhan [23] on $8 \oplus 9$ based on the reported processor speeds. We compute the complete Minkowski sum 3 times faster than Campen [22] computes the outer boundary.

We have developed an implementation that distributes steps 2–5 of the algorithm, which account for 90% of the computation time, over k cores of a 32-core processor. The computation time of the distributed part decreases linearly for small values of k, decreases further up to $k = 16$, then increases. The overall speedup factor for $k = 16$ is 9 for $5 \oplus 5$, 5 for $5 \oplus 7$, 7 for $7 \oplus 7$, and 9 for $8 \oplus 9$.

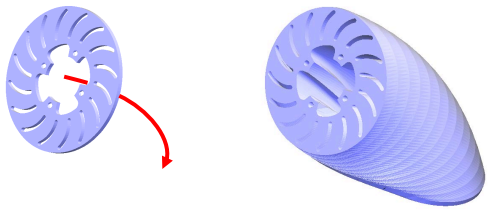
We evaluated the benefit of rewriting singular predicates (Sec. 4.2). Not replacing a with $\hat{a}$ makes all the tests fail with an initial $\delta_m = 10^{-10}$, almost all fail with $\delta_m = 10^{-5}$, and some fail even with $\delta_m = 10^{-3}$. Not rewriting predicates with duplicate arguments causes the five hard tests and some of the other tests to fail.

We evaluated the CLP strategy of computing δ_m incrementally by analyzing a variant that doubles δ_m and restarts on the first unsafe predicate. The number of restarts is $k = \lceil \max(0, \log_2(\delta_f/\delta_i)) \rceil$ with δ_i and δ_f the initial and final δ_m values. Using $\delta_i = 10^{-10}$ yields $k = 9, 10, 12, 12, 18$ for the five hard tests and $k = 0$ for the rest. Linear estimation reduces the running time by an order of magnitude on hard inputs. On inputs without restarts, the variant is slightly faster because it does not compute derivatives.

5. Discussion

We have presented the controlled linear perturbation robustness strategy and have argued that it addresses the problems of prior strategies. We have backed up this argument by developing the first robust implementation of an 18 year old algorithm for Minkowski sums of polyhedra. The implementation solves in seconds a large set of tests that appear beyond the scope of prior work. We have achieved similar results in the CAD related tasks of assembly planning, robot path planning, and swept volume computation. In particular, we compute swept volumes (Fig. 10) by a hybrid of Campen's algorithm [22] and our Minkowski sum algorithm.

We are extending CLP to input parameters that satisfy geometric constraints, such as points on the unit circle. The constraints are k polynomials, $g(x)$. We pick an orthogonal basis, $b_1, \ldots, b_{n-k}$ for the complement of ∇g and set the perturbation direction to $v = \sum_i v_i b_i$ with v_i uniform in $[-1, 1]$. We compute the perturbed parameter values



(a) (b)

Figure 10: Rotor with sweep path (a) and swept volume (b).

with an iterative numerical solver. As constraints are added, the space of perturbations shrinks, so the set of singular predicates grows. We need to study this phenomenon.

We want CLP to support parameter definitions in which parameters $y = (y_1, \ldots, y_k)$ are defined by k equations, $g(x, y) = 0$. Unlike an explicit definition, the parameter values must also be specified to identify a unique component of the g variety. The y perturbation direction, u, is computed from the linear equations $g_x v + g_y u = 0$ where g_x and g_y are the Jacobian matrices of g with respect to x and y. The definition is inaccurate when g_y is ill-conditioned and $\|u\|$ can be large. Handling these cases with restarts can be inaccurate and fails when the definitions are invalidated by increasing δ_m, such as $y = \sqrt{x_i}$ with a_i a small positive number and v_i negative.

We will develop a general treatment of parameter definitions based on singularity theory. Parameters are δ-monotone curves in (δ, x) space and singularities occur at isolated points where curves meet. The only generic case involves two real roots that merge and become complex. In the canonical example, $g(x, y) = x - y^2$ with $a = 0$ and $v = 1$, the roots are $y = \pm\sqrt{\delta}$ for $\delta > 0$ and the merge occurs at $\delta = 0$. Other singularities are possible in symmetric definitions.

We aim to automate the handling of singular predicates. We will generalize the strategy that replaces a by $\hat{a}$ to handle any rank deficient determinant predicate. We plan to replace f with a regular polynomial, f^*, and to equate the signs of f and f^* with equality constraints. The constraints and f^* contain new parameters whose values are chosen to make f^* regular. We conjecture that any singular predicate due to duplicated parameters can be expressed in terms of predicates of lower degree without duplicated parameters. We aim to automate this process.

CLP fails when δ_m grows too large because there are too many unsafe predicates. We envision a restart strategy in which the unsafe predicates with the smallest derivatives in the v direction are assigned better velocities, perhaps via linear programming.

We need to extend our error analysis to cover input constraints and general parameter definitions. We will use second derivatives to bound the truncation errors. The bounds will replace the final restart. A greater challenge is a rigorous, yet practical treatment of rounding error. One option is to adjust the floating point precision to match the worst case rounding error, using an arbitrary precision floating point library, such as MPFR [26]. Another option is to derive probabilistic error bounds by comparing the predicate signs due to several perturbations.

Another research direction is to improve the distributed implementation. The first

step is to accelerate sum polygon construction. Further improvement requires major changes in the algorithm to address the fundamental challenges of contention and of limited memory band width.

Vitae

Elisha Sacks is a professor in the Department of Computer Science of Purdue University. His research interests are Computational Geometry, Computer Graphics, and Mechanical Design.

Victor Milenkovic is a professor in the Department of Computer Science of the University of Miami where he has been on the faculty since 1994. His interests are Computational Geometry, Packing and Nesting, Graphics, and Visualization.

Min-Ho Kyung is an associate professor in the Department of Digital Media at Ajou University. His research interests are Computer Graphics and Computational Geometry.

Acknowledgments

Thanks to Eric Fischer, a Purdue undergraduate, for testing the CGAL Minkowski sum software. Milenkovic supported by NSF grant CCF-0904707. Sacks supported by NSF grant CCF-0904832 and by a PLM grant. Kyung supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (2010-0012021).

References

[1] A. Kaul, M. A. O'Connor, Computing minkowski sums of regular polyhedra, Tech. Rep. RC 18891, IBM Research Division, Yorktown Heights (1993).

[2] C. Yap, Robust geometric computation, in: J. E. Goodman, J. O'Rourke (Eds.), Handbook of discrete and computational geometry, 2nd Edition, CRC Press, Boca Raton, FL, 2004, Ch. 41, pp. 927–952.

[3] Computational geometry algorithms library, http://www.cgal.org/.

[4] A. Eigenwillig, M. Kerber, Exact and efficient 2d-arrangements of arbitrary algebraic curves, in: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA08), 2008, pp. 122–131.

[5] Exact computational geometry, http://cs.nyu.edu/exact.

[6] S. Fortune, Polyhedral modelling with multiprecision integer arithmetic, Computer-Aided Design 29 (2) (1997) 123–133.

[7] S. Fortune, Vertex-rounding a three-dimensional polyhedral subdivision, Discrete and Computational Geometry 22 (1999) 593–618.

[8] H. Edelsbrunner, E. P. Mücke, Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms, ACM Transactions on Graphics 9 (1) (1990) 66–104.

[9] I. Emiris, J. Canny, R. Seidel, Efficient perturbations for handling geometric degeneracies, Algorithmica 19 (1–2) (1997) 219–242.

[10] V. Milenkovic, E. Sacks, An approximate arrangement algorithm for semi-algebraic curves, International Journal of Computational Geometry and Applications 17 (2).

[11] V. Milenkovic, E. Sacks, Approximate planar shape manipulation, submitted to *International Journal of Computational Geometry and Applications* (2010).

[12] V. Milenkovic, E. Sacks, A monotonic convolution for Minkowski sums, International Journal of Computational Geometry and Applications 17 (4) (2007) 383–396.

[13] V. Milenkovic, E. Sacks, Two approximate minkowski sum algorithms, International Journal of Computational Geometry and Applications 20 (4) (2010) 485–509.

[14] C. Burnikel, S. Funke, M. Seel, Exact arithmetic using cascaded computation, in: Proceedings of the Symposium on Computational Geometry, 1998, pp. 175–183.

[15] S. Raab, Controlled perturbation for arrangements of polyhedral surfaces, in: Proceedings of the 15th Symposium on Computational Geometry, ACM, 1999, pp. 163–172.

[16] D. Halperin, C. Shelton, A perturbation scheme for spherical arrangements with application to molecular modeling, Computational Geometry: Theory and Applications 10 (4) (1998) 273–288.

[17] D. Halperin, E. Leiserowitz, Controlled perturbation for arrangements of circles, International Journal of Computational Geometry and Applications 14 (4–5) (2004) 277–310.

[18] S. Funke, C. Klein, K. Mehlhorn, S. Schmitt, Controlled perturbation for delaunay triangulations, in: SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2005, pp. 1047–1056.

[19] P. Hachenberger, Exact minkowski sums of polyhedra and exact and efficient decomposition of polyhedra into convex pieces, Algorithmica 55 (2009) 329–345.

[20] E. Fogel, D. Halperin, Exact and efficient construction of Minkowski sums of convex polyhedra with applications, Computer-Aided Design 39 (11) (2007) 929–940.

[21] B. M. Chazelle, Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm, SIAM Journal on Computing 13 (3) (1984) 488–507.

[22] M. Campen, L. Kobbelt, Polygonal boundary evaluation of Minkowski sums and swept volumes, Eurographics Symposium on Geometry Processing 29 (5).

[23] G. Varadhan, D. Manocha, Accurate minkowski sum approximation of polyhedral models, Graphical Models 68 (4) (2006) 343–355.

[24] W. Li, S. McMains, a GPU-based voxelization approach to 3D minkowski sum computation, in: J. Keyser, M.-S. Kim (Eds.), Solid Modeling, ACM, 2010, pp. 31–40.

[25] J.-M. Lien, Covering minkowski sum boundary using points with applications, Computer Aided Geometric Design 25 (2008) 652–666.

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