

Exact Minkowski Sums of Polyhedra and Exact and Efficient Decomposition of Polyhedra in Convex Pieces

Peter Hachenberger*

Department of Computing Science,
TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
phachenb@win.tue.nl

Abstract. We present the first exact and robust implementation of the 3D Minkowski sum of two non-convex polyhedra. Our implementation decomposes the two polyhedra into convex pieces, performs pairwise Minkowski sums on the convex pieces, and constructs their union. We achieve exactness and the handling of all degeneracies by building upon 3D Nef polyhedra as provided by CGAL. The implementation also supports open and closed polyhedra. This allows the handling of degenerate scenarios like the tight passage problem in robot motion planning.

The bottleneck of our approach is the union step. We address efficiency by optimizing this step by two means: we implement an efficient decomposition that yields a small amount of convex pieces, and develop, test and optimize multiple strategies for uniting the partial sums by consecutive binary union operations.

The decomposition that we implemented as part of the Minkowski sum is interesting in its own right. It is the first robust implementation of a decomposition of polyhedra into convex pieces that yields at most $O(r^2)$ pieces, where r is the number of edges whose adjacent facets comprise an angle of more than 180 degrees with respect to the interior of the polyhedron.

1 Introduction

The Minkowski sum of two point sets P and Q in \mathbb{R}^d , denoted by $P \oplus Q$, is defined as the set $\{p + q : p \in S_1, q \in S_2\}$. Minkowski sums are used in a wide range of applications such as robot motion planning [15], computer-aided design and manufacturing [8], penetration depth computation [14], offset computation [17], morphing [13], and mathematical morphological operations [18].

In several applications (e.g. in GIS or imaging) one deals with Minkowski sums of two-dimensional objects, and several implementations exist. When the two objects are non-convex polygons, two approaches are commonly used. The first approach computes the convolution of the boundary of two polygons [10].

* This work was partially supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301.

The other approach decomposes both polygons into convex pieces, computes the pairwise Minkowski sums of the pieces, and unites the pairwise sums. Both approaches have also been studied and implemented in combination with exact geometric computation [1, 20]. The *Library for Efficient Data Types and Algorithms* (LEDA) ¹ offers an exact implementation based upon the first method. The *Computational Geometry Algorithm Library* (CGAL) ² offers exact implementations of both methods.

There are also many applications, however, that require the computation of the Minkowski sums of three-dimensional objects. Examples can be found in CAD/CAM, assembly planning, and motion planning. Implementations of the 3D Minkowski sum exist, but they are neither exact nor robust. The most efficient such implementation is probably by Varadhan and Manocha [19]. It is based upon the convex decomposition approach as described above for the two-dimensional case. They guarantee the correct topology of the result, but are limited to manifold boundaries. Although many input objects are commonly two-manifolds, this limitation seems to be a major robustness issue, because the primitives of the union step, i.e., the Minkowski sum of two convex pieces, are not allowed to touch tangentially. Thus, at the moment there is no implementation available that is robust (that is, can deal with all possible degenerate cases), nor is there an implementation that is exact. This is the goal of our work: to provide a solution for the computation of Minkowski sums of 3D polyhedra that can handle all degenerate cases and is exact.

We present the first exact implementation of the Minkowski sum of two non-convex polyhedra. Our implementation is based on the convex decomposition approach. For the union step we use 3D Nef polyhedra [11] as provided by CGAL, which provide exact and efficient Boolean operations and handle all degeneracies. Our solution handles regularized solids with open or closed boundary. As a consequence, it can also be applied to degenerate scenarios like the tight passage problem in robot motion planning.

In addition to exactness, we also emphasize efficiency. We mostly concentrate on optimizing the time needed by the union of the pairwise Minkowski sums of convex pieces, which is the bottleneck of the used approach. For reducing the runtime of the union it is essential to have a decomposition that yields a low number of convex pieces. The decomposition into a minimum number of convex pieces is known to be NP-hard [16]. More than 20 years ago Chazelle proposed a decomposition method, which generates $O(r^2)$ convex pieces in $O(nr^3)$ time and $O(nr^2)$ space, where n is the complexity of the polyhedron and r is the number of *reflex edges*—edges, whose adjacent facets form an angle larger than 180 degrees with respect to the interior of the polyhedron. However, no robust implementation of this algorithm is known. Most of the practical methods perform surface decomposition or tetrahedral volumetric decomposition [5, 7, 12]. These methods generate $O(n)$ convex pieces of constant complexity.

¹ <<http://www.algorithmic-solutions.com>>

² <<http://www.cgal.org>>

As part of our Minkowski sum, we present the first robust implementation of a decomposition of a non-convex 3D polyhedron into $O(r^2)$ convex pieces. We use Chazelle’s main idea of inserting facets that resolve reflex edges. On the other hand, we construct the facets by a completely different method. Apart from the technical differences, we keep the actual amount of convex pieces low by scheduling the construction of the facets in an opportune way.

To optimize the union step itself, we develop different union strategies. The union of multiple polyhedra is done by consecutive binary union operations. Here, the order of these unions is essential for the runtime. Our union strategies use different heuristics to minimize the complexity of the intermediate results and to reduce the total amount of memory usage. We compare the efficiency of the heuristics experimentally.

The paper is organized as follows. In Section 2 we discuss how to efficiently decompose a non-convex polyhedron into convex pieces. The Minkowski sum of convex 3D polyhedra is discussed in Section 3. Section 4 compares multiple union strategies and refines the most promising. Also, we perform one larger experiment to get an idea of the performance. In Section 5, we briefly discuss the handling of tight passage problems. Finally, a conclusion is given in Section 6.

2 Decomposing a Polyhedron Into Convex Pieces

The problem of partitioning a polyhedron into convex pieces is more complex than its two-dimensional counterpart. In general it is not possible to decompose a polyhedron into simplices, i.e., into tetrahedra, without introducing Steiner points [16]. The decomposition of a polyhedron into a minimum number of convex pieces is known to be NP-hard [16].

A basic decomposition method was introduced and analyzed by Chazelle [4]. The idea is to remove each *reflex edge*, i.e., each edge whose adjacent facets have an angle larger than 180 degree with respect to the interior of the polyhedron, by inserting an additional facet that cuts the angle into two parts smaller than 180 degrees. Chazelle showed that a polyhedron with input complexity n and r reflex edges, can be decomposed into $O(r^2)$ convex pieces in $O(nr^3)$ time and $O(nr^2)$ space. He also provided an example for which the bound of $O(r^2)$ convex sub-polyhedra is tight.

We follow the common decomposition approach of inserting only vertical facets usually denoted as walls. A *wall* $W(e)$ of some non-vertical edge e is a connected subset of the vertical plane p_e that supports e . Walls were first defined by Aronov and Sharir [2]. Because their definition was given for a decomposition of the three-dimensional space with respect to a set of triangles we adapt their definition to our problem as follows: Let $A(p_e)$ be the planar arrangement of the intersection of the polyhedron (including previously erected walls) with the vertical plane p_e through e . Then, the wall of $W(e)$ consists of all faces of $A(p_e)$ that are incident to e and inside the polyhedron. The left graphic of Figure 1 illustrates the planar arrangement $A(p_e)$ and the wall $W(e)$.

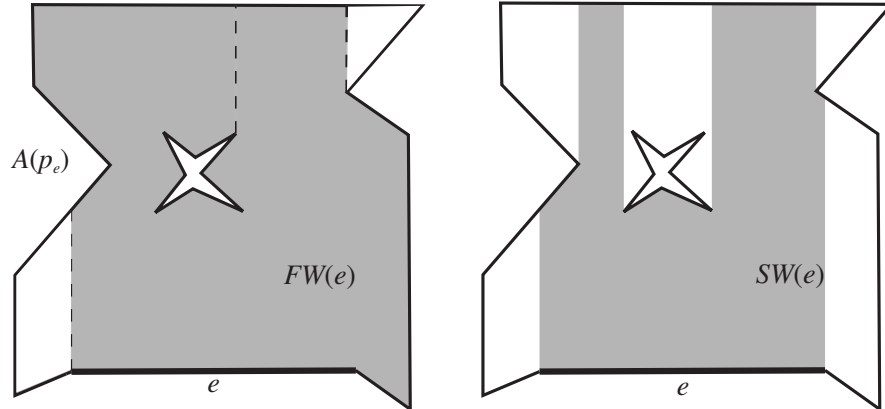


Fig. 1. *Left:* The flood wall $FW(e)$ consists of all facets adjacent to e in the planar arrangement $A(p_e)$ of the intersection of the polyhedron and the vertical plane p_e through e . The arrangement $A(p_e)$ may also contain previously inserted walls (dashed lines). *Right:* A sight wall $SW(e)$ covers all points that can be connected to e by a vertical edge without intersections.

Later de Berg, Guibas, and Halperin defined a vertical wall as the set of all points that can be connected to e via a vertical segment that does not intersect a face, edge, or vertex [6]. Adapting their definition to our setting, we consider points that can be connected to e via a vertical segment that completely lies within the polyhedron. The right graphic of Figure 1 illustrates the definition. To distinguish the two wall types, we further-on refer to the walls as defined in [2] as flood walls and to the walls as defined in [6] as sight walls.

The convex decomposition by vertical walls, also denoted as *vertical decomposition*, works in two steps. In the first step, vertical walls are erected for all non-vertical reflex edges. As a consequence, the decomposed volume becomes subdivided into cylindrical cells. In the second step, walls parallel to the yz -plane resolve the vertical reflex edges. Figure 2 illustrates the two steps of the vertical decomposition. Our decomposition deviates from the common approach to reduce the number of sub-polyhedra.

Using sight walls in the first phase often yields fewer, and never yields more pieces than using flood walls. This becomes clear if we consider the construction of two flood walls of reflex edges e and e' . Let us assume that $FW(e)$, if built first, seals a pocket in the boundary of the polyhedron. But if another flood wall $FW(e')$ is built before, it may intersect e and therefore split the pocket into two halves. Both halves of the pocket will later be sealed separately by the walls constructed for the two halves of e . Thus, depending on the building order of the flood walls, such a pocket can be decomposed into multiple pieces, although fewer pieces (or even one piece) are sufficient. Using sight walls instead, no $SW(e')$ splits the pocket into two halves; vertical segments cannot intersect such a pocket.

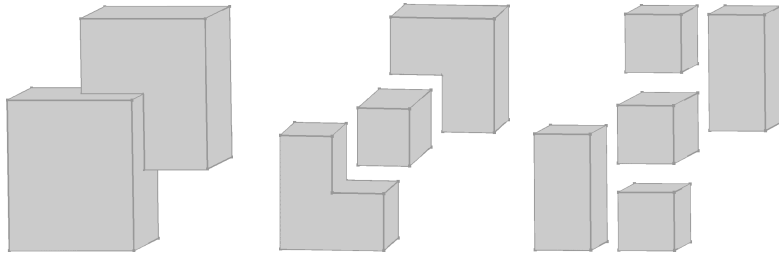


Fig. 2. Vertical decomposition based on the insertion of walls (viewed from the top). In the first step, the polyhedron is decomposed into xy -monotone sub-cells. Then, further vertical walls are inserted to subdivide the cells into convex sub-cells.

Because the reflex edges in the second phase are vertical, the definition of sight walls cannot be applied to these edges. Also, there is no unique vertical supporting plane that defines the flood wall of a reflex edge. The procedure of vertical decomposition suggests to create y -vertical flood walls. We deviate from this procedure to reduce the number of walls and therefore the number of sub-polyhedra. An y -vertical flood wall divides a volume into two or three parts. Instead, we insert a flood wall along one of the two facets adjacent to the reflex edge. This way, resolving a y -vertical reflex edge splits exactly one volume into two parts.

In the second step of the decomposition, walls can be built easily. Given a reflex edge e , let p_e be the plane in which we intend to build the wall $W(e)$, and let c be the cylindrical cell that will be decomposed by it. Then, $W(e)$ can be created by walking along the intersection of c with p_e and adding the incidences of the new facets to each encountered item. In the first step, the boundary of a wall is more complex. The boundary of $W(e)$ may not only consist of e and intersections with c , but also of intersections with other walls. If the walls are built in random order it is not guaranteed that those intersections exist when constructing a wall. Therefore, we cannot just walk along the boundary. To allow using the walk, we schedule the construction of the walls in such a way that all boundary parts of a wall exist in the moment of its construction. To do this, we must resolve mutually and cyclic dependencies.

The wall $W(e)$ of a reflex edge e may consist of two parts—of points below and points above e . We refer to these two parts as the lower wall $W^-(e)$ and upper wall $W^+(e)$. Often, the lower wall $W^-(e)$ is part of the boundary of some upper wall $W^+(e')$, or vice versa. Therefore, the lower and the upper walls are created in two separate sessions.

Starting with the lower parts, we want to sort the reflex edges, and thereby schedule their construction, from bottom to top. In general, the edges of a polyhedron—and the same holds for the reflex edges of a polyhedron—can not be sorted along a given direction. There can always be cyclic dependencies as illustrated by Figure 3. But, as we will see in the following, it is possible to resolve the dependencies.

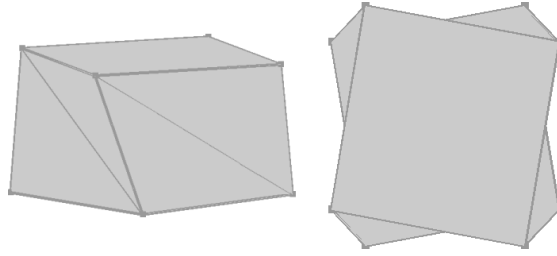


Fig. 3. Variation of Schönhardt polyhedron with a quadratic base viewed from the side and from the top. The diagonals of the sides are reflex edges, which are circular dependent on one another.

We sort the reflex edges by their lower endpoints. As a result, every pair of reflex edges has one of three relations. If (a) they do not overlap vertically, the sorting schedules the edges well. The same holds, if they overlap vertically, but their projection onto the xy -plane does not intersect. If (b) they overlap vertically and their projections onto the xy -plane intersect internally, the lower wall of one of them may be part of the other's boundary. Let's assume such an edge pair e and e' , where $W^-(e)$ is part of the boundary of $W^-(e')$. If e has the smaller lower endpoint the schedule works nicely; otherwise the walk cannot proceed along $W^-(e')$'s boundary because $W^-(e)$ is missing. If (c) they share a common endpoint, the lower walls share a boundary edge. Their construction is mutually dependent.

To solve the problems described above, we cut some reflex edges into two or more parts and insert vertical edges from the endpoints of all reflex edges to the bottom of the polyhedron. The inserted vertical edges exactly resemble the common boundary parts of walls with common endpoints and therefore resolve the problem described in (c). Before inserting the vertical edges, we search the sorted list of reflex edges for situation (b). If we find an edge e part of the boundary of $W^-(e')$, but e' is scheduled before e , then we cut e at the intersection point v_i with the vertical plane $p_{e'}$ supporting e' and put the two edge halves at their proper positions into the sorted set. Later a vertical edge will also be inserted between v_i and the bottom of the polyhedron. It exactly resembles the intersection between $W^-(e)$ and $W^+(e')$. Note that the split of a reflex edge performed to resolve (b) can be unnecessary if e cannot be seen from e' . However, such a split does not introduce an additional convex piece. The upper walls can be handled in the same way as the lower walls.

In the second step of the decomposition, no wall intersects a reflex edge or introduces new reflex edges. Also there are no mutual and cyclic dependencies. Since we use multiple directions, the supporting planes usually intersect and therefore the decomposition is not unique. But the number of sub-polyhedra is constant among all orders of wall creation.

Like the decomposition of Chazelle [4], our decomposition clearly yields at most $O(r^2)$ polyhedra. The wall of a non-vertical reflex edge e can intersect

with each other reflex edge at most once. This does not change if other walls cut e into multiple parts, because all sub-walls have the same supporting plane as $W(e)$. Therefore the first step generates at most $O(r^2)$ cylindrical cells and $O(r^2)$ vertical reflex edges. Then, each vertical reflex edge exactly splits one cell into two. The worst case runtime of our implementation is worse than Chazelle's since we use kd-tree based ray shooting for the insertion of the vertical edges and for the walk along the boundary.

3 The Minkowski Sum of Convex Polyhedra

The Minkowski sum of two convex polyhedra is also a convex polyhedron. Furthermore, it is well known that each vertex $v_{P\oplus Q}$ of the Minkowski sum $P\oplus Q$ is the vector sum of vertices v_P in P and v_Q in Q [15]. Hence, a trivial solution for the Minkowski sum of two convex polyhedra P and Q computes the convex hull of all vector sums of vertex pairs of P and Q . This algorithm performs a convex hull computation on pq vertices, where p and q are the number of vertices in P and Q . Thus, using CGAL's `convex_hull_3` function the trivial algorithm runs in $O(pq \log(pq))$ time.

A more efficient solution can be obtained by using normal diagrams. Each convex polyhedron P has a unique dual representation N_P called the *Gaussian diagram* or *normal diagram*. It is a subdivision of the sphere into vertices, edges and faces, such that the outward-directed normal directions of all planes supporting some item of P constitute an item of N_P . For a facet of P there is exactly one plane supporting it. Thus, its dual item is the single point on the sphere with the same normal direction as the supporting plane. The normal directions of the planes supporting an edge e_P of P form a great arc on the sphere. The endpoints of the great arc are dual items of the facets incident to e_P . A face f_n on N_P is the dual item of a vertex v_p of P . f_n is bound by a convex cycle of edges and vertices, which are the dual items of the edges and facets incident to v_p . The order of the edges and vertices around f_n coincides with the order of dual items around v_p .

The faces of $N_{P\oplus Q}$ are intersections of faces of N_P and N_Q . What is more, the dual face of $v_{P\oplus Q}$ is the intersection of the dual faces of v_P and v_Q with $v_P + v_Q = v_{P\oplus Q}$. As a consequence, the overlay of N_P and N_Q is the normal diagram of the Minkowski sum $P\oplus Q$. Also, the exact point set of $P\oplus Q$ can easily be obtained by storing the primal vertices with their respective dual face and computing the vector sums for each face in the overlay. Thus, using the overlay of normal diagrams improves on the trivial algorithm in two points. First, the construction of $P\oplus Q$ operates on the exact set of vertices, which might be far smaller than pq . However, in the worst case, $P\oplus Q$ still has $O(pq)$ vertices. And second, the incidence structure of $N_{P\oplus Q}$ allows us to construct $P\oplus Q$ from it in time linear to $P\oplus Q$.

With Nef polyhedra embedded on the sphere [11] as provided by CGAL, we already have a tool that can be used to realize normal diagrams, and for which we also have an overlay algorithm that can be reused for the Minkowski sum.

The overlay algorithm also allows to store arbitrary data with each vertex, edge, and face, and to propagate this data properly during the overlay. Therefore, the missing operations are the two conversions between a convex three-dimensional polyhedron and its normal diagram. Both functions are easy to implement.

Spherical Nef polyhedra are not the most efficient solution for the overlay of normal diagrams. Asymptotically, there is no essentially superior solution, but the *Cubical Gaussian Map* of Fogel and Halperin is clearly faster [9]. The binary operations on spherical Nef polyhedra can handle more complex overlays than those of normal diagrams, which are always convex arrangements; they never include nested faces or lower dimensional features. Therefore, our overlay algorithm is obviously more costly than needed. Apart from that, the spherical predicates are too expensive.

For the runtime experiments of this paper, the spherical Nef polyhedra are sufficient, since the runtime of computing the Minkowski sum of the convex parts is always much smaller than the union of these partial solutions. For a future release of our implementation of non-convex Minkowski sum computations, we plan to exploit other efficient implementations of algorithms that compute Minkowski sums of convex polyhedra such as the one based on Cubical Gaussian Maps [9], or the one based on Arrangement on Surfaces [3].

4 Uniting a Set of Polyhedra

The union of the Minkowski sums of the convex sub-polyhedra is done by multiple binary union operations of 3D Nef polyhedra. Since the complexity of the binary union operation depends on the complexities of both input and the result polyhedron in equal shares, it is essential not to perform the binary operations in arbitrary order. The trivial method for instance maintains one Nef polyhedron holding the current intermediate result. It starts with an empty polyhedron and adds the polyhedra one by one. This method performs very badly, since most of the union operations involve at least one big polyhedron, namely the intermediate result. Experiments showed that examples that can be computed in less than 10 minutes with efficient methods, run for more than a day with the trivial approach. Clever methods unite small polyhedra first, and try to keep intermediate results as small as possible. Since we cannot foresee the optimal order, we develop and test different strategies.

Our first method performs in a greedy fashion. It maintains the set of all unhandled primitives and intermediate results, and unites the two smallest polyhedra in each step. This can be realised by a priority queue. The priority of a polyhedron is its size measured by the number of its vertices. The priority queue is initialized with all pairwise Minkowski sums of the convex pieces. Then, repeatedly the two smallest polyhedra are extracted from the queue, and their union is inserted into the queue. The method terminates with the result left as the final remaining element in the queue.

Our second strategy tries to unite neighboring polyhedra to reduce the size of the intermediate results. For this purpose, we put the primitives into a queue and

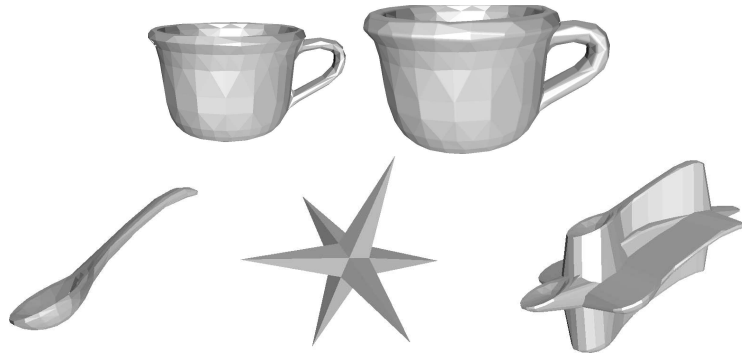


Fig. 4. Example Minkowski sums. Top: cup and cup \oplus sphere. Bottom: spoon, star, and spoon \oplus star.

sort the queue by the lexicographically smallest vertex of the primitives. In order to unite polyhedra of the same complexity, we proceed similar to the priority queue approach. We extract and unite the first two polyhedra, and append their result to the end of the queue. The neighboring relation used for the sorting is maintained throughout the whole process. Sorting the polyhedra by their lexicographically smallest vertex does not specify how we compare two such vertices in the sorting function. We test three comparison types: lexicographical comparison of the coordinates, comparison of the L1 distance from a point in a corner of the scenery, and comparison of the L2 distance from the origin.

Performing larger examples, it becomes obvious that memory is major issue in either of the above strategies. The queue-based approach can be adapted, such that no more than $\frac{\log p}{2}$ need to be stored, where p is the number of primitives. Instead of a queue we maintain a stack. The primitives are computed and inserted to the stack one by one. After pushing the i th primitive onto the stack, we $\lfloor \log i \rfloor$ times pop and unite the respective top two items and push the result back on the stack. Note that every binary union (besides the ones after the insertion of the final primitive) combines two polyhedra that are unions of the same number of primitives. Although we construct primitives just before they are pushed on the stack, we also want to sort the set of primitives in advance. For this purpose, we additionally store all normal diagrams and a sorted list of all ordered pairs of pointers to the normal diagrams that schedules the creation of the primitives. The list is sorted by the sum of the smallest vertices of the respective sub-polyhedra. Again we test the same three comparison types.

Table 2 summarizes the tests of the strategies. The stack strategy proved to be superior to the others. This becomes clearer the larger the examples get. The difference between the comparison types is very small. Lexicographical comparisons show the best result. The last line of the table shows the runtime of a much bigger example.

	cube	ball1	ball2	star	spoon	mushroom	cup
facets	6	128	1000	24	336	448	1000
parts	1	1	1	5	186	255	774

Table 1. Models used in the experiments.

model1	model2	priority queue	queue			stack			convex decomp	convex sum
			lexi	L1	L2	lexi	L1	L2		
mushroom	cube	170	151	151	152	147	147	149	43	14
mushroom	ball1	608	529	534	545	408	415	423	43	102
spoon	star	963	930	921	925	755	726	732	43	84
cup	ball2					8497			415	939

Table 2. Performance of the different union strategies, the decomposition and the Minkowski sum of the convex pieces. For each model the number of facets and the number of convex sub-polyhedra are listed. The runtimes are given in seconds.

5 Tight Passages

Computing the Minkowski sum of a tight passage scenario requires the handling of open polyhedra. A Nef polyhedron stores set selections marks for every vertex, edge, facet, and volume, which indicate whether the respective item is part of the polyhedron. In case of an open polyhedron the marks of the boundary items are unselected. These selection marks can additionally be stored in the normal diagram. During the overlay operation the marks are transferred in the following way. Given normal diagrams N_P and N_Q , we consider intersecting items i_P and i_Q . Their intersection forms item $i_{P \oplus Q}$ in the overlay $N_P \oplus N_Q$. Then, the selection mark stored with $i_{P \oplus Q}$ can be computed as $i_P \wedge i_Q$. This means for instance, that a vertex $v_{P \oplus Q} = v_P + v_Q$ of $P \oplus Q$ is selected iff v_P and v_Q are selected.

If we allow arbitrary selection mark for the input, i.e., each boundary part may have a different mark, the computation of the Minkowski sum becomes more complicated. The reason is, that each side of a convex sub-polyhedron may consist of several facets, some of which are boundary parts of the input and some of which are walls inserted by the decomposition. The selection marks of these facets can differ. As a consequence, the selection marks cannot be handled as described above.

Fortunately, we don't need arbitrary selection marks to handle tight passages. It is only necessary to allow polyhedra that are either open or closed, i.e., all selection marks of boundary items are either selected or unselected. In this situation it suffices to ignore the selection marks of the walls. If the side of a sub-polyhedron consists of original facets and walls, we use the uniform mark of the original facets; if there are only walls, we can assign an arbitrary mark. Naturally, the walls must be selected because they represent a point set inside a polyhedron. Thus, not selecting a wall yields an unselected facet in a convex Minkowski sum pr_1 that should be selected. On the other hand, this wall is adjacent to another sub-polyhedron. Because of the adjacency and because

of the convexity of primitives, the convex Minkowski sums computed on this other sub-polyhedron must generate a primitive pr_2 that overlaps pr_1 such that the wrongly unselected facet is in the interior of pr_1 . The final result of the Minkowski sum operation will not contain any of these facets.

Figure 5 shows a tight passage scenario solved with our implementation.

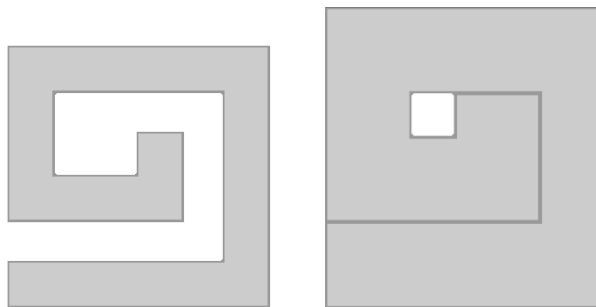


Fig. 5. A maze whose corridors have unit width, and the Minkowski sum of a unit cube and the maze.

6 Conclusion

We have presented an exact implementation of the Minkowski sum of non-convex 3D polyhedra. Our implementation also supports open and closed polyhedra, which allows to solve extreme scenarios like tight passage problems.

As part of our Minkowski sum, we have also presented the first robust decomposition of non-convex 3D polyhedra into $O(r^2)$ convex pieces, where r is the number of reflex edges.

As part of future work, we want to further improve the efficiency of our implementation and release it as part of CGAL. We plan two major points of improvement. First, we want to improve the second step of the decomposition by adapting polygon decomposition methods for the decomposition of the cylindrical cells. Those methods would sometimes resolve two vertical reflex edges with one wall and therefore generate fewer convex pieces. As a second step, we plan to replace our solution for the convex Minkowski sums by a faster approach.

Acknowledgements

The author likes to thank Lutz Kettner for stimulating the work of this paper. I thank Efi Fogel for helping me repeat the comparison of theirs and our implementation of the Minkowski sum on convex polyhedra. Furthermore, I thank Liangjun Zhang and Dinesh Manocha for providing 3D models, such that I could repeat two of the experiments in [19]. Also, I would like to thank Mark de Berg for valuable discussions on the presented topic.

References

1. P. K. Agarwal, E. Flato, and D. Halperin. Polygon decomposition for efficient construction of Minkowski sums. *Comp. Geom.: Theory and Appl.*, 21:39–61, 2002.
2. B. Aronov and M. Sharir. Triangles in space or building (and analyzing) castles in the air. In *SCG '88: Proceedings of the fourth annual symposium on Computational geometry*, pages 381–391, New York, NY, USA, 1988. ACM Press.
3. E. Berberich, E. Fogel, D. Halperin, and R. Wein. Sweeping over curves and maintaining two-dimensional arrangements on surfaces. In *Proc. 23rd Workshop on Computational Geometry (EWCG'07)*, pages 223–226, 2007.
4. B. Chazelle. Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM Journal on Computing*, 13(3):488–507, 1984.
5. B. Chazelle, D. Dobkin, N. Shouraboura, and A. Tal. Strategies for polyhedral surface decomposition: an experimental study. *Comput. Geom. Theory Appl.*, 7(5-6):327–342, 1997.
6. M. de Berg, L.J. Guibas, and D. Halperin. Vertical decompositions for triangles in 3-space. In *SCG '94: Proceedings of the tenth annual symposium on Computational geometry*, pages 1–10, New York, NY, USA, 1994. ACM Press.
7. S. A. Ehmann and M. C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. In *Comp. Graph. Forum (Proc. Eurographics 2001)*, volume 20(3), pages 500–510. 2001.
8. G. Elber and M.-S. Kim, editors. *Special Issue of Computer Aided Design: Offsets, Sweeps and Minkowski Sums*, volume 31, 1999.
9. E. Fogel and D. Halperin. Exact and efficient construction of Minkowski sums of convex polyhedra with applications. In *7th Workshop on Algorithm Engineering and Experiments (ALENEX 06)*, 2006. to appear.
10. L. J. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *24th Symp. on Found. of Comp. Sci. (FOCS)*, pages 100–111, 1983.
11. P. Hachenberger, L. Kettner, and K. Mehlhorn. Boolean operations on 3D selective Nef complexes: Data structure, algorithms, optimized implementation and experiments. *Comp. Geometry: Theory and Applications*, 38(1–2):64–99, 2007.
12. B. Joe. A software package for the generation of meshes using geometric algorithms. *Advances in Engineering Software and Workstations*, 13(5–6):325–331, 1991.
13. A. Kaul and J. Rossignac. Solid-interpolating deformations: Construction and animation of pips. *Computers & Graphics*, 16(1):107–115, 1992.
14. Y. J. Kim, M. A. O., M. C. Lin, and D. Manocha. Fast penetration depth computation using rasterization hardware and hierarchical refinement. In *Proc. of Workshop on Algorithmic Foundations of Robotics*, 2002.
15. J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.
16. J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, Inc., New York, NY, USA, 1987.
17. J R Rossignac and A A G Requicha. Offsetting operations in solid modelling. *Comput. Aided Geom. Des.*, 3(2):129–148, 1986.
18. C. Rössl, L. Kobbelt, and H.-P. Seidel. Extraction of feature lines on triangulated surfaces using morphological operators. In *Proc. of the 2000 AAAI Symp.*, 2000.
19. G. Varadhan and D. Manocha. Accurate Minkowski sum approximation of polyhedral models. In *Proc. Comp. Graphics and Appl., 12th Pacific Conf. on (PG'04)*, pages 392–401. IEEE Computer Society, 2004.
20. R. Wein. Exact and efficient construction of planar Minkowski sums using the convolution method. In *14th Europ. Symp. Alg. (ESA 06)*, pages 829–840, 2006.