# Robust and Efficient Construction of Planar Minkowski Sums*

Eyal Flato  Dan Halperin
Department of Computer Science
Tel Aviv University, Tel-Aviv 69978, Israel
{flato, halperin}@math.tau.ac.il

## Abstract

The Minkowski sum (also known as the vector sum) of two sets $P$ and $Q$ in $\mathbb{R}^2$ is the set $\{p + q \mid p \in P, q \in Q\}$. Minkowski sums are useful in robot motion planning, computer-aided design and manufacturing (CAD/CAM) and many other areas. We present a software package for robust and efficient construction of Minkowski sums of planar polygonal sets. We describe the different algorithms that we implemented and an experimental comparison between them. A distinctive feature of our implementation is that it can accurately handle degenerate input and in particular it can identify degenerate "holes" in the Minkowski sum which consist of a line segment or a singular point.

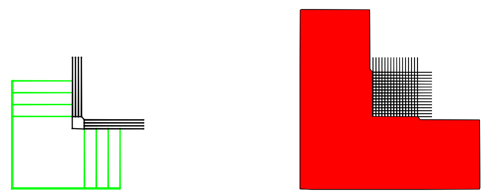

Figure 1: Fork input: $P$ and $Q$ are polygons with $m$ and $n$ vertices each having horizontal and vertical teeth. The complexity of $P \oplus Q$ is $\Theta(m^2n^2)$

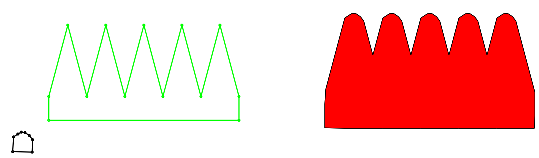

Figure 2: If $P$ is a single convex polygon, then a result of Kedem et al. [10] implies that $P \oplus Q$ has $\Theta(mn)$ vertices.

## 1 Introduction

Given two sets $P$ and $Q$ in $\mathbb{R}^2$, their *Minkowski sum* (or vector sum), denoted by $P \oplus Q$, is the set $\{p + q \mid p \in P, q \in Q\}$. Minkowski sums are used in a wide range of applications, including robot motion planning [11], assembly planning [8], and computer-aided design and manufacturing (CAD/CAM ) [2].

Consider for example an obstacle $P$ and a robot $Q$ that moves by translation. We can choose a reference point $r$ rigidly attached to $Q$ and suppose that $Q$ is placed such that the reference point coincides with the origin. If we let $Q'$ denote a copy of $Q$ rotated by 180° degrees, then $P \oplus Q'$ is the locus of placements of the point $r$ where $P \cap Q \neq \emptyset$. In the study of motion planning this sum is called a *configuration space obstacle* because $Q$ collides with $P$ when translated along a path $\pi$ exactly when the point $r$, moved along $\pi$, intersects $P \oplus Q'$.

There has been much work on obtaining sharp bounds on the size of the Minkowski sum of two sets in two and three dimensions, and on developing fast algorithms for computing Minkowski sums. It is well known that if $P$ is a polygonal set with $m$ vertices and $Q$ is another polygonal set with $n$ vertices, then $P \oplus Q$ is a portion of the *arrangement* (see Section 2) of $mn$ segments, where each segment is the Minkowski sum of a vertex of $P$ and an edge of $Q$, or vice-versa. Therefore the size of $P \oplus Q$ is $O(m^2n^2)$ and it can be computed within that time; this bound is tight in the worst case [9]; see Figure 1. The sum has lower worst-case complexity when one of the polygons or both are convex; see for example Figure 2.

We devised and implemented three algorithms for computing the Minkowski sum of two polygonal sets

*This work has been supported in part by ESPRIT IV LTR Projects No. 21957 (CGAL) and No. 28155 (GALIA), and by a Franco-Israeli research grant (monitored by AFIRST/France and The Israeli Ministry of Science). D.H. has also been supported by a grant from the U.S.-Israeli Binational Science Foundation, by The Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (Center for Geometric Computing and its Applications), and by the Hermann Minkowski – Minerva Center for Geometry at Tel Aviv University.

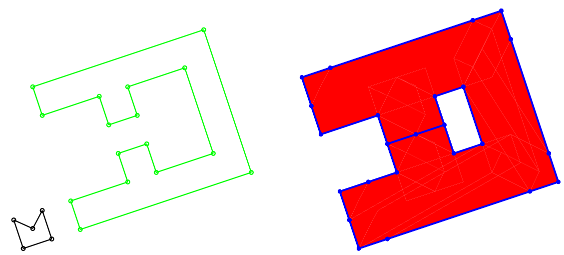

Figure 3: Tight passage: the desired target placement for the small polygon (left-hand side) is inside the inner room defined by the larger polygon. In the configuration space (right-hand side) the only possible path to achieve this target passes through the line segment emanating into the hole in the sum.

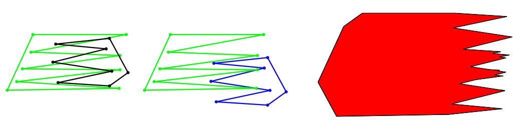

Figure 4: Minimal distance separation

based on the CGAL software library [1, 5]. Our main goal was to produce a *robust* and exact implementation. This goal was achieved by employing the CGAL *planar map* package [6] while using exact number types. We are currently using our software to solve translational motion planning problems in the plane. We are able to compute collision-free paths even in environments cluttered with obstacles, where the robot could only reach a destination placement by moving through tight passages, practically moving in contact with the obstacle boundaries. See Figure 3 for an example. This is in contrast with most existing motion planning software for which tight or narrow passages constitute an insurmountable hurdle.

We also solve the following polygon separation problem:[1] Given two polygons find the minimum length translation of one polygon relative to the other that will make the two polygons interior disjoint; see Figure 4. Assuming that in their original placement $P$ and $Q$ intersect and that the reference point of $Q$ is at the origin $O$, it is not difficult to see that the minimum translation of $Q$ relative to $P$ is described by the point on the boundary of $P \oplus Q'$ which is closest to $O$ where $Q'$ is a copy of $Q$ rotated $180°$ degrees.

All our algorithms start with decomposing the input polygons into convex subpolygons. We discovered that the choice of the decomposition can have a dramatic effect on the running time of the Minkowski-sum algorithms. In a companion paper we investigate what constitute good decompositions for efficient construction of Minkowski sums [3]. Throughout the experiments that we describe here we use the same decomposition which has proved very efficient for our purposes. It is based on a heuristic method proposed in [4] which we have slightly improved; for details see [3].

In the next section we survey the Minkowski sum algorithms that we implemented. Experimental comparison between the algorithms is given in Section 3.

## 2 Minkowski Sum Algorithms

Given a collection $\mathcal{C}$ of curves in the plane, the *arrangement* $\mathcal{A}(\mathcal{C})$ is the subdivision of the plane into vertices, edges and faces induced by the curves in $\mathcal{C}$. *Planar maps* are arrangements where the curves are pairwise interior disjoint. Our algorithms for computing Minkowski sums rely on arrangements and planar maps, and in the discussion below we assume some familiarity with these structures, and with a refinement thereof called the *vertical decomposition*; we refer the reader to [7, 14] for information on arrangements and vertical decomposition, and to [6] for a detailed description of the planar map package in CGAL on which our algorithms are based.

The input to our algorithms are two *polygonal sets* $P$ and $Q$ (each being an arbitrary collection of simple polygons), with $m$ and $n$ vertices respectively. Our algorithms consist of the following three steps:

**Step 1:** Decompose the polygons of $P$ into convex subpolygons $P_1, P_2, \ldots, P_s$ and the polygons of $Q$ into convex subpolygons $Q_1, Q_2, \ldots, Q_t$.

**Step 2:** For each $i \in [1..s]$ and for each $j \in [1..t]$ compute the Minkowski *subsum* $P_i \oplus Q_j$ which we denote by $R_{ij}$. We denote by $R$ the set $\{R_{i,j} \mid i \in [1..s], j \in [1..t]\}$.

**Step 3:** Construct the union of all the polygons in $R$, computed in Step 2; the output is represented as a planar map.

The Minkowski sum of $P$ and $Q$ is the union of the polygons in $R$. Each $R_{ij}$ is a convex polygon, and it can easily be computed in time that is linear in the sizes of $P_i$ and $Q_j$ [11]. Let $k$ denote the overall number of edges of the polygons in $R$, and let $I$ denote the overall number of intersections between (edges of) polygons in $R$

We briefly present three different algorithms for performing Step 3, computing the union of the polygons in $R$, which we refer to as the *arrangement* algorithm, *incremental union* algorithm and *divide-and-*

[1] This question was posed by Marc van Kreveld as a *cartographic generalization* problem in the Dagstuhl meeting on Computational Geometry in March 1999.

*conquer* algorithm.

**Arrangement algorithm.** The algorithm constructs the arrangement $\mathcal{A}(R)$ induced by the polygons in $R$ (we refer to this arrangement as the *underlying arrangement* of the Minkowski sum) by adding the polygons of $R$ one by one in a random order and by maintaining the vertical decomposition of the arrangement of the polygons added so far; each polygon is chosen with equal probability at each step. Once we have constructed the arrangement, we traverse all its cells (vertices, edges or faces) and we mark a cell as belonging to the Minkowski sum if it is contained inside at least one polygon of $R$. The construction of the arrangement takes randomized expected time $O(I + k \log k)$ [13]. The traversal stage takes $O(I + k)$ time.

**Incremental union algorithm.** In this algorithm we incrementally construct the union of the polygons in $R$ by adding the polygons one after the other in random order. We maintain the planar map representing the partial union of polygons in $R$. For each $r \in R$ we insert the edges of $r$ into the map and then remove redundant edges from the map. All these operations can be carried out efficiently using the planar map package. We could only give a naive bound on the running time of this algorithm, which in the worst case is higher than the worst-case running time of the arrangement algorithm. Practically however the incremental union algorithm works much better than the arrangement algorithm on most problem instances.

**Divide and Conquer algorithm.** This algorithm is a combination of both previous algorithms, attempting to overcome the shortcomings of both. First we use the *incremental union* algorithm to compute the Minkowski sums of $P$ and $Q_j$ for each $1 \leq j \leq t$. This results in $t$ polygonal sets (each represented as a planar map) $S_1, \ldots, S_t$, where $S_j$'s complexity is $O(n|Q_j|)$ [10]. In the second phase we compute the union of pairs of maps from $S_1, \ldots, S_t$ using the *arrangement* algorithm, obtaining $t/2$ new maps. We continue to compute union of pairs of maps $\log t$ times until we end up with one map describing the Minkowski sum of $P$ and $Q$.

## 3 Experiments

We present experimental results of applying the algorithms described in the previous section to a collection of input pairs of polygonal sets. The input data that we present here is just a small representative sample of the polygonal sets on which tests were performed.

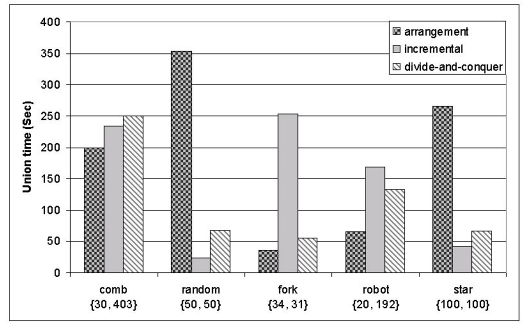


Figure 5: Running times for computing the Minkowski sum of different input data with all three union algorithms. The sizes of $P$ and $Q$ are in parenthesis.

Our implementation of the Minkowski sum package is based on the CGAL (version 2.0) [1] and LEDA (version 4.0) [12] libraries. Our package works with Linux (g++ compiler) as well as with WinNT (Visual C++ 6.0 compiler). The tests were performed under WinNT workstation on a 266 MHz PentiumII machine with 64 Mb of RAM. We measured the running times for the various algorithms with different input data.

The Minkowski sum of a *comb* and a convex polygon has complexity $\Theta(mn)$ (see Figure 2), while the *fork* input results in $\Theta(m^2n^2)$ Minkowski sum complexity (see Figure 1). The rest of the input data results in Minkowski sum complexity that is between $\Theta(m+n)$ and $\Theta(m^2n^2)$. The *star* input consists of two star shaped polygons. In the *robot* input $P$ is a star shaped robot and $Q$ is a set of convex obstacles. The *random* input consist of two random-looking polygons. The results of running the three union algorithms are presented in Figure 5.

We can see that for polygonal sets for which the Minkowski sum is complex (e.g. the fork input) the arrangement algorithm performs better. When the sum is relatively small (e.g. the star input) the incremental algorithm has the best running times. The divide-and-conquer algorithm performance is mostly between the other two algorithms and is likely to be closer to the best algorithm.

Another factor that affects the running time of the union algorithm is the order in which the polygons of $R$ are inserted into the planar map. Consider for example the *covered fork* input data (suggested to us by R. Wenger). It consists of two fork polygons whose Minkowski sum has complexity $\Theta(m^2n^2)$ and two long triangles whose Minkowski sum is a large hexagon that covers (contains) the grid created by the sum of the fork polygons. Therefore, The Minkowski sum in this case has $\Theta(mn)$ vertices while the underlying arrangement has $\Theta(m^2n^2)$ vertices. If we use the incremental algorithm and insert the large hexagon first, we can avoid handling the (complex) grid planar-map and we get output sensitive running time. This example shows that an algorithm that inserts the subsum polygons of $R$ in random order cannot be output sensitive.

If we insert the polygons of $R$ into the map in descending order of *fatness* (we use here a very simple measure of fatness— the area divided by the diameter squared) we will get the desired output-sensitivity effect in this special case. The results are given in Figure 6. This permutation, however, does not always result in better running times. Consider for example Figure 7 where all the thinner polygons of $R$ are intersecting the fatter polygons. We can see in the results that for this input (*fat grid*) the union time when using the fatness permutation is about two times slower than when using a random permutation.

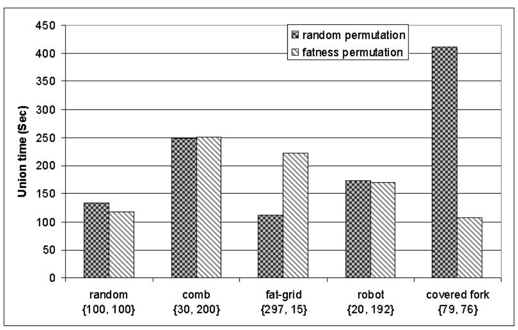


Figure 6: Running times for computing the Minkowski sum of different input data using the incremental algorithm and both random and fatness permutations on the polygons of $R$. The sizes of $P$ and $Q$ are in parenthesis.

**Acknowledgement**

The authors thank Pankaj Agarwal for discussions and valuable comments on this work.

## References

[1] *The CGAL User Manual, Version 2.0*, 1999. http://www.cs.ruu.nl/CGAL.

[2] Special issue: Offsets, sweeps and Minkowski sums. *Comput. Aided Design*, 31(4), 1999.

[3] P. K. Agarwal, E. Flato, and D. Halperin. Polygon decomposition for efficient construction of Minkowski sums. Manuscript. http://www.math.tau.ac.il/~flato/TriminkWeb, 1999.

[4] B. Chazelle and D. P. Dobkin. Optimal convex decompositions. In G. T. Toussaint, editor, *Computational Geometry*, pages 63–133. North-Holland, Amsterdam, Netherlands, 1985.

[5] A. Fabri, G. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the design of CGAL, the Computational Geometry Algorithms Library. Technical Report MPI-I-98-1-007, MPI Inform., 1998.

[6] E. Flato, D. Halperin, I. Hanniel, and O. Nechushtan. The design and implementation of planar maps in CGAL. In J. Vitter and C. Zaroliagis, editors, *Proceedings of the 3rd Workshop on Algorithm Engineering*, volume 1148 of *Lecture Notes Comput. Sci.*, pages 154–168. Springer-Verlag, 1999. Full version: http://www.math.tau.ac.il/~flato/WaeHtml/index.htm.

[7] D. Halperin. Arrangements. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 21, pages 389–412. CRC Press LLC, Boca Raton, FL, 1997.

[8] D. Halperin, J.-C. Latombe, and R. H. Wilson. A general framework for assembly planning: The motion space approach. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 19–28, 1998. To appear in *Algorithmica*.

[9] A. Kaul, M. A. O'Connor, and V. Srinivasan. Computing Minkowski sums of regular polygons. In *Proc. 3rd Canad. Conf. Comput. Geom.*, pages 74–77, Aug. 1991.

[10] K. Kedem, R. Livne, J. Pach, and M. Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.*, 1:59–71, 1986.

[11] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.

[12] K. Melhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[13] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[14] M. Sharir and P. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, 1995.

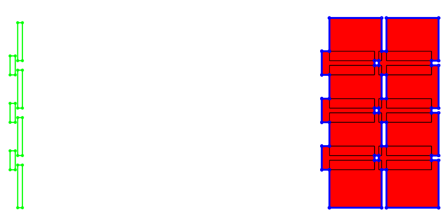

Figure 7: Fat grid input on the left. The Minkowski sum of the vertical polygons with the horizontal polygons is on the right.