Robust approximate assembly partitioning

Elisha Sacks* Victor Milenkovic† Yujun Wu‡

Abstract

We present a robust approximate assembly partitioning algorithm for polyhedral parts. We achieve robustness by applying our controlled linear perturbation strategy to Minkowski sums of polyhedra and to arrangements of great circle arcs. Our algorithm is far faster than a prior robust algorithm based on exact computational geometry. Its error is small even on degenerate input.

1 Introduction

We present a robust approximate assembly partitioning algorithm. Given a set of polyhedral parts, the task is to find a direction in which a subset of the parts can translate unboundedly without touching the other parts. Assembly partitioning is a key step in the larger task, called assembly planning, of devising a sequence of coordinated part translations and rotations that builds an assembly from a set of parts. An efficient assembly partitioning algorithm is crucial because assembly planning is computationally intractable. Halperin [7] presents a real RAM algorithm for generic input. Actual input is typically degenerate because useful parts usually have symmetric features. The *robustness* problem is how to implement the algorithm accurately, efficiently, and for any input.

Fogel [4] uses exact computational geometry [9] to implement Halperin's algorithm. Error is avoided by exactly evaluating polynomials in the input parameters, called predicates, whose signs determine the output. Although most predicates can be evaluated quickly via floating point filtering [1], near-zero predicates require expensive rational arithmetic. Typical assembly partitioning tasks have many such predicates, which makes Fogel's approach slow. Degenerate (zero value) predicates require explicit handling, which complicates the algorithm. Exact computation also increases bit complexity, hence memory use, which is the computational bottleneck for large inputs.

We [8] advocate an alternate robustness strategy, called controlled linear perturbation (CLP), based on approximate computation with floating point arithmetic. CLP uses differential calculus to compute a small input perturbation that makes the output accurate. The running time is insensitive to near-zero predicates, degeneracy handling is avoided, and the bit complexity is low. We use CLP to implement the assembly partitioning algorithm (Sec. 2). The computational geometry steps are Minkowski sums, using our prior algorithm [8], and arrangements of great circle arcs, using a plane sweep algorithm (Secs. 3–4). We demonstrate that our algorithm is far faster than its exact counterpart (Sec. 5). Its error is small even on degenerate input. We conclude with a discussion of the two robustness strategies (Sec. 6).

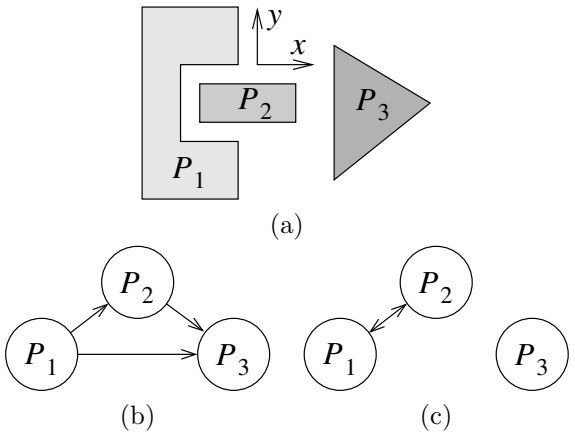

Figure 1: Assembly (a) and directional blocking graphs for x (b) and y (c).

2 Assembly partitioning algorithm

The input to the algorithm is n disjoint polyhedral parts, $A = \{P_1, \ldots, P_n\}$. Let $P_i + v = \{p + v | p \in P_i\}$ denote the translation of P_i by the vector v. A motion direction is represented by a unit vector. The direction d is free for $\langle P_i, P_j \rangle$ if $(P_i + kd) \cap P_j = \emptyset$ for every $k \geq 0$; otherwise d is blocked for $\langle P_i, P_j \rangle$. Part P_i can translate unboundedly along a free d without hitting P_j, but not along a blocked d. We seek a proper subset, $S \subset A$, and a direction, d, that is free for every $\langle P_i, P_j \rangle$ with $P_i \in S$ and $P_j \notin S$. In Fig. 1a, $\hat{x}$ is free for $\langle P_2, P_1 \rangle$, and blocked for $\langle P_1, P_2 \rangle$ and $\langle P_2, P_3 \rangle$. One solution is $S = \{P_3\}$ and $d = \hat{x}$; another is $S = \{P_1, P_2\}$ and $d = \hat{y}$.

The algorithm for a fixed d is combinatorial. Form the *directional blocking graph* with a node for each part and with a link from P_i to P_j if d is blocked for $\langle P_i, P_j \rangle$. If

*Department of Computer Science, Purdue University, eps@cs.purdue.edu
†Department of Computer Science, University of Miami
‡Department of Computer Science, Purdue University

the graph is strongly connected, there is no solution because every $S \subset A$ has a link from some $P_i \in S$ to some $P_j \notin S$. Otherwise, any component without outgoing links is a solution. In our example, the $\hat{x}$ graph components are $\{\{P_1\}, \{P_2\}, \{P_3\}\}$ and the $\hat{y}$ graph components are $\{\{P_1, P_2\}, \{P_3\}\}$ (Fig. 1).

The assembly partitioning algorithm computes a subdivision of the unit sphere such that all the directions in each face have the same graph. It traverses the faces of the subdivision, analyzes their graphs, and returns the first solution or reports failure. The subdivision is computed in two steps. 1) Partition the unit sphere into free and blocked faces for each $\langle P_i, P_j \rangle$. The graph has a link from P_i to P_j for d in the blocked faces of $\langle P_i, P_j \rangle$. 2) Compute the overlay of the partitions.

We illustrate the algorithm on an assembly comprised of ring P_1, ring P_2, and cone P_3 with axis $z = (0, 0, 1)$ (Fig. 2). Faces e–g of the overlay are in the northern hemisphere, h straddles the equator, and the southern hemisphere is symmetric. Face e has solutions with $S = \{P_1\}$ (Fig. 2c). Face f has one more link, from P_2 to P_3, and also has solutions with $S = \{P_1\}$. Face g has no solutions (Fig. 2d). Face h has one less link, from P_2 to P_1, and no solutions.

Figure 3 illustrates step 1 of the algorithm in 2D. A direction, d, is blocked for $\langle P_i, P_j \rangle$ if the ray kd intersects the Minkowski sum

$$M_{ij} = P_j \oplus -P_i = \{a - b | a \in P_j, b \in P_i\},$$

which comprises the vectors, v, such that $P_i + v$ intersects P_j. Let Q_{ij} denote the projection of M_{ij} onto the unit sphere: a vector, v, projects to $\hat{v} = v/||v||$. The projection is defined because the parts are disjoint, so $(0, 0, 0) \notin M_{ij}$. The connected components of Q_{ij} are the blocked faces of $\langle P_i, P_j \rangle$. We compute them for $i < j$ and handle $Q_{ji} = -Q_{ij}$ by symmetry.

Figure 4 illustrates projection. The boundary of Q is a subset of the projected silhouette edges of M. Let $e = ab$ denote an edge of M with tail a and head b; its twin is the edge with tail b and head a. Let e have tangent u, and faces to the left and right with outward normals m and n. If $a \cdot m > 0$ and $a \cdot n < 0$, e is a silhouette edge. Project it to the arc, $\hat{e}$, with tail $\hat{a}$ and head $\hat{b}$ (Fig. 4a,c) on the great circle defined by the plane with normal $\widehat{a \times b}$. Label the silhouette arcs positive and label their twins negative. Compute the induced subdivision of the unit sphere. A face is in Q if its boundary contains a positive edge or if the ray kd intersects M for any point, d, in its interior. The blocked faces of Q are bounded by the edges that bound Q, but whose twins do not.

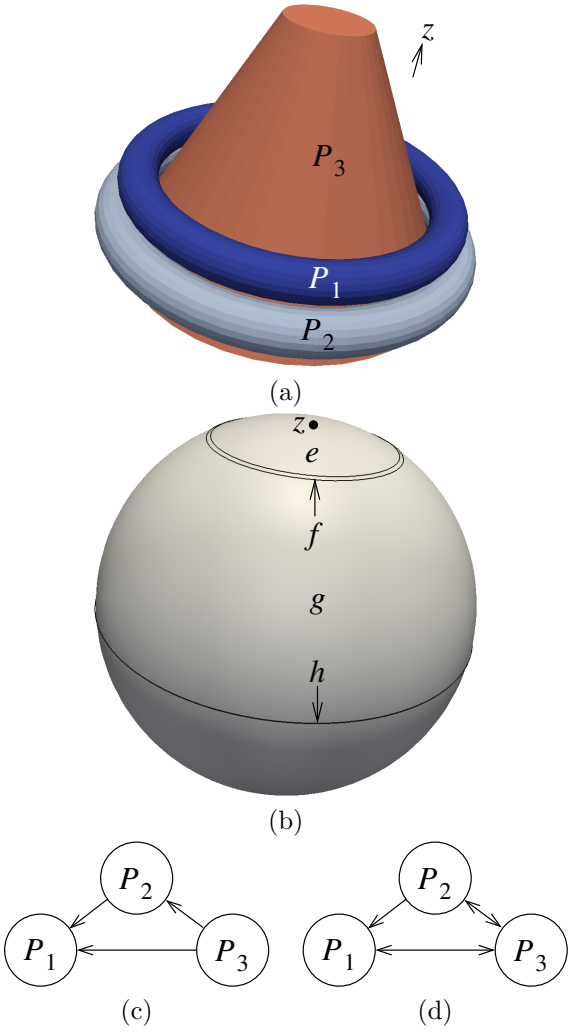

Figure 2: Ring assembly (a), overlay (b), and directional blocking graphs for faces e (c) and g (d).

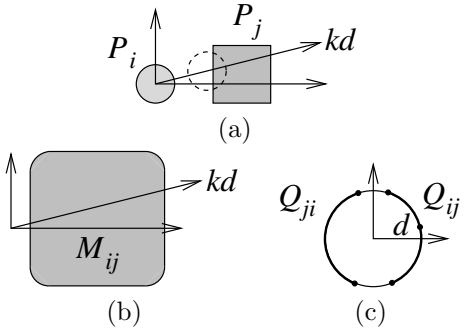

Figure 3: Partition: (a) parts; (b) Minkowski sum; (c) projection.

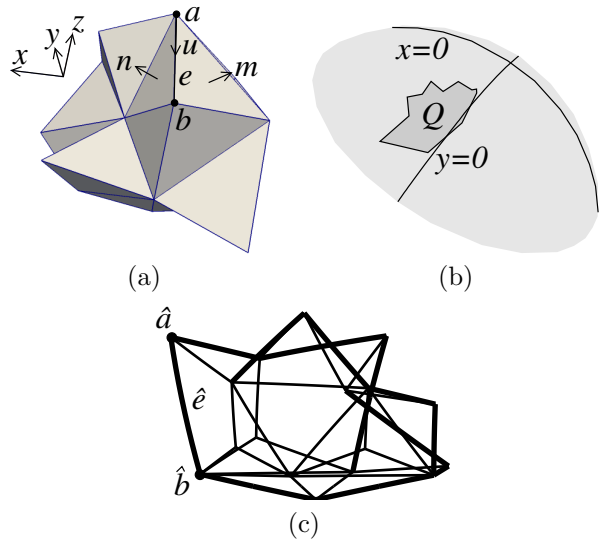

Figure 4: Projection: (a) M; (b) Q; (c) projected edges with silhouette edges drawn thickly.

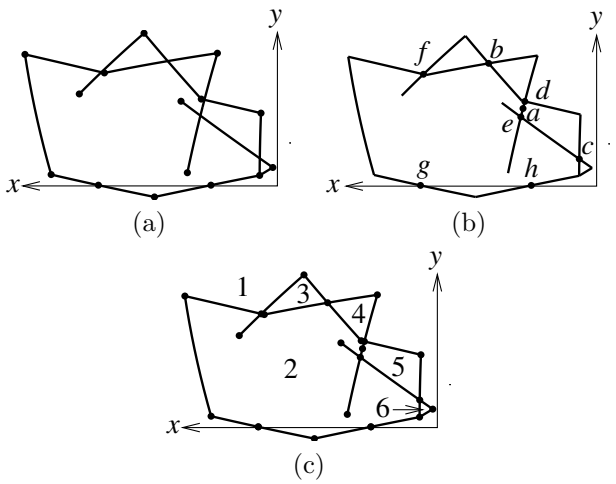

Figure 5: Arrangement algorithm: (a) input arcs; (b) split points; (c) faces.

3 Arrangement algorithm

We compute arrangements of great circle arcs for projection and for overlay. The algorithm is a plane sweep, which splits the arcs at their intersection points to obtain the vertices and edges of the arrangement, followed by a traversal of the vertex/edge graph, which derives the faces. Fig. 5 illustrates on the Fig. 4 example.

Preprocessing Split the input arcs (Fig. 5a) at z turning points (a in Fig. 5b). An arc, $e = ab$, with normal n has a turning point if $u_z v_z < 0$ with $u = \widehat{n \times a}$ and $v = \widehat{n \times b}$ the tangents at a and b. If $u_z > 0$, e has a maximum at $\hat{p}$ with $p = \text{sign}(n_z)(-n_x, -n_y, 1/n_z - n_z)$; if $u_z < 0$, e has a minimum at $-\hat{p}$. Splitting the arcs yields z-monotone edges. Split the edges at intersection points, q, with the great circle with normal $(0, 1, 0)$ such that $q_x < 0$ (g and h in Fig. 5b).

Place the incident edges of each vertex, a, in clockwise order around the outward normal. An edge, $e = ab$, is *forward* or *backward* if $a_z < b_z$ or $b_z < a_z$. Forward edges precede backward edges. An edge with normal m precedes one with normal n if $a \cdot (u \times v) < 0$ with $u = \widehat{m \times a}$ and $v = \widehat{n \times a}$ the tangents at a. This predicate is identically zero if a is a z turning point because $u = -v$. Instead, the positive edge precedes the negative edge for $a_z n_z > 0$ and *vice versa* for $a_z n_z < 0$.

Sweep Sweep a plane along the z axis from $z = -1$ to $z = 1$. The plane intersects the unit sphere in a circle. The sweep list consists of the forward edges that intersect this circle in counterclockwise order. The events are the input vertices and the intersection vertices (b–f in Fig. 5b). The z order is calculated by comparing vertex z coordinates, except that a always precedes or follows b when a is a z minimum or maximum of $e = ab$.

An input vertex is handled by removing the twins of its backward edges from the sweep list, recording the edge that follows it in the sweep list, inserting its forward edges, and checking if any newly adjacent edges intersect. Two edges cannot intersect if they come from the same input arc. Otherwise, edges $e = ab$ with normal m and $f = cd$ with normal n intersect at an intersection point, $p = \pm\widehat{m \times n}$, of their great circles if their tangents at p, $\widehat{m \times p}$ and $\widehat{n \times p}$, have positive z components and $\max(a_z, c_z) < p_z < \min(b_z, d_z)$. The intersection vertex is handled by splitting e into ap and pb, splitting f into cp and pd, placing the p edges in the order (pd, pb, pc, pa), replacing e by pd and f by pb in the sweep list, and checking the newly adjacent edges.

We represent the sweep order as a linear order on $[-\pi, \pi]$. The transitions between $-\pi$ and π occur at vertices because of the preprocessing. If a transition occurs at a, $e = ab$ is inserted at the start or the end of the sweep list when $b_y < 0$ or $b_y > 0$. Otherwise, e is inserted by repeatedly comparing it to an edge, $f = cd$, in the list. If $a = c$, the sweep order is the counterclockwise order around a. Otherwise, compute the sweep order of a and the intersection point, p, of f with the sweep plane $z = a_z$. Edge e precedes f if $a_y < 0$ and $p_y > 0$ or if $a_y p_y > 0$ and $a_x p_y - a_y p_x > 0$.

Graph traversal Mark the edges as untraversed. Visit each vertex in sweep z order and trace an edge loop starting at each of its untraversed edges. While the current edge, $e = ab$, is untraversed, mark it as traversed and replace it by the successor of its twin among the edges incident on b. For the first vertex or for a vertex with an edge that was traversed before it was visited, each edge loop defines a face. The six faces in Fig. 5c are

generated in this manner in numerical order. Otherwise, the first loop is added to the enclosing face and the other loops define faces. The enclosing face is bounded by the following edge of the vertex, if defined, or by the first edge of the previous vertex.

4 Robustness

A direct floating point implementation of the arrangement algorithm is not robust. Even tiny computation errors 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 its value is on the order of the computation error. The main cause of unsafe predicates is degeneracy. A degenerate input manifests itself as a predicate that evaluates to zero, so approximate computation assigns it an unsafe value.

We prevent unsafe predicates with our controlled linear perturbation (CLP) algorithm [8]. CLP assigns signs, s_i, to a sequence of predicates, $f_i(x)$, with input values $x = a$. It picks a random unit vector, v, and computes a $\delta \geq 0$ such that $s_i f_i(p) > \epsilon$ with $p = a + \delta v$ and with ϵ a safety threshold that depends on f and on a. If $|f_i(p)| > \epsilon$ with the current δ, $s_i = \text{sign}(f_i(p))$; otherwise, $s_i = \text{sign}(w)$ with $w = \nabla f \cdot v$ and with ∇f the gradient, and δ is increased by $(s\epsilon - f(p))/w$ to the minimum value that makes f_i safe based on its linear Taylor series.

We employ the backward error metric: the error in a computation is the minimum distance from the input to a perturbed input for which the output is correct. For a CLP algorithm, the input is a, a perturbed input is p, and the error is at most $||p - a|| = \delta$. We assume that the signs, s_i, are correct at p, which holds when the safety thresholds exceed the predicate rounding error. The rounding error in a single arithmetic operation is bounded by the rounding unit of $\mu \approx 10^{-16}$. The error in a sequence of n operations is exponential in n in the worst case, but is essentially constant in practice. We employ a safety threshold of $\epsilon = 100\mu$, which is conservative by numerical analysis standards given that $n < 50$ in our algorithm.

The sign assignment algorithm performs poorly on singular predicates ($\nabla f = 0$). Singularity is much rarer than degeneracy because both f and ∇f must be zero. Yet a single singular predicate can invalidate the arrangement computation by increasing δ unacceptably. The sweep has singularities when vertices coincide with z turning points. We prevent this by sweeping along a random axis. This strategy suffices for the arrangement algorithm. We discuss a general strategy in Sec. 6.

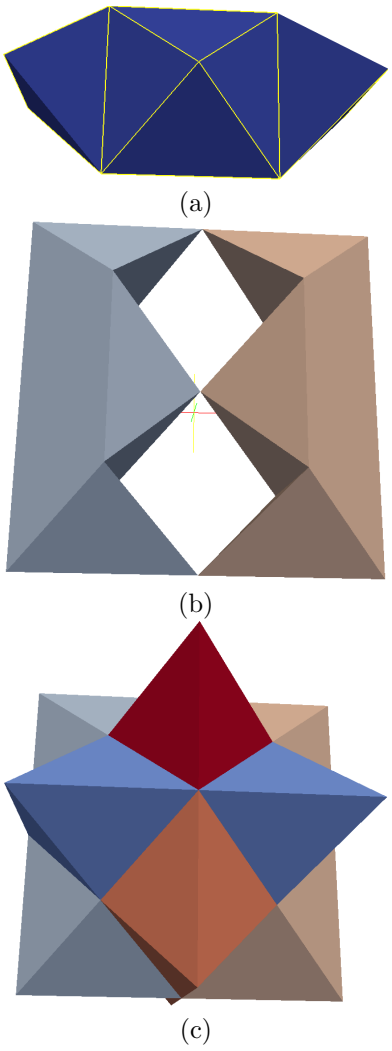

Figure 6: Star puzzle: (a) one part, (b) two parts, (c) all six parts.

5 Performance

We tested our algorithm on Fogel's star puzzle example (Fig. 6). The assembly has six parts that are rotational images of each other. Each part has 14 boundary triangles. The Minkowski sums and the arrangements are degenerate because the parts are symmetric, the pairs consist of isometric parts, and the assembly contains many isometric pairs. Nevertheless, the backward error is only $\delta = 10^{-12}$. The running time, on one core of an Intel Core 2 Duo with 4 GB RAM, is 0.024 seconds with 82% for Minkowski sums, 8% for projection, and 7% for overlay. This is about 100 times faster than Fogel's best time of 5.2 seconds, since our CPU is about 50% faster.

We also tested our algorithm on two engineering examples. The first is the ring assembly (Fig. 2). Each ring has 2068 boundary triangles and the cone has 160. The running time is 3.2 seconds with 94% for the three Minkowski sums. The error is $\delta = 4 \times 10^{-9}$. The second

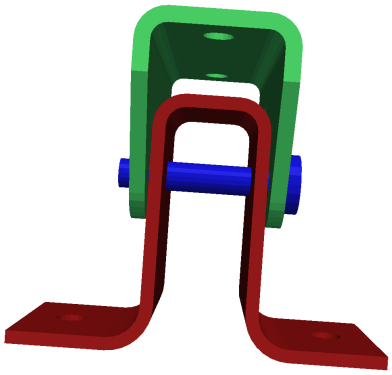

Figure 7: Hinge assembly.

is a hinge assembly comprised of a bolt and two plates (Fig. 7). The bolt has 160 boundary triangles, the inner plate has 596, and the outer plate has 572. The running time is 1.1 seconds with 85% for the three Minkowski sums. The error, $\delta = 10^{-7}$, is larger than before because the parts have many parallel boundary triangles, each of which causes multiple degeneracies.

6 Discussion

The performance of our assembly partitioning algorithm supports our thesis that the CLP robustness strategy is accurate, is far faster than exact computation, and avoids degeneracy handling. One reason for the speedup is that floating point arithmetic is faster than rational arithmetic. A second reason is that we compute Minkowski sums, which are the dominant cost in assembly planning, via a convolution algorithm that is output sensitive in practice. Fogel decomposes the polyhedra into convex pieces by hand, computes the piece sums, and forms their union. Since the pieces have many non-input features, the complexity far exceeds the output size in typical examples. The same is true of a later algorithm that automates the decomposition [6]. We attribute the lack of a robust exact convolution algorithm to the daunting degenerate cases, including collinear faces, identical faces, and edges on faces.

Clearance Fogel's algorithm has the theoretical advantage that it can find solutions where parts have zero clearance, meaning their boundaries intersect and their interiors are disjoint, by examining the degenerate faces of the overlay. We cannot compute faces whose diameter is less than δ. Since the maximal part clearance for directions on a face is proportional to its diameter, we are limited to solutions whose clearance exceeds δ.

We see no practical significance to this limitation. Parts are subject to manufacturing variation and assembly mechanisms are subject to motion variation. An assembly plan must handle all parts and mechanisms of a specified accuracy. A plan with a clearance of 10^{-6} is unsafe for any conceivable accuracy, whereas δ is always far smaller. The standard planning strategy is to replace each ideal part by an expanded part that bounds its shape variation. The simplest and most common replacement is the Minkowski sum of the ideal part and an s-sphere centered at the origin. The existence of an exact solution for $s = k$ implies the existence of an approximate solution for $s = k + \delta$. The solutions are equivalent in practice because δ is negligible with respect to k.

Designers sometimes consider ideal parts before modeling part variation. Assembly plans with zero clearance are then of interest. For example, the star puzzle is more elegant when the parts fit together perfectly. We can approximate zero clearance solutions by dilating the parts by r_1 (subtracting a sphere of radius r_1) until an approximate solution is found, expanding them by r_2 until they overlap, and performing binary search on $[-r_1, r_2]$ for the smallest parts that yield an approximate solution. We implemented this procedure for the star puzzle, but using scaling instead of dilation and expansion. Scaling by 99% yields a solution, scaling by 101% makes the parts overlap, and 7 iterations yield an approximate solution with $\delta = 10^{-12}$ in 0.17 seconds, versus 5.2 seconds for Fogel's fastest run.

Correctness CLP algorithms can fail due to extreme rounding error, whereas exact algorithms cannot. On the other hand, exact algorithms effectively halt when they run out of memory, which already occurs on modest size Minkowski sums. CLP is correct assuming the same empirical bounds on rounding error that underlie every numerical library in the scientific computing community. We have never observed a CLP failure despite extensive consistency checking of every Minkowski sum and spherical arrangement that we compute. We aim to replace this empirical evidence with a rigorous, yet practical error analysis. 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 [5]. Another option is to derive probabilistic error bounds by comparing the predicate signs due to several perturbations.

Algorithm design We conclude with a comparison of algorithm design using CLP versus exact computation. An exact algorithm has to address degeneracy, whereas a CLP algorithm does not. Explicit degeneracy handling appears impractical in most 3D algorithms. The alternative to explicit handling is symbolic perturbation [2, 3], which yields predicate signs that are correct for an arbitrarily small input perturbation. Symbolic perturbation further increases the computational complexity of exact computation. Neither CLP nor exact compu-

tation with symbolic perturbation can solve degenerate problems. We have illustrated that degenerate assembly partitioning problems can be solved approximately with CLP, but the process is not automated.

A CLP algorithm has to address singularity, whereas an exact algorithm with explicit degeneracy handling does not, since singularity is a special case of degeneracy. An exact algorithm with symbolic perturbation has to address singularity because it computes the first non-vanishing derivative of degenerate predicates. The computational complexity rises sharply with the degree of singularity.

We classify singularities as artifacts, coincidences, and special cases. Artifacts occur in algorithms that impose extra structure on the input, such as the z order in our sweep algorithm. They can be avoided by randomization. Coincidences occur when combinatorially distinct elements are numerically equal, for example $a \cdot (b \times c)$ with $a = b = c$. We replace $u = b \times c$ by $\hat{u}$. A generalization of this strategy handles any rank deficient determinant predicate. Special cases occur when the parameters of a predicate are related. We exploit the parameter relationship to derive an equivalent regular predicate, such as the clockwise edge order at a z turn (Sec. 3). We have employed this strategy in several complicated 3D algorithms and aim to automate it.

Acknowledgment

Milenkovic supported by NSF grant CCF-0904707. Sacks supported by NSF grant CCF-0904832.

References

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

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

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

[4] E. Fogel and D. Halperin. Polyhedral assembly partitioning with infinite translations or the importance of being exact. In *Eighth International Workshop on the Algorithmic Foundations of Robotics*, pages 417–432, 2009.

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

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

[7] D. Halperin, J.-C. Latombe, and R. H. Wilson. A general framework for assembly planning: The motion space approach. *Algorithmica*, 26:577–601, 2000.

[8] V. Milenkovic, E. Sacks, and M.-H. Kyung. Robust minkowski sums of polyhedra via controlled linear perturbation. In *Solid Modeling*, pages 23–30. Springer, 2010.

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