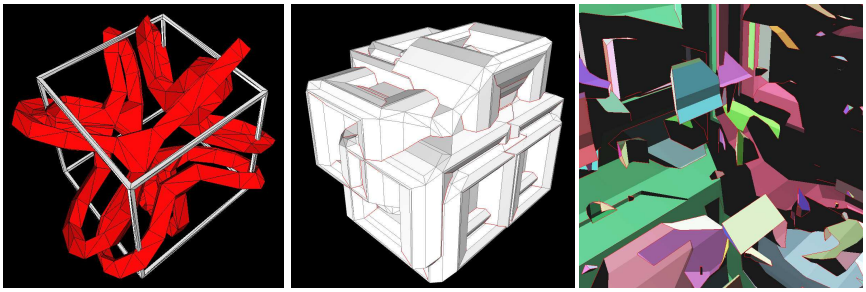
A Simple Method for Computing Minkowski Sum Boundary in 3D Using Collision Detection

Jyh-Ming Lien

Abstract: Computing the Minkowski sum of two polyhedra exactly has been shown difficult. Despite its fundamental role in many geometric problems in robotics, to the best of our knowledge, no 3-d Minkowski sum software for general polyhedra is available to the public. One of the main reasons is the difficulty of implementing the existing methods. There are two main approaches for computing Minkowski sums: divide-and-conquer and convolution. The first approach decomposes the input polyhedra into convex pieces, computes the Minkowski sums between a pair of convex pieces, and unites all the pairwise Minkowski sums. Although conceptually simple, the major problems of this approach include: (1) The size of the decomposition and the pairwise Minkowski sums can be extremely large and (2) robustly computing the union of a large number of components can be very tricky. On the other hand, convolving two polyhedra can be done more efficiently. The resulting convolution is a superset of the Minkowski sum boundary. For non-convex inputs, filtering or trimming is needed. This usually involves computing (1) the arrangement of the convolution and its substructures and (2) the winding numbers for the arrangement subdivisions. Both computations are difficult to implement robustly in 3-d. In this paper we present a new approach that is simple to implement and can efficiently generate accurate Minkowski sum boundary. Our method is convolution based but it avoids computing the 3-d arrangement and the winding numbers. The premise of our method is to reduce the trimming problem to the problems of computing 2-d arrangements and collision detection, which are much better understood in the literature. To maintain the simplicity, we intentionally sacrifice the exactness. While our method generates exact solutions in most cases, it does not produce low dimensional boundaries, e.g., boundaries enclosing zero volume. We classify our method as 'nearly exact' to distinguish it from the exact and approximate methods.

Jyh-Ming Lien
George Mason University, 4400 University Dr., Fairfax VA, 22030, USA, e-mail: jmlien@cs.gmu.edu



(a) (b) (c)

Fig. 1: Can you find a configuration that keeps the knot (in red) interlocked but without colliding with the cubic frame (in white) in the figure (a) above? Although it seems, from an external view (b), the Minkowski sum boundary of the knot and the frame models is simple, the inside view (c) shows that the Minkowski sum contains many holes. By placing the knot's reference point in one of these holes, the knot remains interlocked and collision free with the frame. There are in total 10 510 facets in this Minkowski sum boundary.

1 Introduction

Given two geometric models and their configurations in the space, such as the knot and the frame models shown in Fig. 1(a), there are several important questions that we can ask about these two models. For example, what is their shortest separation distance? Is it possible to physically separate the knot and the frame without intersections? If not, can we modify the knot, e.g., make the knot thinner, so the problem above becomes solvable? What are the set of the collision-free configurations that makes the knot and the frame interlocked? The answers to these questions play central and fundamental roles in algorithmic robotics, such as motion planning, penetration depth estimation, and object containment. However, all these questions are not easy to answer either visually or computationally due to the geometrical and topological complexity of the problem. In fact, these questions are all closely related to the concept of set sum (also known as the Minkowski sum). The Minkowski sum of two polyhedra P and Q is defined as:

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

In Fig. 1(b) and Fig. 1(c), we show the Minkowski sum of the knot and the frame. The inner view reveals a large number of holes in their Minkowski sum despite the simplicity of the input models. Indeed, computing the Minkowski sum of non-convex polyhedra can have the time complexity as high as $O(n^3m^3)$ [12], where m and n are the complexity of the input models.

Given two polyhedral models P and Q represented by their boundaries ∂P and ∂Q, the *boundary* of their Minkowski sum $\partial(P \oplus Q) \neq \partial P \oplus \partial Q$. Therefore, computing the boundary-based representation of the Minkowski sums is more than applying Eq. 1 to P and Q. Many methods have been proposed during the last three decades.

Even though several methods [13, 4, 8, 5] are known to compute the Minkowski sum of *convex* polyhedra efficiently in 3-dimensions, most approaches proposed for general polyhedra remain in theoretical stage. Only a few practical implementations exist and none of them are available to the public. We will provide a more detailed review on the related work in Section 2.

Our approach. An important goal of our work is to provide a simple method that can efficiently and accurately compute the Minkowski sum boundaries. The proposed method is based on *convolution*. The convolution of two polyhedra P and Q is a set of facets in 3-d that is generated by 'combining' the facets of P and Q and forms a superset of the Minkowski sum boundary of P and Q. Convolution will be defined more carefully in Sections 2 and 3.1.

Briefly, our method first generates the convolution and computes the facet-facet intersections within the convolution. These intersections then induce an arrangement of line segments embedded on each facet. The cells from all the (2-d) arrangements are then merged into 'simple regions' (defined in Section 3.4), which are then filtered so that only the regions on the boundary are kept. We deliberately avoid computing the 3-d arrangement and the winding numbers, which have been shown difficult to compute robustly. Our method is designed to tolerate inaccuracy in the convolution and depends only on solving the problems of 2-d arrangement and collision detection, which are much better understood in the literature. We will discuss the details of our method in Section 3.

Our method does not solve the problem of 3-d Minkowski sum entirely. The simplicity of our method is gained by sacrificing the exactness. That is our method provides only *nearly exact* Minkowski sum whose low dimensional boundaries, e.g., boundaries enclosing zero volume, are *not* identified. Fortunately, when P and Q do not interlock too tightly, the proposed method keeps all boundaries exact (although may still suffer from numerical errors), thus provides more accuracy than the approximate methods [19, 14] do. We should also point out that our method shares some similarity with our previous work on the point-based method [14]. Beside the difference in their representations (mesh vs. points), the proposed method provides significant improvements over the point-based method in terms of both quality and efficiency. We will carefully compare these two approaches in Section 4.

2 Related Work

During the last three decades, many methods have been proposed to compute the Minkowski sums of polygons or polyhedra; see more detailed surveys in [6, 19, 4] for the Minkowski sums of the models in boundary-based representation. Despite the large volume of work, most methods can be categorized into one of the two main frameworks: divide-and-conquer and convolution.

Divide-and-Conquer. In the divide-and-conquer framework, the input models are decomposed into components. Because computing the Minkowski sum of convex shapes is easier than non-convex shapes, convex decomposition (either surface

or solid) is widely used. The next step in this framework computes the pairwise Minkowski sums of the components. Finally, all these pairwise Minkowski sums are united to form the final Minkowski sum of the input shapes.

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 difficult to implement robustly in practice, in particular when the input shapes are complex. For example, it is known that creating solid convex decomposition robustly is difficult, e.g., it is necessary to maintain the 2-manifold property after the split [2]. In addition, Agarwal et al. [1] have shown that different decomposition strategies can greatly affect the efficiency of this approach. Hachenberger [11] 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. The decomposition step normally generates many components. Even though methods exist to perform union operation, no existing methods can robustly compute the union of thousands even millions of pairwise Minkowski sums. In particular, the size and the complexity of the geometry generated during the intermediate steps can be overwhelming. 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 [11] also studies how the order of the union operation affects the efficiency. To avoid this explicit union step, Varadhan and Manocha [19] proposed an approach that generates meshes approximating the Minkowski sum boundary using marching cube technique to extract the iso-surface from a signed distance field. They proposed an adaptive cell to improve the robustness and efficiency of their method. Because their approach still depends on convex decomposition, it still suffers from the excessive number of convex components from decomposition.

Convolution. The convolution of two shapes P and Q, denoted as $P \times Q$, 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 [9]. 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 their Minkowski sum [6], i.e., $\partial(P \oplus Q) \subset P \times Q$. 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 [10] 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. Slope diagram is closely related to *Gaussian map*, which has been recently used to compute to implement robust and efficient Minkowski sum computation of convex objects by Fogel and Halperin [4]. Kaul and Rossignac [13] proposed a linear time method to generate a set of Minkowski sum facets. Output sensitive methods that compute the Minkowski sum of polytopes in d-dimension have also been proposed by Gritzmann and Sturmfels [8] and Fukuda [5].

The main difficulty of the convolution-based methods is to remove the portion of the facets that are inside the Minkowski sum. Recently, Wein [20] shows 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 [18].

Point-Based Representation. Alternatively, points have been used to represent the Minkowski sum boundary. 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. Moreover, point-based representation can be generalized to higher dimensional motion planning problems [15].

Peternell et al. [17] proposed a method to compute the Minkowski sum of two solids using points densely sampled from the solids, and compute local quadratic approximations of these points. However, their method only identifies the outer boundary of the Minkowski sum using a regular grid, i.e., no hole boundaries are identified. This can be a serious problem in particular when we study problems in motion planning and penetration depth computation.

We proposed a completely different method [14] that guarantees to produce a point set *covering* the boundary. However, our method also has drawbacks. For example, a large number of points are required if the Minkowski sum has small features (e.g., the models in Fig. 9). In addition, our method treats each point independently. This is good for the purpose of parallelization but the local relationship between the neighboring points is completely ignored. The method proposed in this paper does not suffer from these problems.

3 Our Method

In this section, we begin to discuss more details about the proposed method. The proposed method is convolution based and comprises five main steps. Our method first computes the convolution using a brute force method (Section 3.1). Then, we identify all the intersecting facets in the convolution (Section 3.2). Next, each facet is subdivided into sub-facets from the facet-facet intersections (Section 3.3). All sub-facets are stitched into *simple regions* based on the properties that will be discussed later (in Section 3.4). A simple region is either entirely inside or entirely on the boundary of the Minkowski sum. Finally, we use a collision detector to remove the regions inside the Minkowski sum (Section 3.5). We conclude this section by providing a discussion on the benefits provided by the proposed method and its current limitations (Section 3.6).

3.1 Brute force convolution

We use a brute force method to compute the convolution because of its simplicity. As we will see in our experiments, the convolution step actually takes very little time (on average 0.4% of the entire computation), even using the brute force method, comparing to all the other steps.

Our brute force method checks all possible facet/vertex and edge/edge pairs of P and Q and keeps all the facets that satisfy the criteria stated in Observation 3.1. The result of the brute force convolution is a set of facets that reside in the interior and on the boundary of the Minkowski sum.

Given two polyhedra P and Q, the convolution of P and Q can only come from two sources [13]: (i) the facets, called fv-facets, generated from a facet of P and a vertex of Q or vice versa and (ii) the facets, called *ee*-facets, generated from a pair of edges from P and Q, respectively.

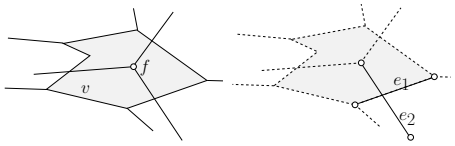


Fig. 2: Gaussian map of fv- (left) and *ee*- (right) facets.

Observation 3.1 *A facet f and a vertex v produce a valid fv-facet if and only if the normal of f is inside the region enclosed by the normals of the facets incident to v in the Gaussian map. Similarly, a pair of edges e_1 and e_2 form an ee-facet if the corresponding edges in the Gaussian map cross each other. Fig. 2 illustrates the necessary conditions of both fv- and ee-facets.*

Our goal in the next few steps is to remove the portions of the convolution inside the Minkowski sum.

3.2 Facet-facet intersections

The goal of this step is to identify all the intersecting facets for each facet in the convolution. To do so, we construct a bounding volume hierarchy from top-down using spheres that enclose all the facets. For each facet, we use its bounding sphere to identify all the intersecting spheres, which contain potential intersections. Finally, the intersecting facets are then determined from all these spheres. Because all the facets generated in the convolution must be convex if the input models have only convex facets, exact facet-facet intersection can be performed efficiently in 3-d. Without the loss of generality, we assume that the models used in this paper are composed of triangles.

3.3 Split facets

We use the intersections above to split the convolution facets. Essentially, this step computes the 2-d arrangements of the facet-facet intersections obtained from the previous step. For each facet, we project the intersections to the supporting plane of the facet. The arrangement embedded in the facet is induced by these projected line segments and the boundary of the facet. It should be noted that when the interior of a segment partially or entirely overlaps with other segments, we handle this degenerate case by creating cells with zero areas enclosed by the overlapped segments. As we will see later, these 'area-less' regions also serve as a form of 'insulator' to prevent the facets from being stitched.

For the facet without any intersections, we simply treat it as an arrangement with a single cell (two cells if we count the unbounded subdivision). To simplify our discussion, we call a cell created in this step a 'sub-facet.'

3.4 Stitch sub-facets

Our goal in this step is to stitch all the sub-facets into *simple regions*. A simple region is composed of a set of contiguous sub-facets that are completely on the Minkowski sum boundary or are completely inside the Minkowski sum. Our method constructs the simple regions by stitching the neighboring sub-facets iteratively until all sub-facets are stitched. We say that two sub-facets f_1 and f_2 are neighbors if they share an edge.

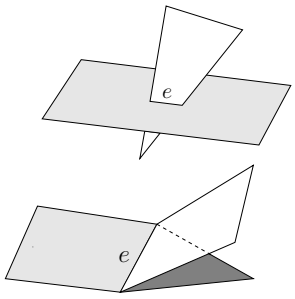


Fig. 3: Examples of facets that cannot be stitched.

Stitching criteria. Let C be an existing component and let f_1 be a facet on the boundary of C. We further let f_2 be a neighbor of f_1 that is not in C and let e_{12} be the edge shared by f_1 and f_2. Then f_1 and f_2 are stitched if they do not violate the following constraints.

1. e_{12} does not overlap with the intersections of the *interior* of the convolution facets.
2. e_{12} is 2-manifold.

Note that the first constraint can be readily checked from the intersection step earlier and is in fact a special case of the second constraint. This is because a pair of intersecting facets must generate a non-manifold region. The second constraint is used to check for non-manifold edges shared by more than two the adjacent (non-intersecting) sub-facets. Fig. 3 shows two examples that violate these criteria.

3.5 Determine the boundary regions

In this final step, we determine which simple regions are non-boundary regions and should be discarded using collision detection calls. Our method uses the close relationship between the Minkowski sum boundary and the concept of "contact space" in robotics. Every point in the contact space represents a configuration that places the robot in contact with (but without colliding with) the obstacles. Given a translational robot P and obstacles Q, the contact space of P and Q can be represented as $\partial((-P) \oplus Q)$, where $-P = \{-p \mid p \in P\}$. In other words, if a point x is on the boundary of the Minkowski sum of two polyhedra P and Q, then the following condition must be true:

$$(-P^\circ + x) \cap Q^\circ = \emptyset \ ,$$

where Q° is the open set of Q and $(P+x)$ denotes translating P to x.

Using this observation, we can determine if a simple region R is on the boundary by simply placing $-P$ at a point x sampled from a facet $f \in R$ and testing if $(-P+x)$ is in collision with Q. If $(-P+x)$ is collision free, then we can conclude that R is on the Minkowski sum boundary. Otherwise, we discard R.

3.6 Discussion and Implementation Details

The proposed method is simple and efficient, but it does not produce low dimensional (isolated) boundaries composed of only edges and vertices. In this section, we provide more detailed discussion regarding the implementation and the advantages and the limitations (and possible improvements) in some steps of the proposed method. The readers can also skip these details and go to Section 4 for experimental results.

Convolution. Our brute-force method does not compute exact 3-d convolutions, but a superset of the convolution. As far as we know, no practical method can compute the convolution of polyhedra exactly and robustly, even though methods exist to compute the convolution of polygons, such as the techniques in [10, 20]. Our method, unlike [20, 10], does not use the (mesh) connectivity of P and Q to construct the convolution, and, due to numerical errors, may generate 'isolated' facets in the final 'convolution' instead of a set of closed 2-manifolds. Note that all the isolated facets are inside the Minkowski sum boundary.

These two weaknesses of our brute-force method make the computations of the arrangement and the winding numbers even more difficult. However, because we intentionally avoid these two steps, our method does not suffer from the inaccuracy.

Given two polyhedra P and Q with size m and n, the brute-force method takes $O(mn)$ time. As we mentioned earlier, the convolution step is not the bottleneck of the entire computation. Even though computing the convolution from the non-planar Gaussian maps using a strategy similar to the ideas in [10, 20] can definitely increase the efficiency, the improvement to the entire computation is limited.

Facet-facet intersection. We use bounding sphere hierarchy to detect the intersections. We use spheres because they are invariant under rotation. This step takes $O((N+k)\log N)$ time, where $N = mn$ is the size of the convolution and k is the intersection size.

Stitch sub-facets. The idea of stitching is to maintain a set of the largest 2-manifolds from the convolution. We claim that each of this 2-manifold is a simple region. The criteria proposed to construct the simple regions (in Section 3.4) also focus this goal. In Lemma 0.1, we show that these two criteria is indeed sufficient to generate simple regions.

Lemma 0.1. *A simple region is either on the Minkowski sum boundary or in the interior of the Minkowski sum if the simple region is constructed using the criteria in Section 3.4.*

Proof (Sketch). Let C be the convolution of two polyhedra and let $A(C)$ be the arrangement of C. Essentially, a simple region identified in Section 3.4 is a set of contiguous sub-facets that form or entirely reside on the boundary shared by two (3-d) cells of $C(A)$. Since a cell must not cross the Minkowski sum boundary, the simple region will not cross the boundary. Thus, a simple region is either on the boundary or in the interior of the Minkowski sum.

Given the strong connection between the simple region and the arrangement cell, one might wonder if we can further stitch the simple regions into cells. There are several reasons that we do not go in this direction. First, given x cells in the arrangement of the convolution, there can be $O(x)$ simple regions, Therefore, further stitching regions into cells may not improve the efficiency (at least asymptotically). Second, this additional step greatly increases the difficulty of the implementation. Many degenerated cases, in particular with isolated regions, should be considered. In addition, from our preliminary results, little or no performance is gained by stitching further. Due to these reasons, we do not further stitch simple regions into arrangement cells.

Determine the boundary regions. We use collision detection calls to determine the type of a simple region. For detecting collisions, we use a modified version of RAPID [7]. An issue that we have to deal with when working with RAPID (and most collision detectors) is that RAPID cannot distinguish if two objects are in the contact configurations or are in fact in collision. To work around this problem, we perturb each point we sampled with an infinitesimally small vector pointing in the outward direction of the facet (from the convolution) where the point is sampled from. Note that the normal directions of all fv- and ee-facets are readily available from the convolution step.

After the perturbation, the point will most likely become collision free if it is indeed on the Minkowski sum boundary. The exceptions to the above case occur when the Minkowski sum boundary degenerates to an isolated vertex, edge or sliver (enclosing zero or a very small volume). This is the reason why our method provides only 'nearly' exact Minkowski sum.

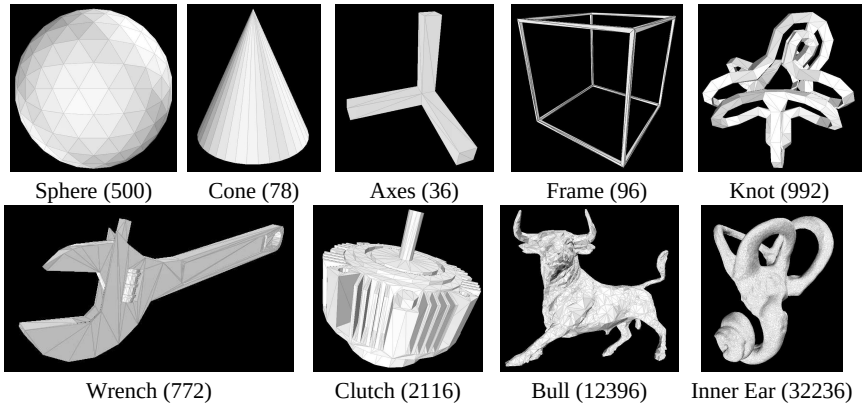


Fig. 4: Models used in this paper. The number following the model name is the number of facets of the model.

Another concern of using collision detection to replace winding number is the efficiency. However, as our experiment shows, collision detection, although dominates the computation in some examples, does not significantly slow down our method.

4 Experimental Results

In this section, we show experimental results. All the experiments are performed on a PC with two Intel Core 2 CPUs at 2.13 GHz with 4 GB RAM. Our implementation is coded in C++. For detecting collisions, we use a modified RAPID [7]. Fig. 4 shows a set of models used in this section. All the models and the Minkowski sum boundaries in our experiments are in Wavefront OBJ format and can be downloaded from our project webpage.

4.1 Geometric modeling

Our method can be used to perform operations such as offsetting, erosion, and sweeping on large geometric models. Fig. 5 shows an example of the offsetting operation of the clutch model. Offsetting is done by computing its Minkowski sum with a sphere. The top figure of Fig. 5 shows the Minkowski sum boundary (13 974 facets) of the clutch model and the sphere model. Each colored patch (best viewed from the submitted *pdf* file) on the Minkowski sum boundary indicates a *simple region* bounded by red line segments. Interestingly, for some models, the red line segments that separate simple regions tend to go through the areas with high concavity. Therefore, the simple regions seem to represent visually meaningful seg-

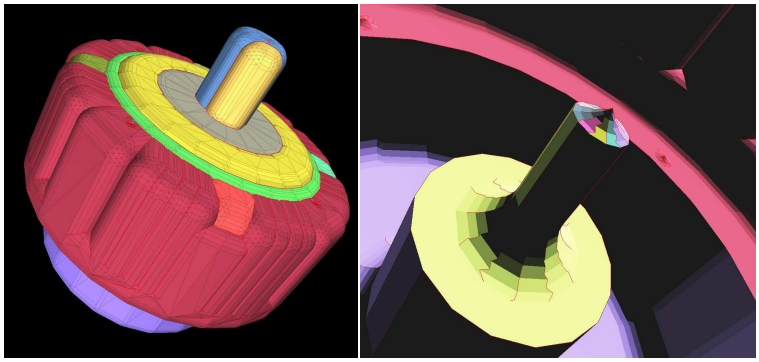


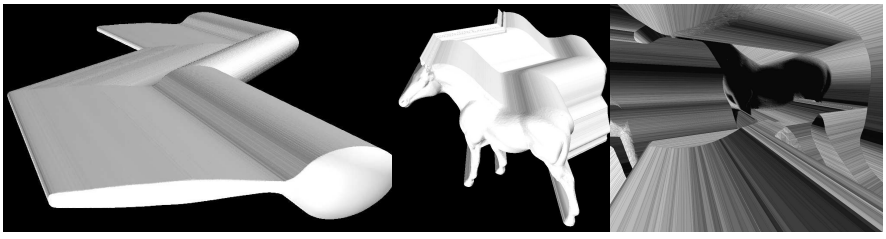
Fig. 5: Offsetting of the clutch model.

mentations of the model. The bottom figure of Fig. 5 shows an internal view of the Minkowski sum.

In Fig. 6, we show an example of the swept volumes of two large models: a spoon and a horse. The swept volume is generated by computing the Minkowski sum of the spoon and the horse models with a thin tube representing a trajectory. An internal view of the horse model's swept volume is also shown.

4.2 Computation time

A step-by-step analysis. Fig. 7 shows our first experiment result using the models in Fig. 4, which include convex/non-convex models, zero and non-zero genus models, and CAD and digitized models. These models are selected carefully to test the proposed method. In Fig. 7, we show the computation time of each Minkowski sum and the ratio of each step in an entire Minkowski sum computation. It is clear that



(a) (b) (c)

Fig. 6: (a) A swept volume of the spoon model (89 822 facets). The boundary is composed of 138 801 facets. (b) A swept volume of the horse model (39 694 facets). The boundary is composed of 73 912 facets. (c) An internal view of (b).

the facet-facet intersection and collision detection steps dominate the computation. We observe that the ratio of the creation time decreases when the size of the model increases. When the size of the model increases, the intersection step becomes more dominating. When the models have handles, the ratio of the collision detection increases due to the increasing number of holes (e.g., Frame and Knot).

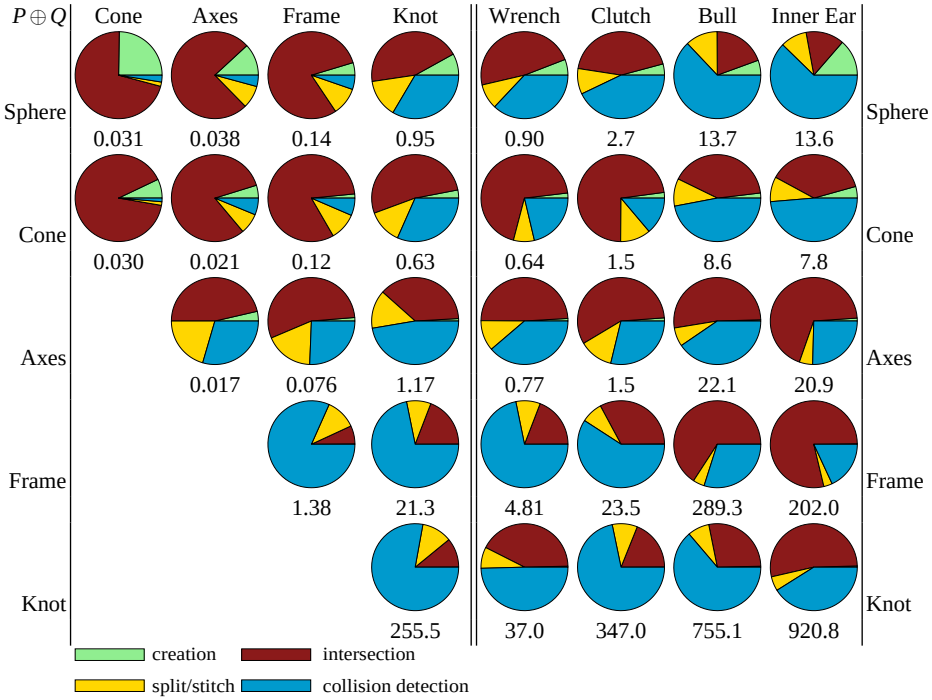


Fig. 7: Computation time of the proposed method. Each Minkowski sum computation is shown as a pie chart, representing the cost of each step, and a number below the chart, representing the total computation time (in seconds). Models used in this experiment can be found in Fig. 4.

Point-based vs. Mesh-based Minkowski sum. We compare the proposed method (hereafter named mesh-based method) to the point-based Minkowski sum [14] since it is the only implementation available to the public that supports general polyhedra. In order to make fair comparisons, we sample points from the facets generated by the mesh-based method. Like point-based Minkowski sum, these points form a d-covering[2] of the Minkowski sum boundary. It is obvious that when d is large point-based method can outperform mesh-based method. In Fig. 8, we vary the value of d from 10 to 0.05. As we can see that, as the value of d decreases, the computation time of the mesh-based method is slightly elevated while the collision detection call

[2] We say that a set of points S is a d-covering of a surface M if, for every point m of M, there exists a point in S whose distance to m is less than d.

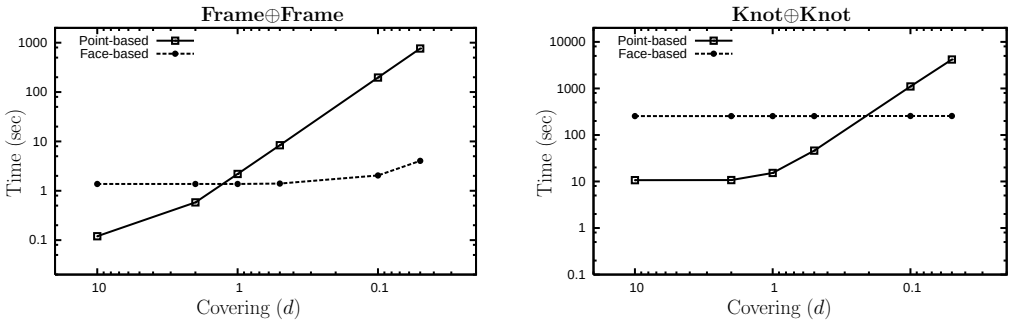


Fig. 8: Computation time for generating points covering the Minkowski sum boundaries. Notice that the *x* and *y* axes are both in logarithmic scale.

number remain the same. On the other hand, the point-based method slows down significantly as *d* decreases due to rapidly increasing detection calls.

In addition to the benefit of being faster than the point-based method, the mesh-based method proposed in this paper does not suffer from the sampling density issues. In particular, when small features are present in the Minkowski sum boundary, high density points (i.e., small *d*) are needed to reveal these features. In Fig. 9, we show a 'classic' example of two grate-like shapes, from which a large number points will need to be sampled in order to capture the long and skinny columns of the Minkowski sum boundary. Our mesh-based method does not suffer from this problem and successfully generates the exact Minkowski sum boundary.

5 Conclusion

In this paper we proposed a simple 3-d Minkowski sum method. In essence, our idea is to avoid computing the exact convolution, 3-d arrangement and the winding numbers. Instead, we filter and trim facets using only 2-d arrangements and collision detector. Our method starts with an inaccurate convolution generated by a brute force method. For each facet in the convolution, we subdivide the facet into sub-facets induced by the arrangement of the facet-facet intersections within the convolution. Sub-facets are then grouped into simple regions, which are filtered by a collision detector. Our method does not solve the problem of 3-d Minkowski sum entirely. The simplicity of our method is gained by sacrificing the exactness. Although providing only *nearly* exact Minkowski sum, our method is more accurate than the approximate methods. In our experiment, we demonstrated the proposed method's ability of handling large geometric models. We also showed the efficiency of the proposed method comparing to the point-based Minkowski sum method. While we are currently optimizing the performance of our implementation, we plan to release the software developed for this paper to the public.

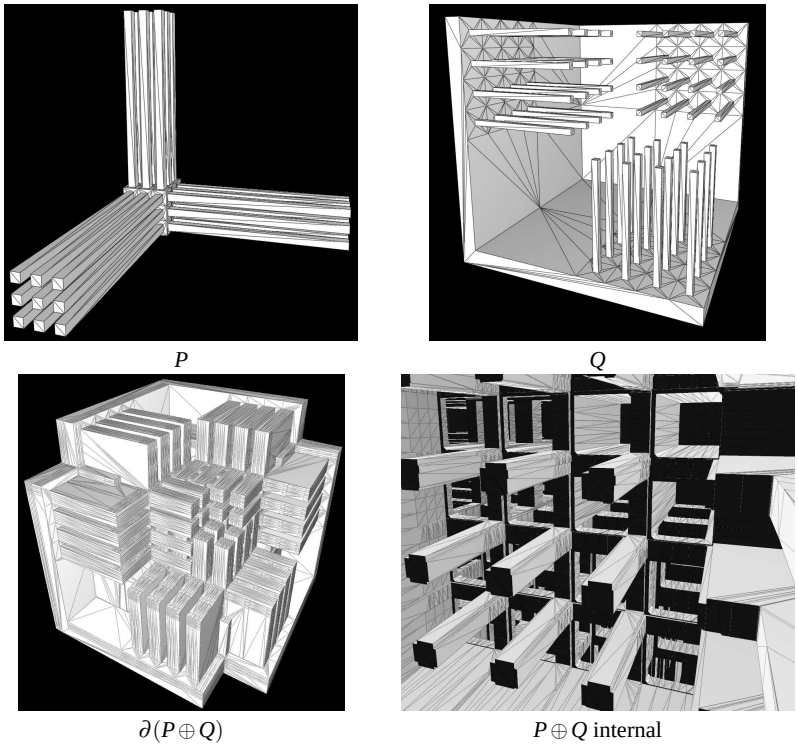


Fig. 9: Minkowski sum of two grate-like models. P has 27 teeth and 540 facets, and Q has 48 teeth and 942 facets, and $P \oplus Q$ has 71043 facets. The total computation time is 318.5 seconds using 1 thread. These models imitate the grate models created by Halperin [12] and from Varadhan and Manocha [19].

Acknowledgements

The geometric models used in this paper are from several sources. The sphere model is from Efi Fogel and Dan Halperin. The knot model is from Ergun Akleman. The wrench and clutch models are from the GAMMA group at UNC-Chapel Hill. The "dancing children" model is provided courtesy of IMATI-GE by the AIMSHAPE Shape Repository.

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.

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. S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. *Computer Graphics*, 30(Annual Conference Series):171–180, 1996.
8. 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.
9. 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.
10. L. J. Guibas and R. Seidel. Computing convolutions by reciprocal search. *Discrete Comput. Geom.*, 2:175–193, 1987.
11. 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.
12. D. Halperin. Robust geometric computing in motion. *The International Journal of Robotics Research*, 21(3):219–232, 2002.
13. A. Kaul and J. Rossignac. Solid-interpolating deformations: construction and animation of PIPs. In *Proc. Eurographics '91*, pages 493–505, 1991.
14. 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.
15. J.-M. Lien. Hybrid motion planning using Minkowski sums. In *Proc. Robotics: Sci. Sys. (RSS)*, Zurich, Switzerland, 2008.
16. T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. Comput.*, C-32:108–120, 1983.
17. M. Peternell, H. Pottmann, and T. Steiner. Minkowski sum boundary surfaces of 3d-objects. Technical report, Vienna Univ. of Technology, August 2005.
18. 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.
19. G. Varadhan and D. Manocha. Accurate Minkowski sum approximation of polyhedral models. *Graph. Models*, 68(4):343–355, 2006.
20. R. Wein. Exact and efficient construction of planar Minkowski sums using the convolution method. In *Proc. 14th Annual European Symposium on Algorithms (ESA)*, pages 829–840, 2006.