Department of Computer Science
George Mason University
Technical Report Series

4400 University Drive MS#4A5
Fairfax, VA 22030-4444 USA
http://cs.gmu.edu/
703-993-1530

Dynamic Minkowski Sum of Convex Shapes

Evan Behar
ebehar@gmu.edu

Jyh-Ming Lien
jmlien@cs.gmu.edu

Technical Report GMU-CS-TR-2010-12

Abstract

Computing the Minkowski sums of rotating objects has always been done naïvely by re-computing every Minkowski sum from scratch. The correspondences between the Minkowski sums are typically completely ignored. We propose a method, called DYMSUM, that can efficiently update the Minkowski sums of rotating convex polyhedra. We show that DYMSUM is significantly more efficient than the traditional approach, in particular when the size of the input polyhedra are large and when the rotation is small between frames. From our experimental results, we show that the computation time of the proposed method grows slowly with respect to the size of the input comparing to the naïve approach.

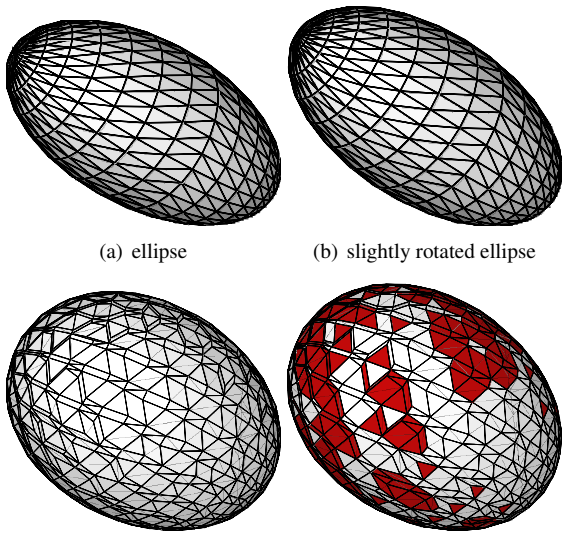
1 INTRODUCTION

The Minkowski sum is an important operation in robotics due to its fundamental role in providing the geometric reasoning ability to the robots, such as configuration space mapping, collision detection and penetration depth estimation. The Minkowski sum of two shapes P and Q is:

$$P \oplus Q = \{p + q \mid p \in P, q \in Q\}. \quad (1)$$

Although the problem of computing Minkowski sums has been studied since the early 70's, researchers have been paying more attention to this problem recently; see the surveys in [6, 20, 4]. In particular, in 3-dimensions, methods [11, 4, 7, 5] are known to compute the Minkowski sum of *convex* polyhedra efficiently.

In this work, we are interested in a method that can efficiently compute the Minkowski sum of *rotating convex polyhedra*. Computing the Minkowski sum of polyhedra undergoing rotations can be found in many problems, such as *general penetration depth* estimation [23] for physically-based simulation and *configuration-space obstacle* mapping [22] for robotic motion planning . Figure 1 shows an example of the Minkowski sums before and after rotating the cube.



(a) ellipse (b) slightly rotated ellipse (c) (a)⊕GS4 (d) (a)⊕GS4

Figure 1: The Minkowski sums of a rotating ellipse and a sphere (GS4, shown in Fig. 3). The ellipse in (b) is rotated by $\pi/40$ from (a). The dark (red) facets in (d) are the differences between (c) and (d).

The main challenge of computing the Minkowski sum of two rotating polyhedra comes from that fact that Minkowski sum can be dramatically different after the input polyhedra rotate. Therefore, existing methods simply re-compute a new Minkowski sum every time when P or Q rotates. For example, this approach is traditionally used to slice the $\mathscr{C}$-space obstacles ($\mathscr{C}$-obst) in motion

planning. When the rotation of the robot is considered, $\mathscr{C}$-obst are approximated by repetitively computing the Minkowski sums of the robot with different orientations. These Minkowski sums are usually separated by a *fixed rotational resolution*. A main problem of re-computing the Minkowski sum from scratch is that it requires the same amount of computation even when a small amount of rotation is applied to P or Q.

Our work is motivated by the observation described above. Thus, our objective is to compute the Minkowski sums of rotating convex polyhedra without re-computing the entire Minkowski sum repetitively. The main idea in our method is to generate the Minkowski sum from the existing Minkowski sum. More specifically, we generate the new Minkowski sum by correcting the "errors" introduced by rotation.

In theory, computing the Minkowski sum of two convex shapes P and Q will take $O(mn \log mn)$ time by overlaying the Gaussian maps of P and Q with complexities $O(m)$ and $O(n)$, respectively [4]. It is also known that the (space) complexity of the Minkowski sum of the same P and Q is $O(mn)$ [4]. Therefore, we expect an algorithm, similar to ours, that updates the Minkowski sum, instead of re-computing from scratch, will be $O(\log mn)$ faster than the traditional (brute force) approach.

Our Contribution. In this paper, we propose a method that provides the desired properties mentioned above. We call this method: DYMSUM (**dy**namic **M**inkowski **sum**). We show that DYMSUM is significantly more efficient than the naïve method of re-computing the Minkowski sum from scratch, in particular when the size of the input polyhedra are large and when the rotation is small between frames. From our experimental results, we show that the computation time of DYMSUM grows slowly (e.g., linearly if inputs are cubes) with respect to the size of the input comparing to the naïve approach (see Section 5). A preliminary version of this work can be found in a video abstract [15]. Although we focus on convex shapes, DYMSUM can be used as the basic operation for non-convex polyhedra using convex decomposition.

2 RELATED WORK

Many methods have been proposed to compute the Minkowski sums of polygons or polyhedra, however, to the best of our knowledge, Mayer et al. [17] and our preliminary work [15] are the only works focusing on the Minkowski problem involving rotating polyhedra.

Mayer et al. [17] observed that the combinatorial structure of the Minkowski sum given two such polyhedra only changes at certain critical rotation values, and construct an efficient search structure they call a criticality map. Unfortunately, as they address in the paper, even for convex polyhedra with moderate numbers of vertices, the criticality map can become very large and costly to construct. They present concepts of the criticality map for two and three axes of rotation, but the growth order on the size of these structures makes them expensive to build and prohibitive to store. Enhancements are presented for using an axis-angle representation to answer general rotation queries, as well as dynamic processing to reduce the size of the structure, which trades off query time to reduce build time and storage requirements for the criticality map.

In order to provide enough background in hope that the readers can appreciate the results of this work more, we will briefly review existing works on the Minkowski sum of static inputs; see more detailed surveys in [6, 20, 4].

Convolution. The convolution of two shapes P and Q, denoted as $P \otimes Q$, is a set of line segments in 2-d or facets in 3-d that is generated by 'combining' the segments or the facets of P and Q [8]. One can think of the convolution as the Minkowski sum that involves only the boundary, i.e., $P \times Q = \partial P \oplus \partial Q$. It is known that the convolution forms a superset of the Minkowski sum [6], i.e., $\partial(P \oplus Q) \subset P \otimes Q$. To obtain the Minkowski sum boundary, it is necessary to trim the line segments or the facets of the convolution.

For 2-d polygons, Guibas and Seidel [9] show an output sensitive method to compute convolution curves. Later, Ghosh [6] proposed an approach, which unifies 2-d and 3-d, convex and non-convex, and Minkowski addition and decomposition operations. The main idea in his method is the negative shape and slope diagram. The slope diagram is closely related to the *Gaussian map*, which has been recently used by Fogel and Halperin [4] to implement robust and efficient Minkowski sum computations of convex objects.

The main difficulty of the convolution-based methods is in removing the portions of the facets that are inside the Minkowski sum. Recently, Wein [21] showed a robust and exact method based on convolution for non-convex polygons. To obtain the Minkowski sum boundary from the convolution, his method computes the arrangement induced by the line segments of the convolution and keeps the cells with non-zero winding numbers. No practical implementation is known for polyhedra using convolution due to the difficulty of computing the 3-d arrangement and its substructures [19].

Divide-and-Conquer. In the divide-and-conquer framework, the input models are decomposed into convex components. Then, this framework computes the pairwise Minkowski sums of the components. Finally, all these pairwise Minkowski sums are united.

This approach is first proposed by Lozano-Pérez [16] to compute $\mathscr{C}$-obst for motion planning. Although the

main idea of this approach is simple, the divide step (i.e., convex decomposition) and the merge step (i.e., union) can be very tricky. For example, it is known that creating solid convex decomposition robustly is difficult [2]. In addition, Agarwal et al. [1] have shown that different decomposition strategies can greatly affect the efficiency of this approach. Hachenberger [10] presents a robust and exact implementation using the Nef polyhedra in CGAL. However, his results are still limited to simple models.

The union step is even more troublesome. Flato [3] computes the unions using the cells induced by the arrangement of the line segments. He uses a hybrid strategy that combines arrangement with incremental insertion to gain better efficiency. Hachenberger [10] also studies how the order of the union operation affects the efficiency. To avoid this explicit union step, Varadhan and Manocha [20] proposed an approach that generates meshes approximating the Minkowski sum boundary using marching cube techniques to extract the iso-surface from a signed distance field. They proposed an adaptive cell to improve the robustness and efficiency of their method.

Point-Based Representation. Alternatively, points have been used to represent the Minkowski sum boundary [18]. Representing the boundary using only points has many benefits. First of all, generating such points is easier than generating meshes and can be done in parallel and in multi-resolution fashion [13]. Moreover, point-based representations can be generalized to higher dimensional motion planning problems [14].

Voxelization. Li and McMains [12] propose a GPU-based voxelization technique which directly computes a volumetric solid with high accuracy instead of an exact boundary representation of the Minkowski sum. The main limitation is that this technique does not generate the inner holes of the Minkowski sum.

3 A Brute Force Method

Without loss of generality, we assume that P is movable while Q is stationary. We let P_s and P_t be two copies of P at two configurations s and t with distinct orientations. Our goal is to compute $M_t = P_t \oplus Q$ from $M_s = P_s \oplus Q$. Moreover, the computation time of the Minkowski sum should be sensitive to the orientation difference between P_s and P_t, i.e., the smaller the difference between P_s and P_t, the faster the computation of M_t.

Computing the Minkowski sum of two convex shapes is usually based on the idea of overlaying two Gaussian maps of the inputs. The Gaussian map $g(P)$ of a polyhedron P is a sub-division of $\mathbb{S}^1$. One can think $g(P)$ and P as a dual. That is, each face f of P with the outward normal n_f corresponds to a vertex $g(f) \in g(P)$ with coordinate n_f, and each vertex v of P corresponds to a face $g(v) \in g(P)$ bounded by the normals of the faces incident to v. When we overlay two Gaussian maps $g(P)$ and $g(Q)$, a vertex v in $g(P)$ must be associated with exactly one face in $g(Q)$ that encloses v and vice versa. Moreover, the edges in $g(P)$ and $g(Q)$ can also intersect.

The facets of a Minkowski sum are defined exactly by these two types of interactions between $g(P)$ and $g(Q)$: the facets generated from a facet of P and a vertex of Q or vice versa, called fv-facets; and the facets generated from a pair of edges from P and Q, respectively, called ee-facets. A facet f and a vertex v produce an fv-facet if and only if the normal of f is a convex combination of the normals of the facets incident to v. Similarly, a pair of edges e_1 and e_2 form an ee-facet if the cross product of vectors parallel to e_1 and e_2 is a convex combination of the normals of the facets incident to e_1 and e_2.

These criteria allow us to test if a given pair of features (a facet/vertex pair or an edge pair) will produce a Minkowski sum facet by checking only the neighborhood of these features. Given a pair of features (facet/vertex or edge/edge), we say that the features are *compatible* if they form either an fv-facet or an ee-facet. When P_s transforms to P_t, some facets (i.e., pairs of features) in M_s will no longer be compatible. We call these facets the "errors" introduced by rotation.

A brute force algorithm, which is used in all existing methods except [17], computes the Minkowski sums from P_s and P_t without considering the correspondences between them as shown in Algorithm 3.1. Given P_s and Q and the existing Minkowski sum M_s. Algorithm 3.1 rotates P_s by θ to obtain P_t. Then it uses an existing Minkowski sum algorithm to compute M_t.

Algorithm 3.1: BRUTEFORCE(M_s, P_s, Q, θ)

$P_t = \text{Rotate}(P_s, \theta)$
$M_t = \text{MinkowskiSum}(P_t, Q)$
return (M_t)

4 DYNAMIC MINKOWSKI SUMS (DYMSUM)

In this section, we describe the details of the proposed method DYMSUM. Our goal is to take advantage of the correspondences between M_s and M_t that are completely ignored by Algorithm 3.1. Let us consider the Gaussian map again. M_s is computed by overlaying $g(P_s)$ and $g(Q)$. To obtain M_t, we need to find out which vertices in $g(P_s)$ are moved to another face in $g(Q)$ and determine if the edges of $g(P_s)$ intersect or stop intersecting with the edges of $g(Q)$ after rotating P_s to P_t . This is exactly what

DYMSUM does. That is, DYMSUM first determines these changes in the overlay introduced by the rotation, and then corrects the errors to generate the new Minkowski sum M_t. Therefore, the Minkowski sum M_t is composed of two types of facets: (1) the facets from M_s that still satisfy the aforementioned criteria after rotation and (2) the facets that are created due to the errors.

A sketch of DYMSUM is shown in Algorithm 4.1. In contrast to the brute-force method, DYMSUM is sensitive to the amount of rotation. That is, when θ is smaller, there will be fewer errors in the Gaussian map overlay. In this case, DYMSUM will likely take less time to compute the result than the naïve method. In the rest of this section, we will discuss how the errors are determined (Section 4.1) and how to correct these errors (Sections 4.2 and 4.3).

Algorithm 4.1: DYMSUM(M_s, P_s, Q, θ)

$P_t = \text{Rotate}(P_s, \theta)$
$E_t = \text{FindErrors}(M_s, P_t, Q)$
$M_t = \text{CorrectErrors}(E_t, M_s, P_t, Q)$
return (M_t)

4.1 Find Errors

There are two types of errors, fv-errors and *ee*-errors, corresponding to fv-facets and *ee*-facets, respectively. If a pair of features was compatible and becomes incompatible after the rotation of P, we call this pair an error.

Before we talk about how these errors can be identified, we will first show the relationship between the fv-errors and the *ee*-errors. Theoretically, the complexity of the Minkowski sum is $O(mn)$ and there can only be $O(m+n)$ fv-facets. Therefore, the number of fv-facets can be far smaller than the number of *ee*-facets. Moreover, it is easy to show that no *ee*-errors can occur if there are no fv errors.

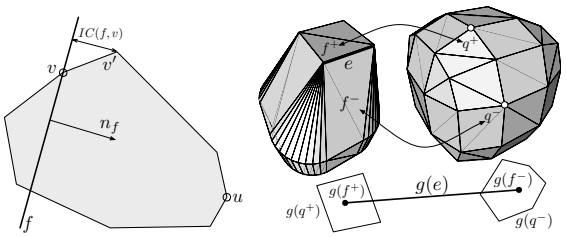
Theorem 4.1. **fv-errors and *ee*-errors must coexist.**

Proof. We first show that if there is an *ee*-error, there must be an fv-error. Let e and e' be a pair of edges that are compatible before rotation and become an *ee*-error after rotation. When e and e' are compatible, $g(e)$ and $g(e')$ must intersect and, after rotating P, $g(e)$ and $g(e')$ no longer intersect. This means at a certain point during the rotation, an end point of $g(e)$ must cross $g(e')$ or vice versa. When a point v crosses the edge $g(e)$, v changes the face with which it is associated from one side of $g(e)$ to the other side of $g(e)$. This change indicates that there must be an fv-error.

We then show that if there is an fv-error, there must be an *ee*-error. If a facet f of P and a vertex v of Q become an fv-error, we know that f now must be compatible with some other vertex $v' \neq v$ of Q. As a result, an edge $g(e)$ incident to $g(f)$ must be moved (or deformed) with $g(f)$. Since the faces in $g(Q)$ are convex and $g(e)$ cannot intersect with a segment more than twice, $g(e)$ must intersect with some new edges of Q when $g(f)$ moves from $g(v)$ to $g(v')$. This indicates that there must be at least one *ee*-error.

Therefore, fv-errors and *ee*-errors must coexist. □

Based on Theorem 4.1, we can find all errors by first exhaustively checking all fv-facets in M_s (the Minkowski sum before rotation) to find fv-errors. Then we use these fv-errors to identify all *ee*-errors. That is, if there are no fv-errors found, then we can immediately conclude that there are no *ee*-errors as well. Otherwise, the *ee*-errors must occur at the edges incident to the vertices involved in the fv-errors. Thus, it is clear that finding all errors will take $O(m+n)$ time.



(a) degree of incompatibility (b) correct an *ee*-error

Figure 2: (a) A 2-d cartoon shows the definition of $IC(f,v)$ and its witness $\overline{vv'}$. Using gradient decent, we will find the compatible vertex u for f. (b) Determine the associated edges for edge e.

4.2 Correct fv-errors

For each fv-error, we perform a gradient descent to compute a new fv-facet. More specifically, given a facet/vertex pair, we can measure the *degree of incompatibility* of the pair and attempt to iteratively minimize the incompatibility until a compatible pair is found.

Let f be a facet of P and v be a vertex of Q. When f and v are compatible, all the edges that are incident to v must be *below* or on the half-plane supported by f. When f and v are incompatible, we can define the *degree of incompatibility*

$$IC(f,v) = \max\{\mathrm{d}(e,f) \mid e \in E_v\} ,$$

where E_v is a set of edges incident to v and $\mathrm{d}(e,f)$ is the longest Euclidean distance from any point on e to f.

We say an edge e is the witness of the incompatibility if $\mathrm{d}(e,f)$ is $IC(f,v)$. Fig. 2(a) illustrates an example of fv-error and $IC(f,v)$.

In order to find the compatible pair, we find the witness of the incompatibility e, and replace v with the other end point $v' \neq v$ of e, and repeat this until f and v become compatible. In Fig. 2(a), this vertex is u. Since Q is convex, this procedure must be able to find a vertex of Q such that all of its incident edges are below f, therefore, will always terminate.

This process is equivalent to finding an extreme point at the outward normal direction of f and therefore can be done in $O(\log n)$ time if Q has n vertices.

4.3 Correct *ee*-errors

After all the fv-errors are corrected, the incident edges associated with these fv-errors are marked as *ee*-errors. Let e be such an edge from P and let f^- and f^+ be the facets in P incident to e. Our goal is to find the edges of Q that are compatible with e. An exhaustive search for compatible edges will certainly be slow. Fortunately, we can find the compatible edges using the results from fv-facets. That is, since we know that the incident facets f^- and f^+ both have the compatible vertices q^- and q^+ of Q, we can find the compatible edges for e using q^- and q^+. The relationships between e, $f^\pm$ and $q^\pm$ are shown in Fig. 2(b).

More specifically, if we overlay the Gaussian map $g(e)$ of e with $g(Q)$, $g(e)$ will intersect a set of faces in $g(Q)$ and the end points of $g(e)$ are inside $g(q^-)$ and $g(q^+)$. See the bottom of Fig. 2(b). If we can determine the rest of the faces intersected by $g(e)$, we can find the compatible edges for e. We further know that these faces form a connected component between $g(q^-)$ and $g(q^+)$, thus the compatible edges for e must be on the boundary of these faces. To find these Gaussian faces, we start from $g(q^-)$, and find an incident edge e' of $g(q^-)$ that is compatible with e. It is obvious that e' must exist unless $q^- = q^+$. From e', we replace q^- with the vertex $x' \neq q^-$ incident to e', and repeat the process until $q^- = q^+$.

The computation time is equal to the sum of the degree of vertices of Q visited during the search process.

5 EXPERIMENTAL RESULTS

In this section, we show that the computation time of DYMSUM is more efficient than the traditional approach (Algorithm 3.1) and is indeed sensitive to the amount of rotation applied to P. In our experiments, the polyhedron P rotates using a sequence of random quaternions. Each quaternion is applied to P for a random period of time. All the computation times that we will show later are obtained by averaging over 100 random rotations. All the experiments are performed on a machine with Intel CPUs at 2.13 GHz with 4 GB RAM. Our implementation is coded in C++.

Figure 3 shows 13 models that we will use in the first two experiments. Many of these models are from [4] and can be obtained from the authors' website. Theoretically, DYMSUM works with polyhedra tessellated with arbitrary polygons, but in our current implementation DYMSUM only takes triangulated polyhedra. Therefore, all the models in Figure 3 are triangulated.

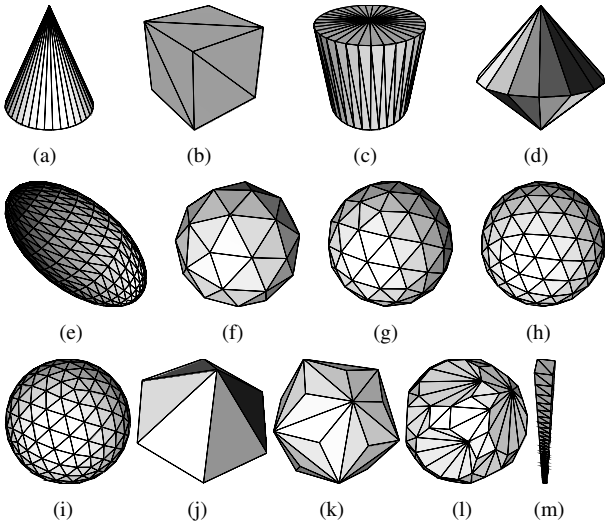


Figure 3: Models used in the experiments. (a) cone, 78 facets (b) cube, 12 facets (c) cylinder, 140 facets (d) dioctagonal dipyramid (DD), 32 facets (e) ellipse, 960 facets (f) geodesic sphere 1 (GS1), 80 facets (g) GS2, 180 facets (h) GS3, 320 facets (i) GS4, 500 facets (j) hexagonal pyramid (HP), 10 facets (k) triakis icosahedron (T), 60 facets (l) truncated icosidodecahedron (TI), 236 facets (m) v-rod, 324 facets.

5.1 Experiment 1: Dymsum vs. Brute-force method

In Table 1, we compare the proposed method, DYMSUM, to a brute-force method (Algorithm 3.1) that re-computes the Minkowski sum in every time step. The brute-force method checks the compatibility of all facet-vertex and edge-edge pairs every time that P rotates. The values in the table are t_d/t_{bf}, where t_d and t_{bf} are the (averaged) updating or re-computing times for DYMSUM and the brute-force method.

From Table 1, it is clear that DYMSUM is always faster than the brute-force method. Even for very simple cases, such as cone$\oplus$HP, DYMSUM is at least 8 times faster. For more complex examples, such as ellipse$\oplus$ellipse,

DYMSUM is about 176 times faster than the brute-force method.

5.2 Experiment 2: Computation time vs. Rotational resolution

In this experiment, we study the computation time of DYMSUM with respect to the rotational resolution of P. Our goal is to show that, in contrast to the brute force approach, DYMSUM is in fact sensitive to the magnitude of the rotation. In the problem of motion planning, this resolution defines the number of slices in mapping the configuration space. In the physically-based simulation, this value defines the number of collision detections and penetration depth estimations per second. Fig. 4 shows the results obtained using DYMSUM. Notice that the x axis is in logarithmic scale.

The x axis of Fig. 4 is the number of steps for P to make a full rotation. For example, when $x = 500$, P will take 500 steps to rotate 360° degree. That is, P rotates $\pi/250$ around a random axis every step. Therefore, when x is large, the changes in the Minkowski sum will be small. From the figure, we can see that the computation time drops quickly around $x = 500$ and then stabilizes below the 0.5 millisecond mark. In Experiments 1 and 3, we set $x = 500$.

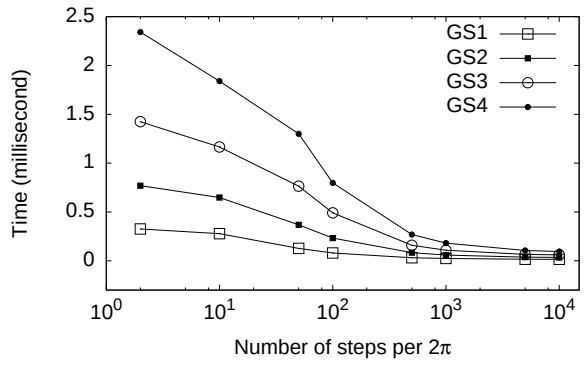


Figure 4: Computation time at different rotational speeds. More steps per 2π means slower rotational speed.

5.3 Experiment 3: Computation time vs. Input size

In this last experiment, we study the relationship between the computational time and the input size. We use a $10 \times 10 \times 10$ cube with different numbers of triangles tessellated on the surface. Fig. 5 shows that the computation time of the brute-force method increases rapidly while that of DYMSUM stays almost constant. When we show DYMSUM's computation time along in Fig. 5(b), DYMSUM's computation time is increased linearly along with the size of the cubes.

Recall that the complexity of a Minkowksi sum of two convex shapes is $O(mn)$, however the number of the fv-facets is $O(m+n)$. Therefore a large portion of the Minkowksi sum is composed of the *ee*-facets. In our experiment, we see a linear increase in computation time. We speculate that only a few errors occur at each step and most of the computation time is spent on verifying and updating the compatibility of the fv-facets.

6 CONCLUSION AND DISCUSSION

We have shown an efficient and robust method for recomputing 3-d Minkowski sums of convex polyhedra under rotation. Experimentally, we have shown that DYMSUM greatly outperforms the brute-force method of recomputing the Minkowski sum from scratch at each iteration.

Also, because the performance of the algorithm increases as the magnitude of the rotation between steps decreases, this method is much more useful than the naïve approach for generating $\mathscr{C}$-obst using discretized rotation intervals, as it enables the fast computation of higher resolution $\mathscr{C}$-obst approximations.

The main steps in our method are those of identifying and rectifying errors introduced by rotation. It is possible to efficiently compute this because of the observation that there are few fv-errors, and that *ee*-errors propagate locally to corresponding fv-errors. Instead of re-computing the entire Minkowski sum at a cost of $O(mn\log n)$ time, we are able to rectify the errors in significantly less time, which is the major efficiency gain from our method.

We hope to extend this method in the future to work with non-convex polyhedra. We also plan to apply this method to general $\mathscr{C}$-space mapping for convex polyhedra.

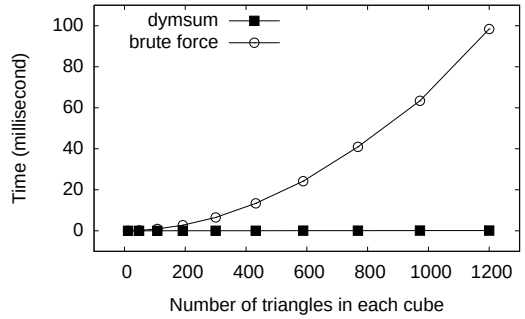
References

[1] P. K. Agarwal, E. Flato, and D. Halperin. Polygon decomposition for efficient construction of minkowski sums. In *European Symposium on Algorithms*, pages 20–31, 2000.

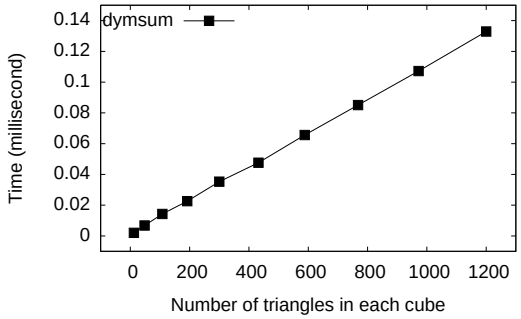
[2] C. Bajaj and T. K. Dey. Convex decomposition of polyhedra and robustness. *SIAM J. Comput.*, 21:339–364, 1992.

Table 1: The speedup of dymsum using the models in Fig. 3.
The values in the table are t_d/t_{bf}, where t_d and t_{bf} are the computation times for DYMSUMand the brute-force method.

cone	9.26												
cube	8.58	10.50											
cylinder	12.83	10.31	18.73										
DD	9.82	9.85	13.33	10.93									
ellipse	24.97	14.50	47.51	21.67	**176.34**								
GS1	15.09	11.21	23.66	15.39	51.38	25.23							
GS2	19.73	12.15	32.96	17.95	86.99	33.16	52.73						
GS3	22.92	12.42	40.45	19.63	120.23	38.88	65.53	86.26					
GS4	24.50	12.70	45.22	20.19	146.15	44.21	73.89	100.15	121.26				
HP	**8.02**	9.32	9.59	9.25	13.19	9.75	10.84	11.58	11.51	9.00			
T	14.50	11.16	20.66	14.05	35.95	20.33	26.01	29.21	31.83	10.28	16.17		
TI	21.30	13.87	35.55	19.53	91.48	38.02	56.86	69.25	78.26	12.78	28.08	63.45	
v-rod	20.30	18.31	34.00	21.92	88.30	46.53	65.46	77.73	83.05	15.76	34.23	67.24	123.75
	cone	cube	cylinder	DD	ellipse	GS1	GS2	GS3	GS4	HP	T	TI	v-rod



(a) DYMSUM vs. brute force



(b) DYMSUM only

Figure 5: Computation times of DYMSUM and brute force of two identical cubes. The numbers of triangles in the cubes are 12, 48, 108, 192, 300, 432, 588, 768, 972 and 1200.

[3] E. Flato. Robuts and efficient construction of planar minkowski sums. M.Sc. thesis, Dept. Comput. Sci., Tel-Aviv Univ., Isael, 2000.

[4] E. Fogel and D. Halperin. Exact and efficient construction of Minkowski sums of convex polyhedra with applications. In *Proc. 8th Wrkshp. Alg. Eng. Exper. (Alenex'06)*, pages 3–15, 2006.

[5] K. Fukuda. From the zonotope construction to the minkowski addition of convex polytopes. *Journal of Symbolic Computation*, 38(4):1261–1272, 2004.

[6] P. K. Ghosh. A unified computational framework for Minkowski operations. *Computers and Graphics*, 17(4):357–378, 1993.

[7] P. Gritzmann and B. Sturmfels. Minkowski addition of polytopes: computational complexity and applications to Gröbner bases. *SIAM J. Discret. Math.*, 6(2):246–269, 1993.

[8] L. J. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 100–111, 1983.

[9] L. J. Guibas and R. Seidel. Computing convolutions by reciprocal search. *Discrete Comput. Geom.*, 2:175–193, 1987.

[10] P. Hachenberger. Exact Minkowksi sums of polyhedra and exact and efficient decomposition of polyedra in convex pieces. In *Proc. 15th Annual European Symposium on Algorithms (ESA)*, pages 669–680, 2007.

[11] A. Kaul and J. Rossignac. Solid-interpolating deformations: construction and animation of PIPs. In *Proc. Eurographics '91*, pages 493–505, 1991.

[12] W. Li and S. McMains. A gpu-based voxelization approach to 3d minkowski sum computation. In

SPM '10: Proceedings of the 14th ACM Symposium on Solid and Physical Modeling, pages 31–40, New York, NY, USA, 2010. ACM.

[13] J.-M. Lien. Point-based minkowski sum boundary. In *PG '07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, pages 261–270, Washington, DC, USA, 2007. IEEE Computer Society.

[14] J.-M. Lien. Hybrid motion planning using Minkowski sums. In *Proc. Robotics: Sci. Sys. (RSS)*, Zurich, Switzerland, 2008.

[15] J.-M. Lien. Minkowski sums of rotating convex polyhedra. In *Proc. 24th Annual ACM Symp. Computat. Geom. (SoCG)*, June 2008. Video Abstract.

[16] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. Comput.*, C-32:108–120, 1983.

[17] N. Mayer, E. Fogel, and D. Halperin. Fast and robust retrieval of minkowski sums of rotating polytopes in 3-space. In *To appear in Proc. Symposium of Solid and Physical Modeling (SPM)*, 2010.

[18] M. Peternell, H. Pottmann, and T. Steiner. Minkowski sum boundary surfaces of 3d-objects. Technical report, Vienna Univ. of Technology, August 2005.

[19] S. Raab. Controlled perturbation for arrangements of polyhedral surfaces with application to swept volumes. In *SCG '99: Proceedings of the fifteenth annual symposium on Computational geometry*, pages 163–172, New York, NY, USA, 1999. ACM.

[20] G. Varadhan and D. Manocha. Accurate Minkowski sum approximation of polyhedral models. *Graph. Models*, 68(4):343–355, 2006.

[21] R. Wein. Exact and efficient construction of planar Minkowski sums using the convolution method. In *Proc. 14th Annual European Symposium on Algorithms*, pages 829–840, 2006.

[22] K. Wise and A. Bowyer. A survey of global configuration-space mapping techniques for a single robot in a static environment. *The International Journal of Robotics Research*, 19(8):762–779, 2000.

[23] L. Zhang, Y. J. Kim, G. Varadhan, and D. Manocha. Generalized penetration depth computation. *Comput. Aided Des.*, 39(8):625–638, 2007.