Computer-Aided Design 108 (2019) 12–18

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Geometric rounding and feature separation in meshes☆

Victor Milenkovic [a], Elisha Sacks [b,*]

[a] Department of Computer Science, University of Miami, Coral Gables, FL 33124-4245, USA
[b] Computer Science Department, Purdue University, West Lafayette, IN 47907-2066, USA

ARTICLE INFO

Article history:
Received 22 May 2018
Accepted 12 October 2018

Keywords:
Geometric rounding
Mesh simplification
Robust computational geometry

ABSTRACT

Geometric rounding of a 3D triangle mesh is the task of approximating the vertex coordinates by floating point numbers while preserving the topology. We present a practical geometric rounding algorithm based on a novel strategy: (1) modify the mesh to achieve a feature separation that prevents topology changes when the coordinates change by the rounding unit; and (2) round each vertex coordinate to the closest floating point number. The geometric rounding algorithm allows computational geometry algorithms to interface with numerical algorithms. Mesh feature separation is also useful on its own, for example for enforcing minimum feature sizes in CAD models. We demonstrate a robust, efficient implementation.

1. Introduction

A common representation for a surface is a triangle mesh: a set of triangles with shared vertices and edges. Meshes are usually constructed from basic elements (polyhedra, triangulated surfaces) through sequences of operations (linear transformations, Booleans, offsets, sweeps). Although typical basic elements have floating point vertex coordinates, mesh vertices can have much higher precision. For example, the intersection point of three triangles has thirteen times the precision of their vertices. High precision coordinates are incompatible with numerical codes that use floating point arithmetic, such as finite element solvers. Rewriting the software to use extended precision arithmetic would be a huge effort and would entail an unacceptable performance penalty. Instead, the coordinates must be approximated by floating point numbers.

One strategy is to construct meshes using floating point arithmetic, so the coordinates are rounded as they are computed. This strategy suffers from the robustness problem: software failure or invalid output due to numerical error. The problem arises because even a tiny numerical error can make a geometric predicate have an incorrect value, which in turn can invalidate the entire algorithm. For a controlled class of inputs, robustness can be ensured by careful engineering of the software, but the problem tends to recur when new domains are explored.

A solution to the robustness problem, called Exact Computational Geometry (ECG) [1], is to represent geometry exactly and to evaluate predicates exactly. ECG is efficient because most predicates can be evaluated exactly using floating point arithmetic. Mesh construction using ECG is common in computational geometry research and plays a growing role in applications. We use ECG to implement Booleans, linear transformations, offsets, and Minkowski sums [2,3]; the CGAL library [4] provides many ECG implementations.

Although ECG guarantees a correct mesh with exact vertex coordinates, it does not provide a way to approximate the coordinates in floating point. Rounding the coordinates to the nearest floating point numbers can cause triangles to intersect. A *geometric rounding* is an intersection free approximation. Prior work provides efficient geometric rounding algorithms for 2D geometry [5–7], but not for 3D triangle meshes (Section 2).

Contribution. We present a practical geometric rounding algorithm. The input and output are sets of 3D triangles whose interiors and edge interiors do not intersect. The input coordinates are exact and the output coordinates are double-floats. There exists a continuous, non-intersecting hence topology preserving, deformation from the input to the output. The algorithm increases the separation of the close features while preserving the topology, then rounds the coordinates. The separation ensures that the rounding also preserves the topology.

The features of a mesh are its vertices, edges, and triangles. Since the interiors of edges and triangles do not intersect, features are disjoint if and only if they share no vertices. The separation of a mesh is the minimum distance between disjoint features. If the separation exceeds d, moving each vertex by a distance of at most $d/2$ cannot make any features intersect, which implies that the mesh topology is preserved. If the vertex coordinate magnitudes are bounded by M, rounding them to the nearest floating point numbers moves a vertex at most $e = \sqrt{3}M\epsilon$ with ϵ the rounding

☆ This paper has been recommended for acceptance by Tao Ju.
* Corresponding author.
E-mail addresses: vjm@cs.miami.edu (V. Milenkovic), eps@purdue.edu (E. Sacks).

https://doi.org/10.1016/j.cad.2018.10.003
0010-4485/

unit. Thus, the topology is preserved by increasing the separation to $2e$ then rounding.

Our mesh separation algorithm increases the separation of a mesh in three stages. Modification removes short edges and skinny triangles by standard mesh edits (Section 3), thereby eliminating many close pairs of features. Expansion and optimization separate the remaining close features via locally minimum vertex displacements. Expansion iteratively increases the separation of the close features until they are separated (Section 4). Optimization takes the separated mesh as input and minimizes the displacement of the vertices from their original positions under the separation constraints (Section 5).

Beyond geometric rounding, mesh separation is a novel form of mesh improvement that is useful in computations that suffer from close features. One application is removing ill-conditioned triangles from finite element meshes. Another application is computer-aided design, which typically requires a minimum feature size due to manufacturing constraints. Current software removes small features heuristically, e.g. by merging close vertices, which can create triangle intersections. Our algorithm provides a safe alternative.

Implementation. We implement the geometric rounding algorithm robustly (Section 6). We use ECG for the computational geometry tasks, such as finding close features and testing if triangles intersect. We use linear programming with floating point arithmetic and adaptive scaling in expansion and optimization. We test the implementation on custom and random meshes (Section 7). We conclude with a discussion (Section 8).

2. Prior work

Fortune [8] rounds a mesh of size n in a manner that increases the mesh size to n^4 and the vertex complexity by $\log n$ bits. Devillers, Lazard, and Lenhart [9] round to a mesh of size n^{15} in time n^{19} without increasing the vertex complexity. Fortune [10] gives a practical rounding algorithm for plane-based polyhedra. We [2] extend this algorithm to polyhedra in a mesh representation. Most output vertices have floating point coordinates. The highest precision output vertex is an intersection point of three triangles with floating point coordinates.

Our new algorithm improves on our prior algorithm. The input can be any set of non-intersecting triangles. All the output vertices have floating point coordinates. The mesh topology is preserved. Close features are removed. Unlike all prior mesh rounding algorithms, the output size is bounded by the input size. However, our error bound is empirical.

Zhou et al. [11] present a heuristic form of geometric rounding akin to our prior algorithm and show that it works on 99.95% of 10,000 test meshes. Our new algorithm is several times faster than this heuristic on our test cases.

There is extensive research [12] on mesh simplification: accurately approximating a mesh of small triangles by a smaller mesh of larger triangles. The mesh edits in the modification stage of our algorithm come from this work. Simplification reduces the number of close features as a side effect of reducing the number of triangles and increasing their size, but removing all the close features is not attempted. Mesh untangling and improvement algorithms [13,14] improve a mesh by computing vertex displacements via optimization. Although we use a similar strategy, our objective functions and optimization algorithms are novel. Cheng, Dey, and Shewchuk [15] improve Delaunay meshes with algorithms that remove some close features, notably sliver tetrahedra.

3. Modification

The modification stage of the mesh separation algorithm consists of a series of mesh edits: edge contractions remove short edges and edge flips remove skinny triangles. We perform every edit that satisfies its preconditions and that does not cause triangles to intersect.

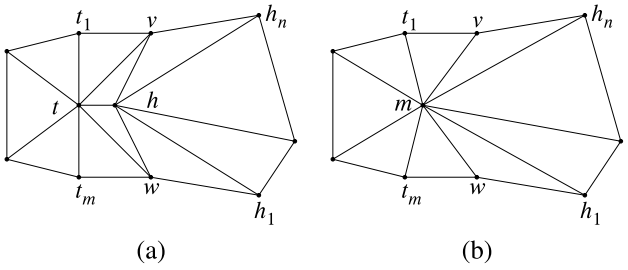

Fig. 1. (a) Short edge th; (b) th contraction.

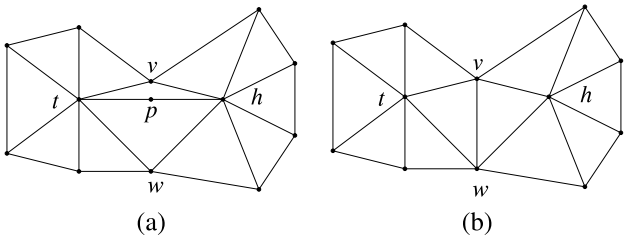

Fig. 2. (a) Skinny triangle thv; (b) th flip.

An edge th is short if $\|h-t\| < d$. For th to be contracted, it must be manifold, that is incident on two triangles thv and htw (Fig. 1a). The triangles incident on t form a surface whose boundary is a simple loop $h, v, t_1, \ldots, t_m, w$; likewise for h with $t, w, h_1, \ldots, h_n, v$. If the t_i and the h_j are disjoint sets, th is contracted: it is replaced with a new vertex $m = (t + h)/2$ and the incident edges and triangles are updated (Fig. 1b).

A triangle thv is skinny if v projects onto a point p in th and $\|p - v\| < d$ (Fig. 2a). The edge th is flipped if it is incident on triangles thv and htw, vw is not an edge of the mesh, and the triangles vwh and wvt are not skinny. The flip replaces th with vw, and replaces thv and htw with vwh and wvt (Fig. 2b).

Analysis

Modification terminates because every contraction reduces the number of edges and every flip maintains this number and reduces the number of skinny triangles. The computational complexity of an edit is constant, except for intersection testing, which is linear in the number of mesh triangles. We cannot bound the number of edits in terms of the number of short edges and skinny triangles in the input because edge contractions can create skinny triangles.

Edits preserve the intrinsic topology of the mesh because they replace a topological disk by another topological disk with the same boundary. The boundary is $t_1, \ldots, t_m, w, h_1, \ldots, h_n, v$ for a short edge and is $vtwh$ for a skinny triangle. If the pair of disks surrounds a subset of the mesh, such as a small separate component, the edit is rejected because it cannot be accomplished by a continuous deformation. This property and the rejection of edits that cause intersections jointly preserve the embedded topology of the mesh.

We measure error by the distance between the input and the output meshes. An edit deforms its topological disk by at most d, so the error due to modification is bounded by d times the number of edits. Although we lack a bound in terms of the input, the median error is only $0.5d$ in our tests (Section 7).

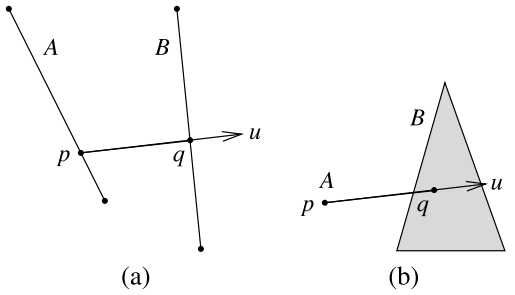

Fig. 3. Features A and B with closest points p and q: (a) two edges, (b) vertex and triangle.

4. Expansion

The expansion algorithm (below) repeatedly increases the separation until it exceeds d. Each step assigns every vertex a displacement that is added to its position to obtain its new position. The displacements are computed by linear programs that approximate the distances between the features by linear functions of the displacements and that bound the displacements by $\Delta = d$ to control the truncation error. If triangles intersect or the separation of the mesh increases, the truncation error is reduced by halving Δ and the displacements are recomputed. Otherwise, the vertices are updated and the step ends.

Expansion Algorithm

While the separation of the mesh is less than d:

1. Formulate linear separation constraints.
2. Set $\Delta = d$.
3. Bound the vertex coordinate displacements by Δ.
4. Compute vertex displacements subject to 1 and 3.
5. If triangles intersect or the separation increases, set $\Delta = \Delta/2$ and go to step 3.
6. . Update the vertices.

The separation of a mesh equals the minimum distance between a vertex and a triangle or between two edges. The distance between features A and B is the maximum over unit vectors u of the minimum of $u \cdot (b - a)$ over the vertices $a \in A$ and $b \in B$. The optimal u is parallel to $q - p$ with p and q the closest points of A and B, which need not be vertices (Fig. 3).

We assign each vertex a a displacement a' and approximate the distance between the displaced features by a linear function of the displacements. The first order displacement of u is expressible as $lv + mw$ with u, v, and w orthonormal vectors. We obtain the first order distance

$$\min_{a\in A, b\in B} u \cdot (b - a) + u \cdot (b' - a') + (lv + mw) \cdot (b - a)$$

by substituting the displaced values into $u \cdot (b - a)$ and dropping the quadratic terms.

We compute approximately optimal displacements subject to constraints on the first order distances between features. It would be wasteful to constrain every pair of features because most features are too far apart to be affected by displacements of size d. On the other hand, we must constrain the pairs that the optimizer might otherwise cause to intersect. A vertex can be displaced by a distance of at most $\sqrt{3}d$ because we bound the coordinate displacements by $\Delta \leq d$. Hence, we need only constrain the pairs of triangles that are at most $2\sqrt{3}d$ apart.

We compute the displacements by solving two linear programs (LPs). A vertex displacement a' is represented by variables a'_x, a'_y, a'_z, abbreviated as $a'_{x,y,z}$. We model the quantities l and m in the first order distance estimates as LP variables, rather than expressing them in terms of the vertex positions. We omit the lengthy proof that the LPs compute their correct values.

First Expansion LP

Constants: d, Δ, the vertices, and the vectors u, v, and w for each pair of features.

Variables: s, $a'_{x,y,z}$ for each vertex a, l and m for each pair of features.

Objective: maximize s.

Constraints: (1) $0 \leq s \leq 1$, (2) $-\Delta \leq a'_{x,y,z} \leq \Delta$, (3) $u \cdot (b - a) + u \cdot (b' - a') + (lv + mw) \cdot (b - a) \geq sd$.

Second Expansion LP

Constants: d, Δ, the vertices, the vectors u, v, and w for each pair of features, s_m.

Variables: $a'_{x,y,z}$ and $a^m_{x,y,z}$ for each vertex a, l and m for each pair of features.

Objective: minimize $\sum_a a^m_x + a^m_y + a^m_z$.

Constraints: (1) $-\Delta \leq a'_{x,y,z} \leq \Delta$, (2) $-a^{\mathrm{m}}_{x,y,z} \leq a'_{x,y,z} \leq a^{\mathrm{m}}_{x,y,z}$, (3) $u \cdot (b - a) + u \cdot (b' - a') + (lv + mw) \cdot (b - a) \geq s_m d$.

The first LP maximizes the mesh separation subject to the Δ bounds. It constrains the distances between the close features to exceed sd with $0 \leq s \leq 1$ a variable that it maximizes to $s = s_{\mathrm{m}}$. Since $s = 1$ implies a separation of d, a larger value would increase the error for no reason. The second LP computes minimal displacements that achieve $s = s_{\mathrm{m}}$. The magnitude of the displacement of a vertex a is represented by variables $a^{\mathrm{m}}_{x,y,z}$ with constraints $-a^{\mathrm{m}}_{x,y,z} \leq a'_{x,y,z} \leq a^{\mathrm{m}}_{x,y,z}$. The objective is to minimize the sum over the vertices of these variables. The variable s is replaced by the constant s_{m} in the constraints.

Analysis

Expansion preserves the mesh topology because each step checks that the linear motion from the initial vertex positions to their updated positions does not cause a triangle intersection.

We prove that expansion terminates. The first LP computes a positive s_m because no triangles intersect. The displacements that it computes are feasible for the second LP. Hence, every expansion step succeeds for some Δ. It remains to bound the number of steps.

We employ the following definitions. A Δ-displacement is a displacement in which the coordinate displacements are bounded by Δ. Let δ denote the minimum separation of the mesh. The *tightness* τ as the minimum over all Δ-displacements of the ratio of Δ to the increase in δ.

Theorem 1. *The number of expansion steps is bounded.*

Proof. A Δ-displacement can increase the separation of a pair by at most $2\sqrt{3}\Delta$, so $\tau > \sqrt{3}/6$. Let M be the maximum vertex coordinate magnitude. The Δ-displacement $a' = \Delta a/M$ increases δ by $\Delta\delta/M$, so $\tau < \Delta/(\Delta\delta/M) = M/\delta$. Since scaling up the mesh is a feasible solution yet does not increase M/δ, neither does the optimal solution. Hence, $\tau < M/\delta$ at every expansion step.

The LP maximizes $S = sd$ for the linear separation constraints. Let S^* be the maximum of S for the true constraints and let T be the maximum truncation error of the linear constraints. The linear separations for the optimal displacement are at least $S^* - T$. Therefore, the LP solution satisfies $S \geq S^* - T$ and the true separation is at least $S - T \geq S^* - 2T$.

For a Δ-displacement with $\Delta = \delta/k$, $T < c\delta/k^2$ for some constant c. For $k = 4c\tau$, the optimal Δ-step increases δ by $\delta/k\tau = \delta/4c\tau^2$, so the LP increases δ by $\delta/4c\tau^2 - 2c\delta/k^2 = \delta/8c\tau^2$. Since one step multiplies δ by $1 + 1/8c\tau^2$, $O(\tau^2 \log(d/\delta))$ steps increase it to d.

5. Optimization

The optimization algorithm takes the output of expansion as input and locally minimizes the displacement of the vertices from their original positions. We apply a gradient descent strategy to the semi-algebraic set of vertex values for which the mesh is d-separated. We compute the direction in which the displacement decreases most rapidly subject to the linear separation constraints. We take a step of size $\beta = d$ in this direction and use the expansion algorithm to correct for the linearization error and return to the d-separated space. If the displacement increases, we divide β by 2 and retry the step. If four consecutive steps succeed, we double β. The stopping criterion is that the displacement decreases by less than $0.01d$.

We compute the descent direction by solving an LP. We represent the pre-expansion value of a vertex a by constants $a^0_{x,y,z}$ and represent its displacement by variables $a^{\mathrm{d}}_{x,y,z}$ with constraints $-a^{\mathrm{d}}_{x,y,z} \le a_{x,y,z} + a'_{x,y,z} - a^0_{x,y,z} \le a^{\mathrm{d}}_{x,y,z}$ and $-\beta \le a'_{x,y,z} \le \beta$. The objective is to minimize the sum over the vertices of $a^{\mathrm{d}}_{x,y,z}$. We drop s from the constraints because they are feasible with $s = 1$ due to expansion.

Optimization LP

Constants: β, d, the vertices, the pre-expansion vertices a^0, the vectors u, v, w.

Variables: the $a'_{x,y,z}$, l, m, and $a^d_{x,y,z}$.

Objective: minimize $\sum_a a^d_x + a^d_y + a^d_z$.

Constraints: (1) $-\beta \le a'_{x,y,z} \le \beta$,

(2) $-a^{\mathrm{d}}_{x,y,z} \le a_{x,y,z} + a'_{x,y,z} - a^0_{x,y,z} \le a^{\mathrm{d}}_{x,y,z}$,

(3) $u \cdot (b - a) + u \cdot (b' - a') + (lv + mw) \cdot (b - a) \ge d$.

Analysis

Our algorithm differs from standard gradient descent in that an expansion step can undo the decrease in displacement from the prior descent step. When this occurs, the algorithm halves β. The error of the expansion is quadratic in β. The correcting displacement is proportional to this error, hence is quadratic in the descent displacement. For a small enough β, the correction no longer undoes the progress of the descent.

6. Implementation

Computation. The input vertex coordinates are real numbers represented exactly using our ECG library [3]. The mesh separation algorithm is implemented exactly, except for linear programming. The inputs to the LPs are floats whose computation is described below. A vertex coordinate is updated by exact addition of its current value and its float displacement. The output vertex coordinates are rounded to double-float.

Modification. We accelerate the intersection tests by storing the mesh in an octree that is updated after each edit. We perform mesh edits in an order that reduces the number of tests. Contractions come before flips because removing a short edge also eliminates two skinny triangles. We perform edits of the same type in order of feature distance. When we contract an edge of length x, we need only test for intersections between the new triangles and old triangles within distance $x/2$. When we remove a skinny triangle of size x, we need only test for intersections between the two new triangles and old triangles within distance x. In both cases, prior edits have eliminated most such old triangles.

Expansion. We combine the two LPs into one LP that achieves the same results in half the time. We take the constants, variables, and constraints from the first LP and add the a^m and their constraints. We maximize $s - kn \sum_a a^m_x + a^m_y + a^m_z$ with k a constant and with n the number of vertices. Any sufficiently large k forces the constraints that determine $s = s_m$ into the basis, thereby making the new LP equivalent to the two LPs. We set $k = 6000$ and the combined LP works well on our tests, although convergence is not guaranteed.

Intersection test. Expansion and optimization verify that each vertex update does not cause any pair of features A and B to intersect. We express the displacement of each vertex v as a linear transform $v + v't$, so A and B become functions of t with initial and displaced values at $t = 0$ and $t = 1$. The pair intersects if A and B are in contact at some $t \in [0, 1]$. We first try a fast, conservative test: there is no intersection if $(u + lv + mw) \cdot (b + b' - a - a') > 0$ for all vertices $a \in A$ and $b \in B$. Otherwise, we compute the t values where the features are coplanar by solving a cubic, transform the vertices by each value, and check if the transformed features intersect.

Linear programs. We use the IBM CPLEX solver. The input size is not a concern because the number of constraints is proportional to the number of close features. The challenge is to formulate a well-conditioned LP. We scale the displacement variables by d because their values are small multiples of d. We divide the separation constraints by d to avoid tiny coefficients. We bound the magnitudes of l and m by 1 and scale them by a parameter α that is initialized to 1. After solving an LP, we check if the solution violates any of the constraints by over 10^{-6}, meaning that the separation is off by over $10^{-6}d$. If so, we multiply α by 10 and re-solve the LP. This strategy trades off accuracy for number of iterations when scaling is poor.

We do not formulate the LP in terms of the rounded vertex coordinates because rounding errors could make it infeasible. Instead, we formulate in terms of the quantities u, $u \cdot (b - a)$, $v \cdot (b - a)$, and $w \cdot (b - a)$, which we calculate to floating point accuracy. The floating point value of $u \cdot (b - a)$ is nonnegative because the exact value is positive, so the LP is satisfied when all the variables are set to zero.

The only accuracy problem that remains after scaling is that $v \cdot (b - a)$ or $w \cdot (b - a)$ can have much larger magnitude than $u \cdot (b - a)$. If that causes an inaccurate output, we detect the resulting constraint violation and change the value of α, thereby dividing the two largest coefficients by 10.

7. Testing

We tested our geometric rounding algorithm on five types of custom meshes and on random meshes. The former validate the algorithm on common applications and the latter strengthen the statistical analysis of its performance. Table 1 shows the results for several meshes of each type. The running times are for an Intel Core i7-6700 CPU at 3.40 GHz with 16 GB of RAM. A statistical analysis of the test results appears below.

(1) Isosurfaces. The meshes are isosurfaces constructed by a marching cubes algorithm, and simplified isosurfaces (Fig. 4). The vertices have double-float coordinates.

(2) Tetrahedral meshes. The meshes are Delaunay tetrahedralizations of polyhedra with double-float coordinates (Fig. 5). Modification is omitted because edge contraction destroys the tetrahedral mesh structure.

(3) Minkowski sums. The meshes are Minkowski sums of polyhedra with double-float coordinates (Fig. 6). We construct the Minkowski sums robustly using our prior algorithm [3], but without perturbing the input. The mesh vertex coordinates are ratios of degree 7 and degree 6 polynomials in the input coordinates, hence have about 800 bit precision.

Table 1
Custom mesh separation: mesh type t, separation $d = 10^{-e}$, f triangles, c_m d-close features, v_m percentage vertices displaced, a_m median and m_m max displacement in units of d, t_m seconds running time for modification; likewise c_e, v_e, a_e, m_e, t_e for expansion and v_o, a_o, m_o, t_o for optimization.

t	e	f	c_m	v_m	a_m	m_m	t_m	c_e	v_e	a_e	m_e	t_e	v_o	a_o	m_o	t_o
1	6	34876	48	0.00	0.00	0.00	0.02	12	0.02	0.00	0.00	0.11	0.02	0.00	0.00	0.14
		35898	48	0.00	0.00	0.00	0.03	12	0.02	0.00	0.00	0.11	0.02	0.00	0.00	0.14
		123534	3	0.00	0.00	0.00	0.10	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		125394	7	0.00	0.00	0.00	0.10	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		1596	185	0.37	0.56	0.65	0.01	14	0.37	0.21	0.23	0.14	0.37	0.21	0.23	0.20
		2568	241	0.07	0.13	0.13	0.01	87	1.94	0.98	3.00	0.38	1.94	0.56	1.25	1.17
2	6	125986	4	0.00	0.00	0.00	0.0	4	0.07	0.83	1.00	2.28	0.08	0.71	2.83	4.10
		154084	0	0.00	0.00	0.00	0.0	0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	6	377534	5567	0.13	0.01	0.70	0.88	94	0.01	0.63	2.14	1.37	0.01	0.47	2.12	15
		407014	20675	0.54	0.02	0.83	1.34	198	0.02	0.38	1.71	3.32	0.03	0.32	1.45	7.87
		436308	28848	0.23	0.33	1.02	2.66	941	0.06	0.57	3.97	57.44	0.07	0.48	3.43	297
		438364	393	0.01	0.40	0.83	0.57	14	0.00	0.36	1.00	1.43	0.00	0.36	0.78	2.59
		641604	313	0.00	0.53	0.70	1.32	20	0.00	0.48	0.89	1.46	0.00	0.46	0.86	2.56
		667838	436	0.00	0.54	0.94	1.31	20	0.00	0.51	1.03	1.29	0.00	0.47	0.83	2.40
		773236	337	0.00	0.28	0.48	0.95	10	0.00	0.43	1.00	1.89	0.00	0.36	0.98	3.27
4	6	1618	2871	6.39	0.12	0.74	0.04	108	3.44	0.91	2.04	0.46	3.44	0.61	2.00	2.77
		1626	3868	7.58	0.06	0.49	0.05	104	3.42	0.69	1.99	0.21	3.54	0.50	1.03	2.21
5	15	465524	22777	0.30	0.00	0.00	2.87	758	0.16	0.00	0.00	9.27	0.00	0.00	0.00	0.00
		473022	59549	0.79	0.00	0.00	5.59	2024	0.62	0.00	0.00	113.11	0.00	0.00	0.00	0.00
6	5	170936	7875	0.02	0.36	0.77	0.47	6036	0.58	0.63	3.94	17.18	0.61	0.52	3.03	120
		725480	48862	0.03	4.09	7.70	2.39	41382	0.86	5.61	44.73	231.73	0.00	0.00	0.00	0.00

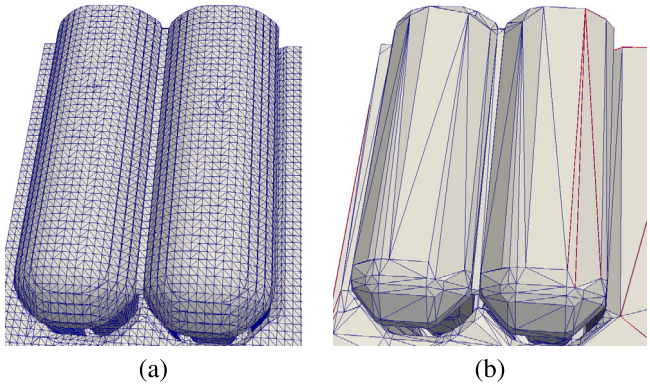

Fig. 4. (a) Isosurface; (b) simplified isosurface with close features in red. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

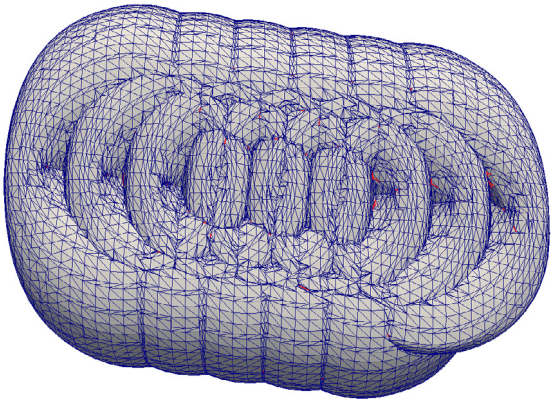

Fig. 6. Minkowski sum with close features in red. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

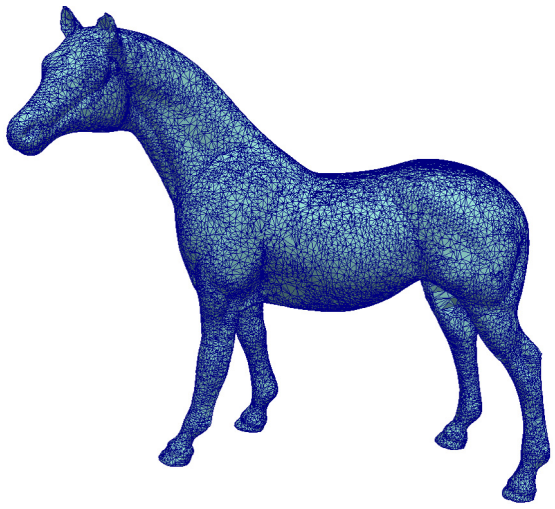

Fig. 5. Polyhedron whose tetrahedral mesh has 150,000 triangles.

(4) Approximate 4D free spaces. The meshes jointly approximate the 4D free space of a polyhedral robot that rotates around the z axis and translates freely relative to a polyhedral obstacle [16]. Each mesh is the complement of the Minkowski sum of the obstacle with a polyhedral approximation of the volume swept by the negative of the robot as it rotates over 1/40 of the circle. We employ a rational parameterization of the rotation matrix with parameter t. The coordinates of a rotated vertex are ratios of polynomials that are quadratic in t and linear in the input coordinates, for total degree 6. The highest precision vertices of the swept volume are intersection points of three triangles composed of rotated vertices. The degree of their coordinates is $13 \times 6 = 78$ and the input has 27 bit precision, so the mesh precision is about 2100 bits.

(5) Triangulated 3D free spaces. The meshes are triangulated 3D free spaces for a polyhedral robot that rotates around the z axis and translates freely in the xy plane relative to a stationary polyhedron, which we compute with our prior algorithm [17]. The free space coordinates are (x, y, t) with t a rational parameterization of the rotation angle. The t coordinates of the vertices are zeros of quartic polynomials whose coefficients are polynomials in the coordinates of the input vertices. Their algebraic degree and precision are very

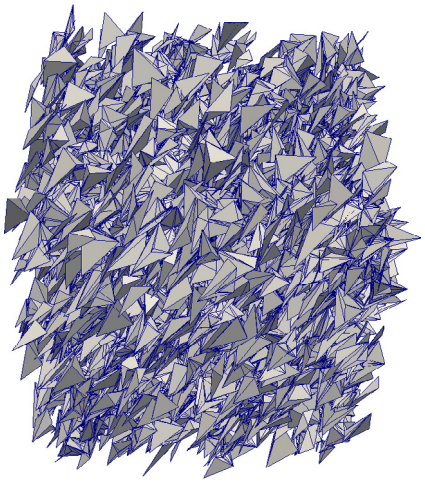

Fig. 7. Typical random tetrahedron.

high. The implementation uses a heuristic to detect degenerate (zero) predicates involving roots of polynomials because exact computation is too slow. It assumes that a predicate is zero if its sign is ambiguous in 848-bit interval arithmetic. The output of the geometric rounding algorithm is verified in exact arithmetic.

(6) Random meshes. Each mesh is the overlay of random tetrahedra in the unit cube. Each tetrahedron has vertices t, $t + a$, $t + b$, and $t + c$ with the coordinates of t uniform in $[0, 0.9]$ and the coordinates of a, b, and c uniform in $[0, 0.1]$ (Fig. 7). The random coordinates are double-floats and the overlay vertices where three triangles intersect have about 700 bit precision.

Results. We tested 126 custom meshes (types 1 to 5). For types 1 to 4, we use $d = 10^{-6}$, which is a typical minimum feature size in CAD software. For type 5, we use $d = 10^{-15}$, which is the smallest separation that supports geometric rounding.

The meshes have median 9200 and maximum 770,000 triangles, and median 400 and maximum 88,300 close feature pairs. Modification reduces the number of close pairs to median 18 and maximum 2000, and displaces median 1.8% and maximum 8.6% of the vertices with median and maximum displacements of $0.08d$ and $10.7d$. Expansion displaces median 2% and maximum 30% of the vertices with median and maximum displacements of $0.4d$ and $6d$. Of the 110 meshes that we optimize (types 1 to 4), expansion displaces the vertices by median $0.5d$ and maximum $4d$, which optimization changes to $0.45d$ and $6d$.

We tested 100 type 6 meshes with 4000 random tetrahedra. We used $d = 10^{-5}$ to increase the number of close features hence the difficulty. The meshes have median 168,000 and maximum 178,000 triangles. They have 13,000–14,000 connected components, whereas the custom meshes have at most 5 components. There are median 8800 and maximum 13,400 close feature pairs. Modification reduces the number of close pairs to median 7400 and maximum 12,000, and displaces median 0.02% and maximum 0.04% of the vertices with median and maximum displacements of $0.4d$ and $0.8d$. Expansion displaces median 0.65% and maximum 0.85% of the vertices with median and maximum displacements of $0.6d$ and $24d$, which optimization changes to $0.5d$ and $29d$.

We tested 10 meshes with 8,000 random tetrahedra, median 720,000 and maximum 740,000 triangles, and median 49,000 and maximum 57,000 close feature pairs. Modification reduces the close pairs to median 41,000 and maximum 49,000, and displaces median 0.03% and maximum 0.04% of the vertices with median and maximum displacements of $0.4d$ and $0.8d$. Expansion displaces median 0.9% and maximum 1.0% of the vertices with median and maximum displacements of $0.6d$ and $21d$. Comparing these results with those on the smaller random meshes shows that increasing the input size by a factor 4.3 has little impact on the performance.

The running time of mesh separation on the random meshes is approximately linear in the number of close features c with median and maximum ratios of 0.02 and 0.06. The running time is similar on most of the custom meshes, but there are a few outliers where it is quadratic in c. The running time of geometric rounding equals that of mesh separation plus the time to round the vertex coordinates to double-float. The latter is independent of the geometric rounding algorithm. It is under 10% of the total running time in our tests.

Standard datasets. We tested the mesh separation algorithm on a few meshes from the ShapeNet [18] and Thingi10K [19] datasets. There is no need for geometric rounding because the coordinates are floating point numbers. The test meshes have few close features, so the algorithm is fast and accurate. We did not test the datasets exhaustively because many of the meshes self-intersect, which is hard to detect and correct. We advocate the alternate strategy of constructing correct meshes using exact computational geometry then performing geometric rounding.

8. Discussion

The mesh separation algorithm performs well on the tests. The median vertex displacement is under $0.5d$. The ratio of displaced vertices to close feature pairs is nearly constant: median 0.03 and maximum 3.0 for the custom meshes and median 0.04 and maximum 0.05 for the random meshes. The median number of expansion iterations is 1 for the custom meshes, 4 for the small random meshes, 5 for the large random meshes, and maximum 11. The ratio of running time to close feature pairs is close to linear. We observe no difference in performance as d varies from 10^{-5} to 10^{-15}.

This performance corresponds to the assumption that every pair of close features is far from every other pair. The complexity of modification is linear in the number of short edges and skinny triangles because mesh edits do not create close features. The error is bounded by d because a vertex is displaced at most once. The expansion LP achieves the optimal displacement with $\Delta = d$ because the truncation error is negligible. Thus, the number of steps is constant. Optimization converges in a few steps because the initial displacement is close to d.

A more realistic assumption is that every pair of close features is close to a bounded number of close feature pairs. An analysis under this assumption is a topic for future work.

We tested the benefit of modification by performing mesh separation via expansion alone. The median vertex displacement increases slightly, the mesh size increases proportionally to the number of short edges, and the running time increases sharply. To understand the small change in displacement, consider two vertices that are e apart. Modification displaces them by $e/2$, versus $(d - e)/2$ for expansion. Assuming e uniform on $[0, d]$, the average displacement is the same. The running time increases because mesh edits take constant time, whereas expansion is quadratic in the number of close feature pairs.

We could contract non-manifold edges subject to a more complicated precondition. We have found no application where this is worthwhile. Closed surfaces have no non-manifold edges and open surfaces have few. Tetrahedral meshes have mostly non-manifold edges, but they cannot be contracted because doing so would destroy the tetrahedral structure.

Expansion has the potential to increase the volume of the mesh unboundedly, which is why we cannot prove a useful bound on the number of iterations or the mesh displacement. We estimate

the expansion in our tests as the ratio of the volumes of the mesh bounding boxes before and after expansion. The largest increase is 0.001%, including the tests without modification where expansion moves many more vertices. This result helps explain why the number of iterations and the mesh displacement are small in practice.

The standard linear distance function, in which l and m are functions of the vertex coordinates, is much slower than our version, in which l and m are LP variables bounded by 1. If the bound is reduced to 0.001, the running time drops and the error grows. If it is set to zero, the running time drops further, but in rare cases expansion does not converge.

Geometric rounding can be sped up by skipping optimization when modification plus expansion achieve mesh separation with a small displacement. This is the case for the custom and random meshes. The median error after expansion is well under d, and optimization reduces it by 10% and 16%, and takes 70% and 88% of the running time.

The results on tetrahedral meshes suggest that mesh separation can play a role in mesh improvement. It simultaneously improves the close features by displacing multiple vertices in a locally optimal manner, whereas prior work optimizes one vertex at a time. The tests show that the simultaneous approach is fast and effective. It readily extends to control other aspects of the mesh, such as triangle normals. Applying the approach to other mesh improvement tasks is a topic for future work.

Acknowledgments

Sacks is supported by National Science Foundation, USA grant CCF-1524455 and by an Intel, USA grant. Milenkovic is supported by National Science Foundation, USA grant CCF-1526335.

References

[1] Yap C. Robust geometric computation. In: Goodman JE, O'Rourke J, editors. Handbook of discrete and computational geometry. 2nd ed. Boca Raton, FL: CRC Press; 2004, p. 927–52 [chapter 41].
[2] Sacks E, Milenkovic V. Robust cascading of operations on polyhedra. Comput Aided Des 2014;46:216–20.
[3] Kyung M-H, Sacks E, Milenkovic V. Robust polyhedral Minkowski sums with GPU implementation. Comput Aided Des 2015;67–68:48–57.
[4] CGAL, Computational Geometry Algorithms Library, http://www.cgal.org.
[5] Halperin D, Packer E. Iterated snap rounding. Comput Geom 2002; 23(2):209–22.
[6] Milenkovic V. Shortest path geometric rounding. Algorithmica 2000; 27(1):57–86.
[7] Goodrich MT, Guibas LJ, Hershberger J, Tanenbaum PJ. Snap rounding line segments efficiently in two and three dimensions. In: Symposium on Computational Geometry; 1997. p. 284–93.
[8] Fortune S. Vertex-rounding a three-dimensional polyhedral subdivision. Discrete Comput Geom 1999;22:593–618.
[9] Devillers O, Lazard S, Lenhart WJ. 3D snap rounding. In: Speckmann B, Tóth CD, (editors). 34th international symposium on computational geometry, 99 of Leibniz International Proceedings in Informatics, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. Dagstuhl, Germany; 2018. p. 30:1–30:14.
[10] Fortune S. Polyhedral modelling with multiprecision integer arithmetic. Comput Aided Des 1997;29(2):123–33.
[11] Zhou Q, Grinspun E, Zorin D, Jacobson A. Mesh arrangements for solid geometry. ACM Trans Graph 2016;35(4):39:1–39:15.
[12] Cignoni P, Montani C, Scopigno R. A comparison of mesh simplification algorithms. Comput Graph 1998;22(1):37–54.
[13] Freitag LA, Plassmann P. Local optimization-based simplicial mesh untangling and improvement. Internat J Numer Methods Engrg 2000;49:109–25.
[14] Knupp PM. Hexahedral and tetrahedral mesh untangling. Eng Comput 2001;17(3):261–8.
[15] Cheng S-W, Dey TK, Shewchuk J. Delaunay mesh generation. Chapman and Hall; 2012.
[16] Arluck C, Milenkovic V, Sacks E. Approximate free space construction and maximum clearance path planning for a four degree of freedom robot. In: Proceedings of the Canadian conference on computational geometry; 2018.
[17] Sacks E, Butt N, Milenkovic V. Robust free space construction for a polyhedron with planar motion. Comput Aided Des 2017;90C:18–26.
[18] Chang AX, Funkhouser TA, Guibas LJ, Hanrahan P, Huang Q, Li Z, et al. Shapenet: An information-rich 3d model repository, CoRR abs/1512.03012.
[19] Zhou Q, Jacobson A. Thingi10k: A dataset of 10, 000 3d-printing models, CoRR abs/1605.04797.

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