ELSEVIER

Computational Geometry 21 (2002) 39–61

Computational Geometry
Theory and Applications

www.elsevier.com/locate/comgeo

Polygon decomposition for efficient construction of Minkowski sums [☆]

Pankaj K. Agarwal [a,1], Eyal Flato [b,*,2], Dan Halperin [b,2,3]

[a] *Department of Computer Science, Duke University, Durham, NC 27708-0129, USA*
[b] *School of Computer Science, Tel Aviv University, Tel-Aviv 69978, Israel*

Communicated by M.J. Katz and K. Kedem; received 4 September 2000; received in revised form 9 February 2001; accepted 9 February 2001

Abstract

Several algorithms for computing the Minkowski sum of two polygons in the plane begin by decomposing each polygon into convex subpolygons. We examine different methods for decomposing polygons by their suitability for efficient construction of Minkowski sums. We study and experiment with various well-known decompositions as well as with several new decomposition schemes. We report on our experiments with various decompositions and different input polygons. Among our findings are that in general: (i) triangulations are too costly, (ii) what constitutes a good decomposition for one of the input polygons depends on the other input polygon – consequently, we develop a procedure for simultaneously decomposing the two polygons such that a "mixed" objective function is minimized, (iii) there are optimal decomposition algorithms that significantly expedite the Minkowski-sum computation, but the decomposition itself is expensive to compute – in such cases simple heuristics that approximate the optimal decomposition perform very well. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Polygon decomposition; Minkowski sum; Convex decomposition; Experimental results

[☆] A preliminary version of this paper appeared in the proceedings of the 8th European Symposium on Algorithms – ESA 2000.

[*] Corresponding author.

E-mail addresses: pankaj@cs.duke.edu (P.K. Agarwal), flato@post.tau.ac.il (E. Flato), danha@post.tau.ac.il (D. Halperin).

[1] Supported by Army Research Office MURI grant DAAH04-96-1-0013, by a Sloan fellowship, by NSF grants EIA-9870724, EIA-997287 and CCR-9732787 and by a grant from the U.S.–Israeli Binational Science Foundation.

[2] 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).

[3] 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.

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 [25], assembly planning [15], computer-aided design and manufacturing (CAD/CAM) [8], and marker making (cutting parts from stock material) [6,30].

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°, 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 π exactly when the point r, moved along π, intersects $P \oplus Q'$. See Fig. 1.

Motivated by these applications, 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* of $O(mn)$ 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 [19] (see Fig. 2). If both P and Q are convex, then $P \oplus Q$ is a convex polygon with at most $m+n$ vertices, and it can be computed in $O(m+n)$ time [25]. If only P is convex, then a result of Kedem et

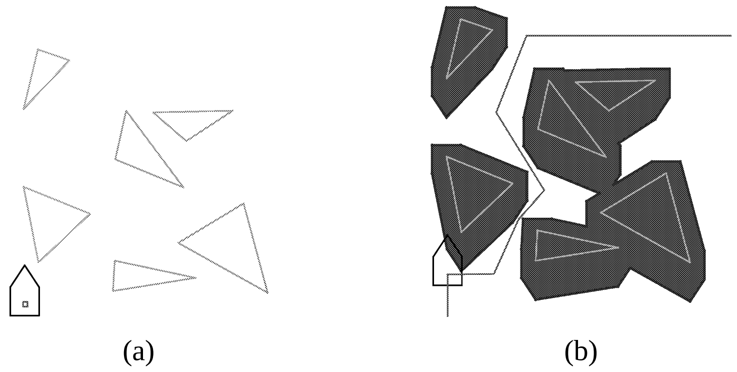

(a) (b)

Fig. 1. Robot and obstacles. (a) A reference point is rigidly attached to the robot. (b) The configuration space obstacles and a free translational path for the robot.

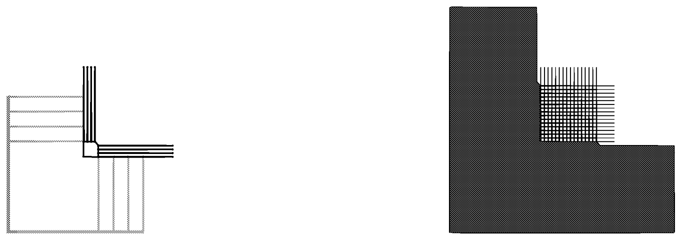


Fig. 2. P and Q are polygons with m and n vertices, respectively, each having horizontal and vertical teeth. The complexity of $P \oplus Q$ is $\Theta(m^2n^2)$.

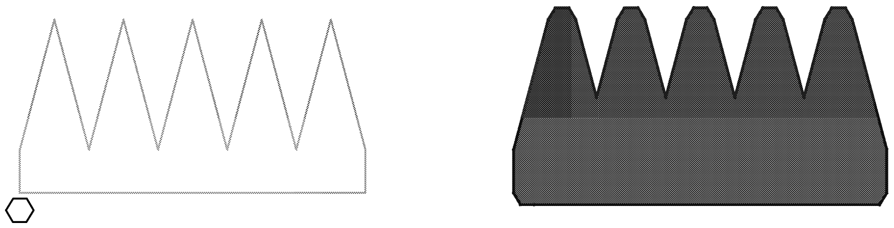


Fig. 3. Comb input: P is a convex polygon with m vertices and Q is a comb-like polygon with n vertices. The complexity of $P \oplus Q$ is $\Theta(mn)$.

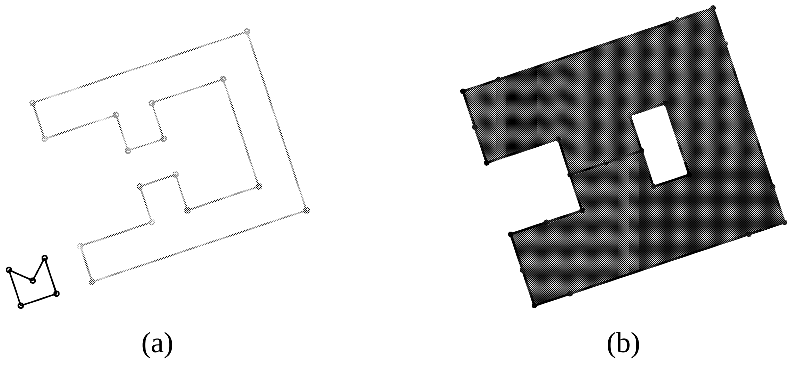


Fig. 4. Tight passage: the desired target placement for the small polygon is inside the inner room defined by the larger polygon (a). In the configuration space (b) the only possible path to achieve this target passes through the line segment emanating into the hole in the sum.

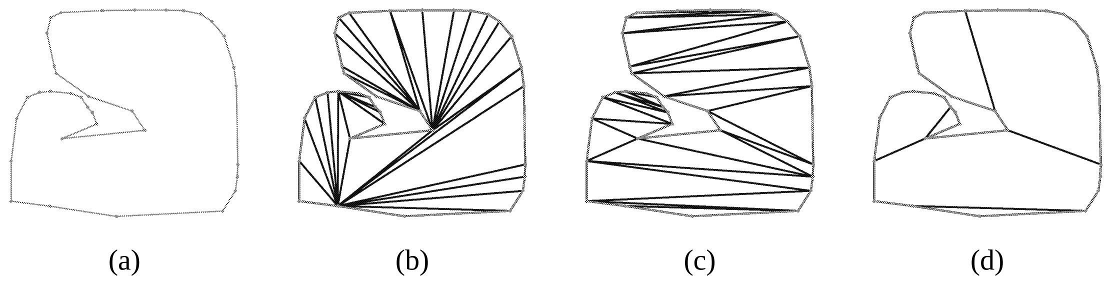
al. [20] implies that $P \oplus Q$ has $\Theta(mn)$ vertices (see Fig. 3). Such a Minkowski sum can be computed in $O(mn\log(mn))$ time [27].

Minkowski sums of curved regions have also been studied (e.g. [3,18,26]), as well as Minkowski sums in three-dimensions (e.g., see a survey paper [1]). Here, however, we focus on sums of planar polygonal regions.

We devised and implemented three algorithms for computing the Minkowski sum of two polygonal sets based on the CGAL software library [10,34]. Our main goal was to produce a *robust* and exact implementation. This goal was achieved by employing the *planar maps* [13] and *arrangements* [16] packages of CGAL while using exact number types. We use rational numbers and filtered geometric predicates from LEDA – the Library of Efficient Data-structures and Algorithms [28,29].

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 Fig. 4 for an example. This is in contrast with most existing motion planning software for which tight or narrow passages constitute a significant hurdle. More applications of our package are described in [11].

The robustness and exactness of our implementation come at a cost: they slow down the running time of the algorithms in comparison with a more standard implementation that uses floating point arithmetic. This makes it especially necessary to expedite the algorithms in other ways. All our algorithms begin with decomposing the input polygons into convex subpolygons. We discovered that not only the number



	P's decomposition		
	naïve triang.	min $\sum d_i^2$ triang.	min convex
$\sum d_i^2$	754	530	192
# of convex subpolygons in *P*	33	33	6
time (mSec) to compute $P \oplus Q$	2133	1603	120

(e)

Fig. 5. Different decomposition methods applied to the polygon P presented in (a): (b) naïve triangulation, (c) minimum $\sum d_i^2$ triangulation, and (d) minimum convex decomposition (the details are given in Section 3). Table (e) illustrates, for each decomposition, the sum of squares of degrees, the number of convex subpolygons, and the time in milliseconds to compute the Minkowski sum of P and a convex polygon, Q, with 4 vertices.

of subpolygons in the decomposition of the input polygons but also their shapes had dramatic effect on the running time of the Minkowski-sum algorithms; see Fig. 5 for an example.

In the theoretical study of Minkowski-sum computation (e.g. [20]), the choice of decomposition is often irrelevant (as long as we decompose the polygons into convex subpolygons) because it does not affect the worst-case *asymptotic* running time of the algorithms. In practice, however, different decompositions can induce a large difference in running time of the Minkowski-sum algorithms. The decomposition can affect the running time of algorithms for computing Minkowski sums in several ways: some of them are global to all algorithms that decompose the input polygons into convex polygons, while some others are specific to certain algorithms or even to specific implementations. The heart of this paper is an examination of these various factors and a report on our findings.

Polygon decomposition has been extensively studied in computational geometry; it is beyond the scope of this paper to give a survey of results in this area, and we refer the reader to the survey papers by Keil [24] and Bern [4], and the references therein. As we proceed, we will provide details on specific decomposition methods that we will be using.

We apply several optimization criteria to the decompositions that we employ. In the context of Minkowski sums, it is natural to look for decompositions that minimize the number of convex subpolygons. As we show in the sequel, we are also interested in decompositions with minimal maximum vertex degree of the decomposition graph, as well as several other criteria.

We report on our experiments with various decompositions and different input polygons. Among our findings are that in general: (i) triangulations are too costly, (ii) what constitutes a good decomposition for one of the input polygons depends on the other input polygon – consequently, we develop a procedure for simultaneously decomposing the two polygons such that a “mixed” objective function is minimized, (iii) there are optimal decomposition algorithms that significantly expedite the Minkowski-sum computation, but the decomposition itself is expensive to compute – in such cases simple heuristics that approximate the optimal decomposition perform very well.

In the next section we survey the Minkowski sum algorithms that we have implemented. In Section 3 we describe the different decomposition algorithms that we have implemented. We present a first set of experimental results in Section 4 and filter out the methods that turn out to be inefficient. In Section 5 we focus on the decomposition schemes that are not only fast to compute but also help compute the Minkowski sum efficiently. We give concluding remarks and propose directions for further work in Section 6.

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 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 [1,14,33] for information on arrangements and vertical decomposition, and to [13,16] for a detailed description of the planar maps and arrangements packages in CGAL on which our algorithms are based.

The input to our algorithms are two polygonal sets P and Q, with m and n vertices, respectively. Our algorithms consist of the following three steps.

Step 1. Decompose P into the convex subpolygons $P_1, P_2, \ldots, P_s$ and Q into the 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 [25]. 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 two different algorithms for performing Step 3, computing the union of the polygons in R, which we refer to as the *arrangement* algorithm and the *incremental union* algorithm. A detailed description of these algorithms is given in [11].

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 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 efficiently 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 $O(I + k \log k)$ [31]. 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 union of the polygons added so far. 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 CGAL planar map package. We can only give a naïve bound $O(k^2 \log^2 k)$ 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 on most problem instances.

Remarks. 1. We also implemented a union algorithm using a divide-and-conquer approach but since it mostly behaves worse than the incremental algorithm we do not describe it here. The full details are given in [11].

2. Our planar map package provides full support for maintaining the vertical decomposition, and for efficient point location in a map. However, using simple point-location strategies (naïve, walk-along-a-line) is often faster in practice [13]. Therefore we ran the tests reported below without maintaining the vertical decomposition.

3. The decomposition algorithms

We describe here the algorithms that we have implemented for decomposing the input polygons into convex subpolygons. We use decompositions both with or without Steiner points. Some of the techniques are optimal and some use heuristics to optimize certain objective functions. The running time of the decomposition stage is significant only when we search for the optimal solution and use dynamic programming; in all other cases the running time of this stage is negligible even when we implemented a naïve solution. Therefore we only mention the running time for the "heavy" decomposition algorithms.

We use the notation from Section 2. For simplicity of the exposition we assume here that the input data for the Minkowski algorithm are two *simple polygons* P and Q. In practice we use the same decomposition schemes that are presented here for general polygonal sets, mostly without changing them at all. However this is not always possible. For example, Keil's optimal minimum convex decomposition algorithm does not work on polygons with holes [4]. Furthermore, the problem of decomposing a polygon with holes into convex subpolygons is proven to be NP-hard irrespective of whether Steiner points are allowed; see [22]. Other algorithms that we use (e.g., AB algorithm) can be applied to general polygons without changes. We discuss these decomposition algorithms in the following sections.

In what follows P is a polygon with n vertices $p_1, \ldots, p_n$, r of which are reflex.

3.1. Triangulation

Naïve triangulation. This procedure searches for a pair of vertices p_i, p_j such that the segment $p_i p_j$ is a diagonal, namely it lies inside the polygon. It adds such a diagonal, splits the polygon into two subpolygons by this diagonal, and triangulates each subpolygon recursively. The procedure stops when the polygon becomes a triangle. See Fig. 5 for an illustration.

[4] In such cases we can apply a first decomposition step that connects the holes to the outer boundary and then use the algorithm on the simple subpolygons. This is a practical heuristic that does not guarantee an optimal solution.

In some of the following decompositions we are concerned with the degrees of vertices in the decomposition (namely the number of diagonals incident to a vertex). Our motivation for considering the degree comes from an observation on the way our planar map structures perform in practice: we noted that the existence of high degree vertices makes maintaining the maps slower. The DCEL structure that is used for maintaining the planar map has a pointer from each vertex of the map to one of its incident halfedges. We can traverse the halfedges around a vertex by using the adjacency pointers of the halfedges. If a vertex v_i has d incident halfedges, then finding the location of a new edge around v_i will take $\mathrm{O}(d)$ traversal steps. To avoid the overhead of a search structure for each vertex, the planar-maps implementation does not include such a structure. Therefore, since we build the planar map incrementally, if the degree of v_i in the final map is d_i then we performed $\sum_1^{d_i} \mathrm{O}(i) = \mathrm{O}(d_i^2)$ traversal steps on this vertex. Trying to minimize this time over all the vertices we can either try to minimize the maximum degree or the sum of squares of degrees, $\sum d_i^2$. Now, high degree vertices in the decomposition result in high degree vertices in the underlying arrangement, and therefore we try to avoid them. We can apply the same minimization criteria to the vertices of the decomposition.

Optimal triangulation – minimizing the maximum degree. Using dynamic programming we compute a triangulation of the polygon where the maximum degree of a vertex $\mathrm{MAX}(d_i)$ is minimal. The algorithm is described in [17], and runs in $\mathrm{O}(n^3)$ time.

Optimal triangulation – minimizing $\sum d_i^2$. We adapted the minimal-maximum-degree algorithm to find the triangulation with minimum $\sum d_i^2$ where d_i is the degree of vertex v_i of the polygon. (See Fig. 5 for an illustration.) The adaptation is straightforward. Since both $\sum d_i^2$ and $\mathrm{MAX}(d_i)$ are global properties of the decomposition that can be updated in constant time at each step of the dynamic programming algorithm – most of the algorithm and the entire analysis remain the same.

3.2. Convex decomposition without Steiner points

Greedy convex decomposition. The same as the naïve triangulation algorithm except that it stops as soon as the polygon does not have a reflex vertex.

Minimum number of convex subpolygons (min-convex). We apply the algorithm of Keil [21], which computes a decomposition of a polygon into the minimum number of convex subpolygons without introducing new vertices (Steiner points). The running time of the algorithm is $\mathrm{O}(r^2 n \log n)$. This algorithm uses dynamic programming. See Fig. 5. This result was recently improved to $\mathrm{O}(n + r^2 \min\{r^2, n\})$ [23].

Minimum $\sum d_i^2$ convex decomposition. We modified Keil's algorithm so that it will compute decompositions that minimize $\sum d_i^2$, the sum of squares of vertex degree. Like the modification of the min-max degree triangulation, in this case we also modify the dynamic programming scheme by simply replacing the cost function of the decomposition. Instead of computing the number of polygons (as the original min-convex decomposition algorithm does) we compute a different global property, namely the sum of squares of degrees. We can compute $\sum d_i^2$ in constant time given the values $\sum d_i^2$ of the decompositions of two subpolygons.

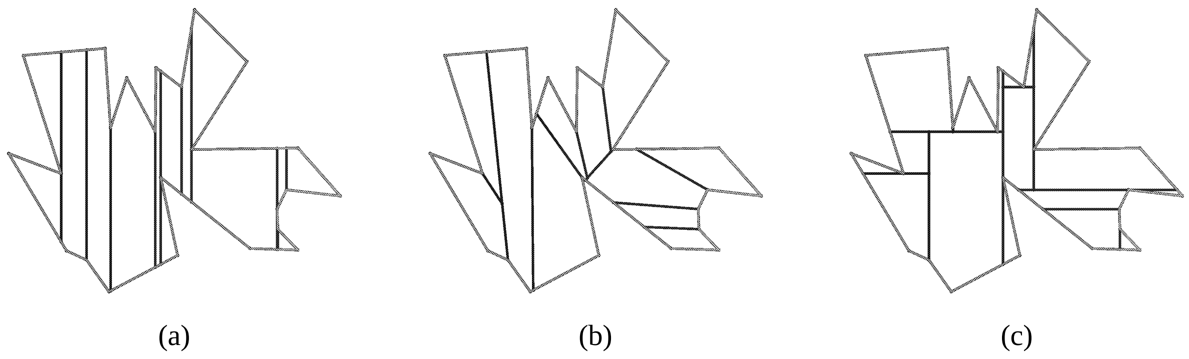


Fig. 6. (a) Slab decomposition, (b) angle “bisector” (AB) decomposition and (c) KD decomposition.

3.3. *Convex decomposition with Steiner points*

Slab decomposition. Given a direction $\vec{e}$, we extend a segment in directions $\vec{e}$ and $-\vec{e}$ from each reflex vertex of the polygon until it hits the polygon boundary. The result is a decomposition of the polygon into convex slabs. If $\vec{e}$ is vertical then this is the well-known vertical decomposition of the polygon. See Fig. 6. This decomposition gives a 4-approximation to the optimal convex decomposition as it partitions the polygon into at most $2r$ subpolygons and one needs at least $\lceil r/2 \rceil + 1$ subpolygons. The obvious advantage of this decomposition is its simplicity.

Angle “bisector” decomposition (AB). In this algorithm we extend the internal angle “bisector” from each reflex vertex until we first hit the polygon’s boundary or a diagonal that we have already extended from another vertex [5]. See Fig. 6. This decomposition (suggested by Chazelle and Dobkin [5]) gives a 2-approximation to the optimal convex decomposition: If P has r reflex vertices then every decomposition of P must include at least $\lceil r/2 \rceil + 1$ subpolygons, since every reflex vertex should be eliminated by at least one diagonal incident to it and each diagonal can eliminate at most 2 reflex vertices. The AB decomposition method extends one diagonal from each reflex vertex until P is decomposed into at most $r + 1$ convex subpolygons.

KD decomposition. This algorithm is inspired by the KD-tree method to partition a set of points in the plane [7]. First we divide the polygon by extending vertical rays inside the polygon from a reflex vertex horizontally in the middle (the number of vertices to the left of a vertex v, namely having smaller x-coordinate than v’s, is denoted v_l and the number of vertices to the right of v is denoted v_r. We look for a reflex vertex v for which $\max\{v_l, v_r\}$ is minimal). Then we divide each of the subpolygons by extending an horizontal line from a vertex vertically in the middle. We continue dividing the subpolygons that way (alternating between horizontal and vertical division) until no reflex vertices remain. See Fig. 6. By this method we try to lower the *stabbing number* of the subdivision (namely, the maximum number

[5] It is not necessary to compute exactly the direction of the angle bisector, it suffice to find a segment that will eliminate the reflex vertex from which it is extended. Let v be a reflex vertex and let u (w) be the previous (respectively next) vertex on the boundary of the polygon, then a segment at the direction $\overrightarrow{uv} + \overrightarrow{wv}$ divides the angle $\angle uvw$ into two angles with less than 180° each.

of subpolygons in the subdivision intersected by any line) – see the discussion in Section 5.2. The decomposition is similar to the quad-tree based approximation algorithms for computing the minimum-length Steiner triangulations [9].

4. A first round of experiments

We present experimental results of applying the decompositions described in the previous section to a collection of input pairs of polygons. We summarize the results and draw conclusions that lead us to focus on a smaller set of decomposition methods (which we study further in the next section).

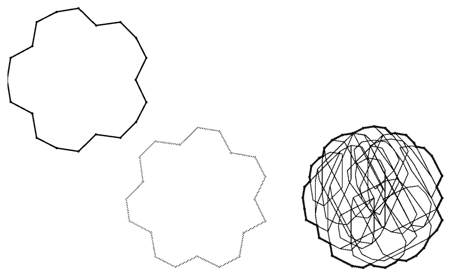
4.1. Test platform and frame program

Our implementation of the Minkowski sum package is based on the CGAL (version 2.0) and LEDA (version 4.0) 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 500 MHz Pentium III machine with 128 Mb of RAM.

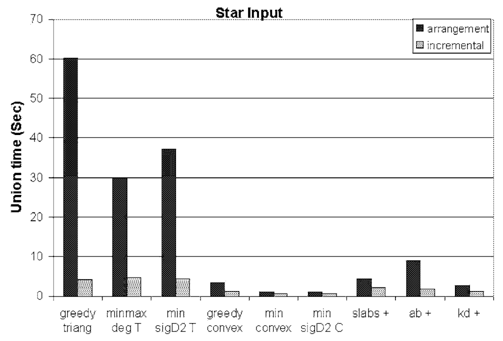
We implemented an interactive program that constructs Minkowski sums, computes configuration space obstacles, and solves polygon containment and polygon separation problems. The software lets the user choose the decomposition method and the union algorithm. It then presents the resulting Minkowski sum and underlying arrangement. The software is available from http://www.cs.tau.ac.il/~flato/.

4.2. Results

We ran the union algorithms (arrangement and incremental-union) with all nine decomposition methods on various input sets. The running times for the computation of the Minkowski sum for four input examples are summarized in Figs. 7–10.

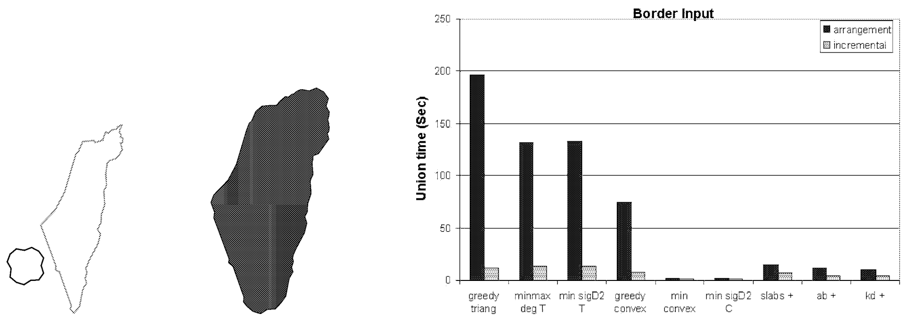


(a)



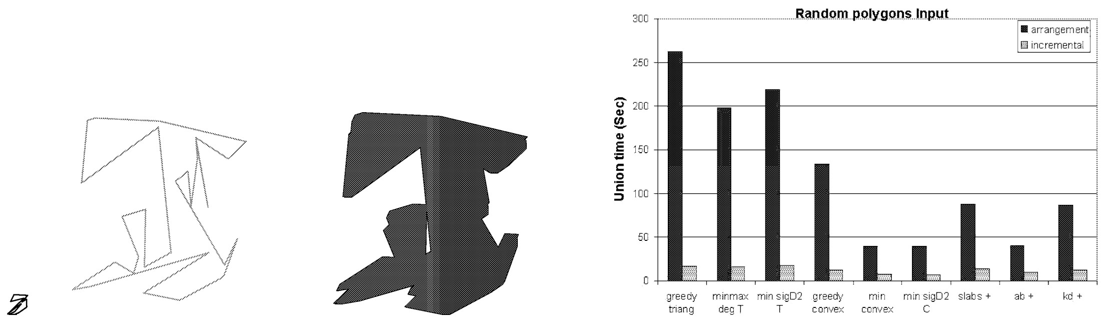
(b)

Fig. 7. Star input. The input (a) consists of two star-shaped polygons. The underlying arrangement of the Minkowski sum is shown in the middle. Running times in seconds for different decomposition methods (for two star polygons with 20 vertices each) are presented in (b).



(a) (b) (c)

Fig. 8. Border input. The input (an example in (a)) consists of a border of a country and a star shaped polygon. The Minkowski sum is shown in (b), and running times in seconds for different decomposition methods (for the border of Israel with 50 vertices and a star shaped polygon with 15 vertices) are shown in (c).



(a) (b) (c)

Fig. 9. Random polygons input. The input (an example in (a)) consists of two random looking polygons. The Minkowski sum is shown in (b), and running times in seconds for different decomposition methods (for two random looking polygons with 30 vertices each) are shown in (c).

It is obvious from the experimental results that using triangulations causes the union algorithms to run much slower (the left three pairs of columns in the histograms of Figs. 7–10). By triangulating the polygons, we create $(n-1)(m-1)$ hexagons in R with potentially $\Omega(m^2n^2)$ intersections between the edges of these polygons. We get those poor results since the performance of the union algorithms strongly depends on the number of vertices in the arrangement of the hexagon edges. Minimizing the maximum degree or the sum of squares of degrees in a triangulation is a slow computation that results in better union performance (compared to the naïve triangulation) but is still much worse than other simple convex-decomposition techniques.

In most cases the arrangement algorithm runs much slower than the incremental union algorithm. By removing redundant edges from the partial sum during the insertion of polygons, we reduce the number of intersections of new polygons and the current planar map features. The fork input is an exception since the complexity of the union is roughly the same as the complexity of the underlying arrangement and the edges that we remove in the incremental algorithm do not significantly reduce the complexity of the

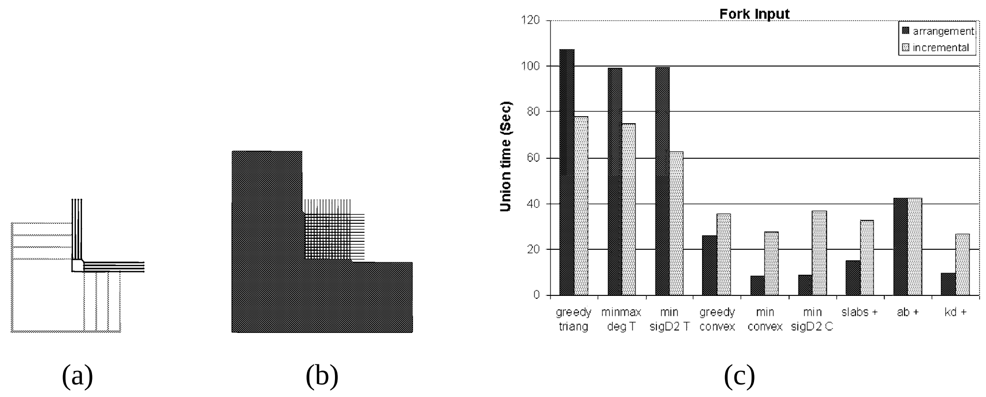


Fig. 10. Fork input. The input consists of two orthogonal fork polygons (a). The Minkowski sum is shown in (b), and running times in seconds for different decomposition methods (for two fork polygons with 8 teeth each) are shown in (c).

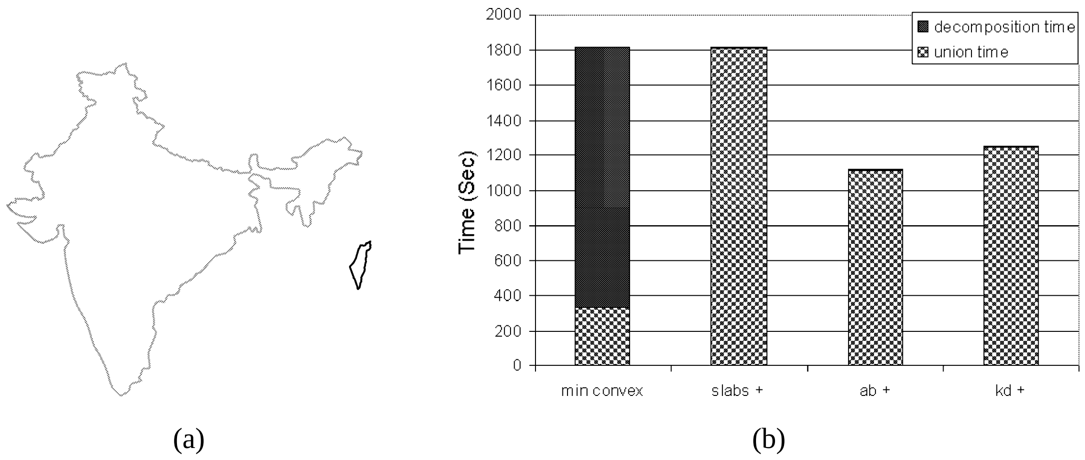


Fig. 11. An example of a case where when using the min-convex decomposition the union computation time is the smallest but it becomes inefficient when considering the decomposition time as well. Graph (b) shows the running times in seconds for computing the Minkowski sum of two polygons (a) representing the borders of India and Israel with 478 and 50 vertices, respectively. Note that while constructing the Minkowski sum of these two polygons the incremental union algorithm handles more than 40000 possibly intersecting segments.

planar map; see Fig. 10. More details on the comparison between the arrangement union algorithm and the incremental union algorithm are given in [12].

Although the min-convex algorithm is almost always the fastest in computing the union, constructing this optimal decomposition may be expensive. For some inputs running with the min-convex decomposition becomes inefficient – see, for example, Fig. 11. Minimizing the sum of squares of degrees in a convex decomposition rarely results in a decomposition that is different from the min-convex decomposition.

This first round of experiments helped us to filter out inefficient methods. In the next section we focus on the better decomposition algorithms, i.e., minimum convex, slab, angle “bisector”, KD. We further study them and attempt to improve their performance.

5. Revisiting the more efficient algorithms

In this section we focus our attention on the algorithms that were found to be efficient in the first round of experiments. As already mentioned, we measure efficiency by combining the running times of the decomposition step together with that of the union step. We present an experiment showing that minimizing the number of convex subpolygons in the decomposition does not always lead to better Minkowski-sum computation time; this is in contrast with the impression that the first round of results may give.

We also show in this section that in certain instances the decision how to decompose the input polygon P may change depending on the other polygon Q, namely for the same P and different Q's we should decompose P differently based on properties of the other polygon. This leads us to propose a "mixed" objective function for the simultaneous optimal decomposition of the two input polygons. We present an optimization procedure for this mixed function. Finally, we take the two most effective decomposition algorithms (AB and KD) – not only are they efficient, they are also very simple and therefore easy to modify – and we try to improve them by adding various heuristics.

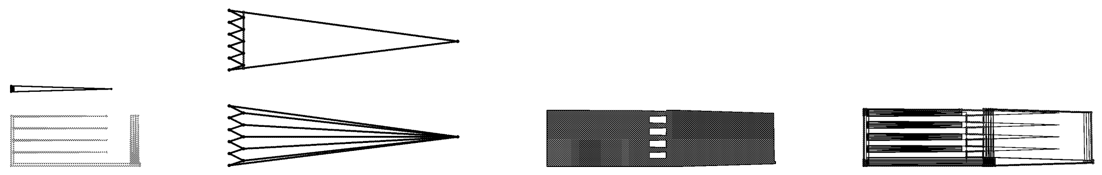
5.1. Nonoptimality of min-convex decompositions

Minimizing the number of convex parts of P and Q can be not only expensive to compute, but it does not always yield the best running time of the Minkowski-sum construction. In some cases other factors are important as well. Consider, for example, the knife input data. P is a long triangle with j teeth along its base and Q is composed of horizontal and vertical teeth. See Fig. 12. P can be decomposed into $j+1$ convex parts by extending diagonals from the teeth in the base to the apex of the polygon. Alternatively, we can decompose it into $j+2$ convex subpolygons with short diagonals (this is the "minimal length AB" decomposition described in Section 5.3). If we fix the decomposition of Q, the latter decomposition of P results in considerably faster Minkowski-sum running time, despite having more subpolygons, because the Minkowski sum of the long subpolygons in the first decomposition with the subpolygons of Q results in many intersections between the edges of polygons in R. In the first decomposition we have $j+1$ long subpolygons while in the latter we have $j+2$ subpolygons when only one of them is a "long" subpolygon and the rest are $j+1$ small subpolygons.

We can also see a similar behavior in real-life data. Computing the Minkowski sum of the (polygonal representation of) countries with star polygons mostly worked faster while using the KD-decomposition than with the AB technique; with the exception of degenerate polygons (i.e., with some reflex vertices that share the same x or y coordinates), the KD decomposition always generates at least as many subpolygons as the AB decomposition.

5.2. Mixed objective functions

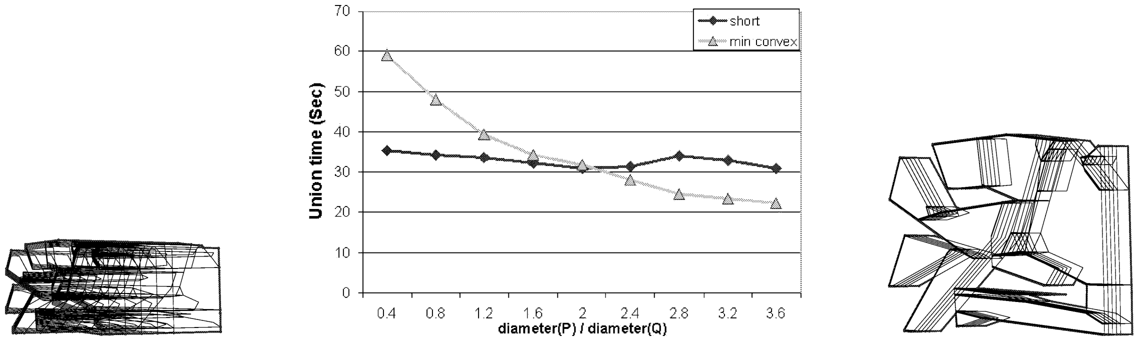
Good decomposition techniques that handle P and Q separately might not be sufficient because what constitutes a good decomposition of P depends on Q. We measured the running time for computing the Minkowski sum of a knife polygon P (Fig. 12 – the knife polygon is in (b)) and a random polygon Q (Fig. 9). We scaled Q differently in each test. We fixed the decomposition of Q and decomposed the knife polygon P once with the short $j+2$ "minimal length AB" decomposition and then with the long $j+1$ minimum convex decomposition. The results are presented in Fig. 13. We can see that for small



	$j+1$ **long decomposition**	$j+2$ **short decomposition**
number of vertices	23448	9379
running time (sec)	71.7	25.6

(e)

Fig. 12. Knife input. (a) The input polygons. (b) Two types of decompositions of P (enlarged): on top, $j+2$ subpolygons with short diagonals length, and below minimum convex decomposition with $j+1$ subpolygons with long diagonals. (c) The Minkowski sum of P and Q. (d) The underlying arrangement (using the short decomposition of P). (e) The table presents the number of vertices in the underlying arrangement and the running time for both decompositions (P has 20 teeth and 42 vertices and Q has 34 vertices).



(a) (b) (c)

Fig. 13. Minkowski sum of a knife, P, with 22 vertices and a random polygon, Q, with 40 vertices using the arrangement union algorithm. (a) The underlying arrangement of the sum with the smallest random polygon. (c) The underlying arrangement of the sum with the largest random polygon. As Q grows, the number of vertices I in the underlying arrangement is dropping from (about) 15000 to 5000 for the "long" decomposition of P, and from 10000 to 8000 for the "short" decomposition.

Q's the short decomposition of the knife P with more subpolygons performs better, but as Q grows the long decomposition of P with fewer subpolygons wins.

These experiments imply that a more careful strategy would be to simultaneously decompose the two input polygons, or at least take into consideration properties of one polygon when decomposing the other.

The running time of the arrangement union algorithm is $O(I + k\log k)$, where k is the number of edges of the polygons in R and I is the overall number of intersections between (edges of) polygons in R

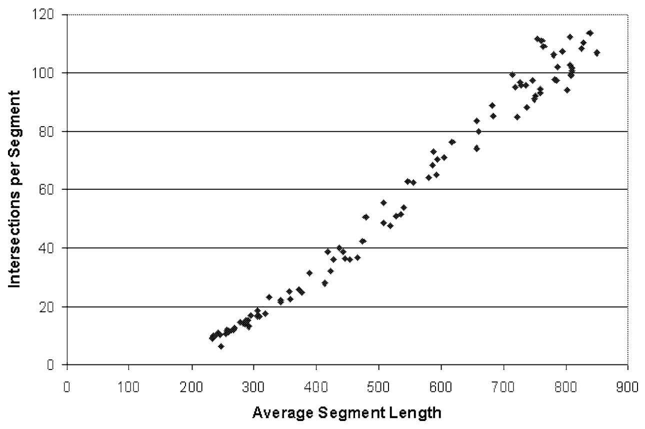


Fig. 14. Average number of intersections per segment as a function of the average segment length. Each point in the graph represents a configuration containing 125 randomly chosen points in a square $[0, 1000] \times [0, 1000]$ in $\mathbb{R}^2$ and 500 randomly chosen segments connecting pairs of these points.

(see Section 2). The value of k depends on the complexity of the convex decompositions of P and Q. Hence, we want to keep this complexity small. It is harder to optimize the value of I. Intuitively, we want each edge of R to intersect as few polygons of R as possible. If we consider the standard rigid-motion invariant measure μ on lines in the plane [32] and use $L(C)$ to denote the set of lines intersecting a set C, then for any polygon R_{ij}, $\mu(L(R_{ij}))$ is the perimeter of R_{ij}. This suggests that we want to minimize the total lengths of the diagonals in the convex decompositions of P and Q (Aronov and Fortune [2] use this approach to show that minimizing the length of a triangulation can decrease the complexity of the average-case ray-shooting query). But we want to minimize the two criteria simultaneously, and let the decomposition of one polygon govern the decomposition of the other.

We can see supporting experimental results for segments in Fig. 14. In these experiments we randomly chose a set T of points inside a square in $\mathbb{R}^2$ and connected pairs of them by a set S of random segments (for each segment we randomly chose its two endpoints from T). Then we measured the average number of intersections per segment as a function of the average length of a segment. To get different average length of the segments, at each round we chose each segment by taking the longest (or shortest) segment out of l randomly chosen segments, where l is a small integer varying between 1 and 15. The average number of intersections is $I/|S|$ where I is the total number of intersections in the arrangement $\mathcal{A}(S)$. We performed 5 experiments for each value of l between 1 and 15. Each plotted point in the graph in Fig. 14 represents such an experiment. The values of l are not shown in the graph – they were used to generate sets of segments with different average lengths. For the presented results, we took $|S| = 4|T|$ (this is a typical ratio between points and segments in the set R for which we compute the arrangement $\mathcal{A}(R)$). As the results show, the intersection count per segment grows linearly (or close to linearly) with the average length of a segment.

Therefore, we assume that the expected number of intersection of a segment in the arrangement $\mathcal{A}(R)$ of the polygons of R is proportional to the total length of edges of $\mathcal{A}(R)$, which we denote by $\pi_{\mathcal{A}(R)}$. The intuition behind the mixed objective function, which we propose next, is that minimizing $\pi_{\mathcal{A}(R)}$ will lead to minimizing I.

Let $P_1, P_2, \ldots, P_{k_P}$ be the convex subpolygons into which P is decomposed. Let π_{P_i} be the perimeter of P_i. Similarly define $Q_1, Q_2, \ldots, Q_{k_Q}$ and π_{Q_j}. If $\pi_{R_{ij}}$ is the perimeter of R_{ij} (the Minkowski sum of P_i and Q_j), then

$$\pi_{R_{ij}} = \pi_{P_i} + \pi_{Q_j}.$$

Summing over all (i, j) we get

$$\pi_{\mathcal{A}(R)} = \sum_{ij} \pi_{R_{ij}} = \sum_{ij} (\pi_{P_i} + \pi_{Q_j}) = k_Q \Big(\sum_i \pi_{P_i} \Big) + k_P \Big(\sum_j \pi_{Q_j} \Big).$$

Let π_P denote the perimeter of P and Δ_P the sum of the lengths of the diagonals in P. Similarly define π_Q and Δ_Q. P has k_P subpolygons and Q has k_Q subpolygons. Let $D_{P,Q}$ be the decomposition of P and Q. Then

$$c(D_{P,Q}) = \pi_{\mathcal{A}(R)} = k_Q(2\Delta_P + \pi_P) + k_P(2\Delta_Q + \pi_Q).$$

The function $c(D_{P,Q})$ is a cost function of a simultaneous convex decomposition of P and Q. Our empirical results showed that this cost function approximates the running time. We want to find a decomposition that minimizes this cost function. Let $c^* = \min_{D_{P,Q}} c(D_{P,Q})$.

If we do not allow Steiner points, we can modify the dynamic-programming algorithm by Keil [21] to compute c^* in $\mathrm{O}(n^2 r_P^4 + m^2 r_Q^4)$ as follows. We define an auxiliary cost function $\widehat{c}(P, i)$, which is the minimum total length of diagonals in a convex decomposition of P into at most i convex polygons. Then

$$c^* = \min_{i,j} \big[j\big(2\widehat{c}(P, i) + \pi_P\big) + i\big(2\widehat{c}(Q, j) + \pi_Q\big)\big].$$

Since the number of convex subpolygons in any minimal convex decomposition of a simple polygon is at most twice the number of the reflex vertices in it, the values i and j are at most $2r_P$ and $2r_Q$, respectively, where r_P (respectively r_Q) is the number of reflex vertices in P (respectively Q). One can compute $\widehat{c}(P, i)$ by modifying Keil's algorithm [21] – the modified algorithm as well as the algorithm for computing c^* are described in detail in Appendix A.

Since the running time of this procedure is too high to be practical, we neither implemented it, nor did we make any serious attempt to improve the running time. We regard this algorithm as a first step towards developing efficient algorithms for approximating mixed objective functions.

If we allow Steiner points, then it is an open question whether an optimal decomposition can be computed in polynomial time. Currently, we do not even have a constant-factor approximation algorithm. The difficulty arises because no constant-factor approximation is known for minimum-length convex decomposition of a simple polygon if Steiner points are allowed [22].

5.3. *Improving the AB and KD methods*

It seems from most of the tests that in general the AB and KD decomposition algorithms work better than the other heuristics. We next describe our attempts to improve these algorithms.

Minimal length angle "bisector" decomposition. In each step we handle one reflex vertex. A reflex vertex can always be eliminated by at most two diagonals. For any three diagonals that eliminate a reflex vertex, at least one of them can be removed while the vertex is still eliminated. In this algorithm, for each reflex vertex we look for the shortest one or two diagonals that eliminate it. As we can see in Fig. 16, the *minimal length AB* decomposition performs better than the naïve AB even though it generally creates more subpolygons.

While the AB decomposition performs very well, in some cases (concave chains, countries borders) the KD algorithm performs better. We developed the KD-decomposition technique aiming to minimize the stabbing number of the decomposition of the input polygons (which in turn, as discussed above, we expect to reduce the overall number I of intersections in the underlying arrangement $\mathcal{A}(R)$ of the polygons of R). This method however often generates too many convex parts. We tried to combine these two algorithms as follows.

Angle "bisector" and KD decomposition (AB + KD). In this algorithm we check the two neighbors vertices v_1, v_2 of each reflex vertex v; if v_1 and v_2 are convex, we extend a "bisector" from v. We apply the KD decomposition algorithm for the remaining non-convex polygons. By this method we aim to lower the stabbing number without creating redundant convex polygons in the sections of the polygons that are not bounded by concave chains). We tested these algorithms on polygons with different number of convex vertices, vertices in concave chains and "tooth vertices". The results in Fig. 15 suggest that AB + KD performs best when the numbers of vertices in concave chains and of tooth vertices are roughly the same. If there are more tooth vertices than the vertices in concave chains, then the AB decomposition performs better.

Next, we tried to further decrease the number of convex subpolygons generated by the decomposition algorithm. Instead of emanating a diagonal from any reflex vertex, we first tested whether we can eliminate two reflex vertices with one diagonal (let us call such a diagonal a *2-reflex eliminator*). All the methods listed below generate at most the same number of subpolygons generated by the AB algorithm but practically the number is likely to be smaller.

Improved angle "bisector" decomposition. For a reflex vertex, we look for 2-reflex eliminators. If we cannot find such a diagonal, we continue as in the standard AB algorithm.

Reflex angle "bisector" decomposition. In this method we work harder trying to find 2-reflex eliminator diagonals. In each step we go over all reflex vertices trying to find an eliminator. If there are no more 2-reflex eliminators, we continue with the standard AB algorithm on the rest of the reflex vertices.

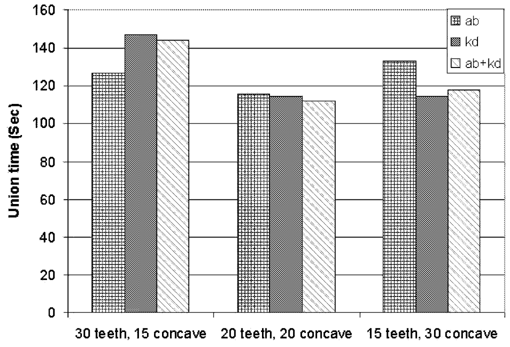
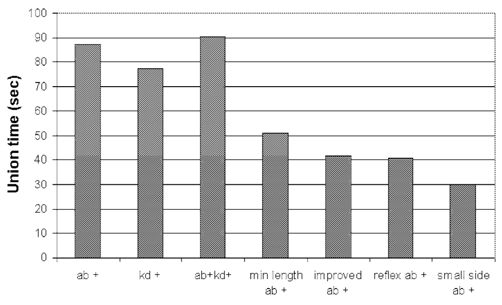
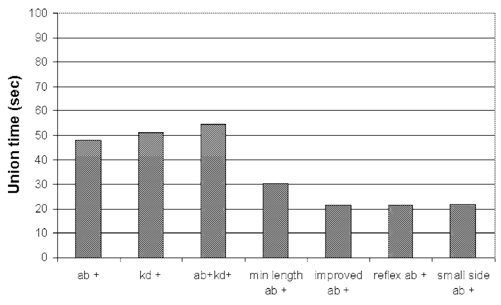


Fig. 15. Running times for computing the chain input using AB, KD and AB + KD decompositions.



(a)



(b)

Fig. 16. Union running times for countries borders input ((a) Minkowski sum of Chile with 368 vertices and a star polygon with 23 vertices, and (b) Minkowski sum of Norway with 360 vertices and a star polygon with 15 vertices) with the improved decomposition algorithms.

Small side angle "bisector" decomposition. As in the *reflex AB* decomposition, we are looking for 2-reflex eliminators. Such an eliminator decomposes the polygon into two parts, one on each of its sides. Among the candidate eliminators we choose the one that has the minimal number of reflex vertices on one of its sides. Vertices on different sides of the added diagonal cannot be connected by another diagonal because it will intersect the added diagonal. By choosing this diagonal we are trying to "block" the minimal number of reflex vertices from being connected (and eliminated) by another 2-reflex eliminator diagonal.

Experimental results are shown in Fig. 16. These latter improvements to the AB decomposition seem to have the largest effect on the union running time, while keeping the decomposition method very simple to understand and implement. Note that the *small side AB* heuristic results in 20% faster union time than the *improved AB* and *reflex AB* decompositions, and 50% faster than the standard *angle "bisector"* method. When we use the *small side AB* with the input set used in Fig. 11 the overall running time is about 376 seconds which is at least three times faster than the results achieved by using other decomposition methods.

6. Conclusions

We presented a general scheme for computing the Minkowski sum of polygons. We implemented union algorithms which overcome all possible degeneracies. Using exact number types and special handling for geometric degeneracies we obtained a robust and exact implementation that could handle all kinds of polygonal inputs. The emphasis of this paper is on the effect of the decomposition method on the efficiency of the overall process.

We implemented over a dozen of decomposition algorithms, among them triangulations, optimal decompositions for different criteria, approximations and heuristics. We examined several criteria that affect the running time of the Minkowski-sum algorithm. The most effective optimization is minimizing the number of convex subpolygons. Thus, triangulations which are widely used in the theoretical literature are not practical for the Minkowski-sum algorithms. We further found that minimizing the number of subpolygons is not always sufficient. Since we deal with two polygonal sets that are participating in the algorithm we found that it is smarter to decompose the polygons simultaneously minimizing a cost function which takes into account the decomposition of both input set. Optimal decompositions for this function and also simpler cost functions like the overall number of convex subpolygons were practically too slow. In some cases the decomposition step of the Minkowski-sum algorithm took more time than the union step. Therefore, we developed some heuristics that approximate very well a cost function and run much faster than their exact counterparts. Allowing Steiner points, the angle “bisector” decomposition gives a 2-approximation for the minimal number of convex subpolygons. The AB decomposition with simple practical modifications (small-side AB decomposition) is a decomposition that is easy to implement, very fast to execute and gives excellent results in the Minkowski-sum algorithm.

We propose several direction for further research:

1. Use the presented scheme and the practical improvement that we proposed with real-life applications such as motion planning and GIS and examine the effect of different decompositions for those special types of input data.
2. Further improve the AB decomposition algorithms to give better theoretical approximation and better running times.
3. We tested the efficiency of the Minkowski-sum algorithm with different convex decomposition methods, but the algorithm will still give a correct answer if we will have a covering of the input polygons by convex polygons. Can one further improve the efficiency of the Minkowski sum program using coverings instead of decompositions.

Appendix A. Polygons decomposition minimizing the mixed objective function

In Section 5.2 we developed a mixed objective function for the decomposition of the two input polygons to the Minkowski sum computation. In this appendix we describe an algorithm based on the optimal convex decomposition of Keil [21] for decomposing the input polygons simultaneously minimizing the mixed objective function. Here we do not allow Steiner points.

Following the notation of Section 5.2, we define an auxiliary cost function $\widehat{c}(P,a)$, which is the minimum total length of diagonals in a convex decomposition of P into at most a convex polygons. Then the cost of the decomposition is

$$c^* = \min_{a,b}\bigl[b\bigl(2\widehat{c}(P,a)+\pi_P\bigr)+a\bigl(2\widehat{c}(Q,b)+\pi_Q\bigr)\bigr].$$

Assuming that we know the decompositions of P and Q that achieve $\widehat{c}(P,a)$ and $\widehat{c}(Q,b)$, respectively, for every a and b, we can compute c^* and find the decomposition in $O(r_P r_Q)$ time (see Section 5.2). The single issue that we need to resolve is how to compute $\widehat{c}(P,a)$. In the following section we describe a dynamic-programing algorithm to compute the minimum length decomposition of a polygon (based on [21]) and in Section A.2 we describe how to modify this algorithm for computing $\widehat{c}(P,a)$.

A.1. Minimum-length decomposition

Let $v_1, v_2, \ldots, v_m$ be the vertices of P given in clockwise order. We call a pair (i, j) *valid* if v_i is visible from v_j and at least one of the two vertices is a reflex vertex. If two vertices are visible from

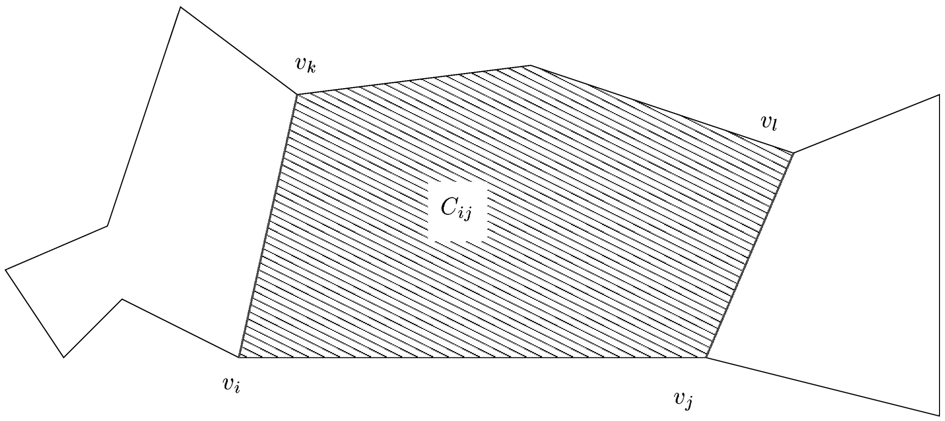


Fig. 17. The base subpolygon C_{ij} of P_{ij} with (i,k) as its first edge and (l,j) as its last edge.

each other and they are not a valid pair, then they must both be convex. A diagonal that connects two convex vertices is redundant in any optimal convex decomposition because it can be removed and the two convex subpolygons on its sides can be merged into a convex polygon. Therefore, for the construction of an optimal decomposition, we can consider only the diagonals that connect two vertices that are a valid pair. For a valid pair (i,j), let P_{ij} be the polygonal chain from vertex v_i to v_j. $P_{1n} = P$ is also a valid chain. Let $d(i,j)$ be the length of the diagonal (i,j).

Let $f(i,j)$ denote the cost of the minimum-length decomposition of P_{ij}; $f(i,j)$ only counts the length of diagonals added, and does not include the perimeter of P_{ij}. For a convex decomposition of P_{ij}, let C_{ij} be the *base* convex polygon that contains the edge (i,j). Let (i,k) and (l,j) be the first and the last edges of C_{ij}; see Fig. 17. The pair (i,k) should be a valid pair unless $k = i+1$. Similarly, the pair (l,j) should be a valid pair unless $l = j-1$.

We define a function $F(i,j;k,l)$ as follows: $F(i,j;k,l)$ is the cost of a minimum-length decomposition under the constraints that (i,k) is the first edge of C_{ij} and (l,j) is the last edge of C_{ij}. If $k \neq i+1$, then (i,k) has to be a valid pair, and a similar condition holds for the pair (l,j).

If the angle (j,i,k) or (l,j,i) is greater than 180°, we set $F(i,j;k,l)$ to infinity, as it is not a valid convex decomposition. Then

$$f(i,j) = \min_{k,l} F(i,j;k,l).$$

We need to compute $F(i,j;k,l)$ for at most $m^2 r_P^2$ pairs because if i is reflex (convex), then k is any (respectively reflex) vertex. The same condition holds for l and j.

We can compute the values of F using the following recursive formula:

$$F(i,j;k,l) = d(l,j) + f(l,j) + \min_{g} F(i,l;k,g), \tag{A.1}$$

where the minimum is taken over all vertices g such that (g,l) is a valid pair (or $g = l-1$) and the angle $(g,l,j) \leqslant 180°$. The recurrence uses a minimum decomposition of P_{lj} along with a minimum decomposition of P_{il} for which the first edge of C_{il} is (i,k) and the last edge is (g,l). A vertex g is chosen only if the polygon C_{il} can merge with the triangle T_{ilj}. Since both T_{ilj} and C_{il} are convex it is sufficient to verify that the angles (g,l,j), $(j,i,k) \leqslant 180°$. See Fig. 18 for an illustration.

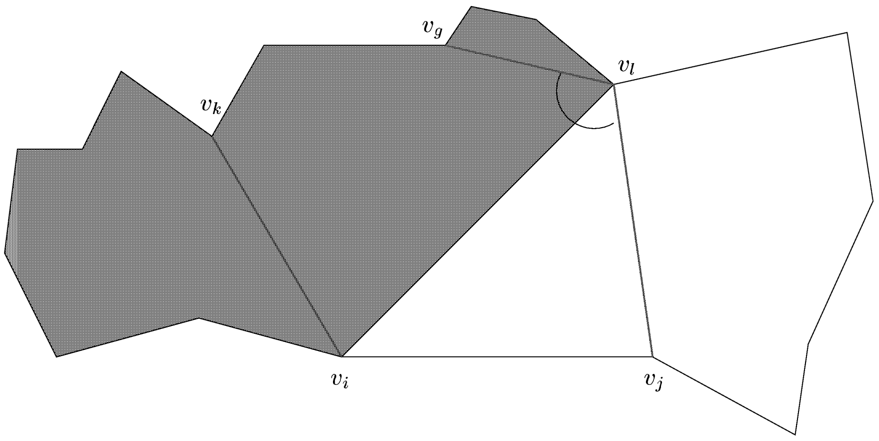


Fig. 18. The recurrence: we compute $F(i, j; k, l)$ using a minimum decomposition of P_{lj} along with a minimum decomposition of P_{il} (shaded) for which the first edge of C_{il} is (i, k) and the last edge is (g, l). The triangle T_{ilj} can merge with the base subpolygon C_{il}.

To complete the algorithm we first need to find all valid pairs and then compute $F(i, j; k, l)$ for them. The result of the algorithm will be $f(1, m)$. We will compute $F(i, j; k, l)$ in ascending order of the difference $j - i$ using Eq. (A.1).

Theorem A.1. *The minimum-length convex decomposition of P can be computed in $O(m^2 r_P^2)$ time.*

Proof. For each pair (i, j) we can compute whether v_i is visible from v_j in $O(m)$ time. A potentially valid pair is a pair (i, j) for which at least one of v_i or v_j is reflex. Computing visibility for all potentially valid pairs will therefore take $O(m^2 r_P)$. Sorting all valid pairs in ascending order of the difference between the indices will take an additional $O(m r_P \log m)$ time.

For a fixed quadruple i, j, k and l, let $g(i, j, k, l)$ denote the index of g that minimizes the recurrence (A.1). For a fixed triple i, k and l, $g(i, j, k, l)$ increases monotonically with j because as we increase j, the angle (j, l, i) can only decrease and more pairs (g, l) become relevant; see Fig. 19. While using the recurrence (A.1) we should only compute the minimum over all relevant g's that are greater than $g(i, j', k, l)$, where j' is the largest index for which (i, j') and (l, j') are valid pairs and $j' < j$. Thus, the amortized time spent in computing each $F(i, j; k, l)$ is $O(1)$. The overall running time of the algorithm is therefore $O(m^2 r_P^2)$. □

A.2. Constrained minimum-length decomposition

We slightly change the above algorithm to compute $\widehat{c}(P, a)$. We define $F(s, i, j; k, l)$ to be the minimum-length convex decomposition of P_{ij} into at most s convex subpolygons, under the constraint that (i, k) is the first edge of the base polygon C_{ij} and that (l, j) is the last edge of C_{ij}. If the angle (j, i, k) or (l, j, i) is greater than 180° or if P_{ij} cannot be decomposed into at most s convex subpolygons, we set the cost to infinity. We define $f(s, i, j)$ to be the cost of any convex decomposition of P_{ij} with at most s subpolygons.

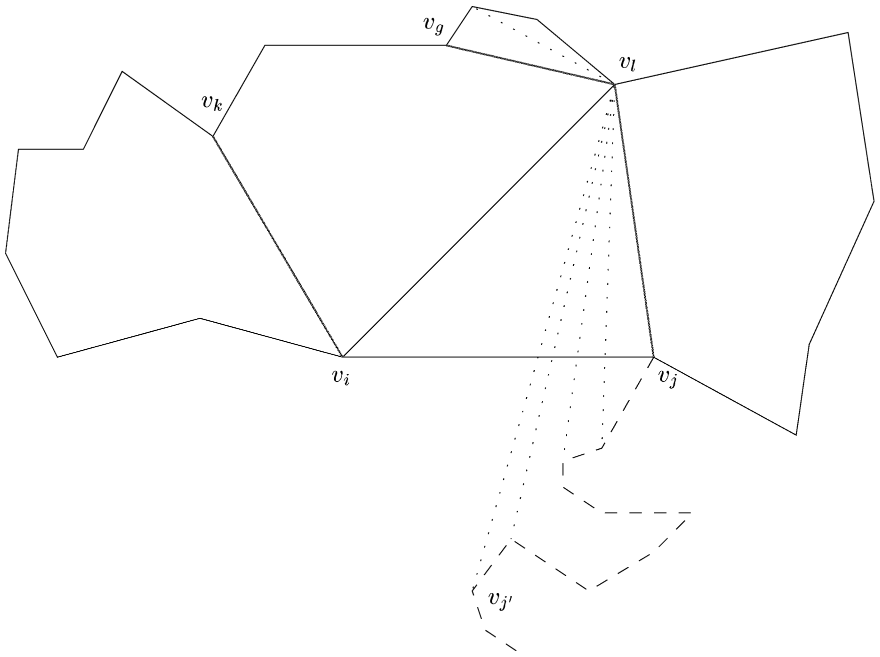


Fig. 19. When j increases, the angle (j, l, i) decreases and more valid pairs (g, l) become relevant.

The recurrence is now given by

$$F(s, i, j; k, l) = d(l, j) + \min_{u \leqslant s} \left\{ f(u, l, j) + \min_{g} F(s - u, i, l; k, g) \right\}, \tag{A.2}$$

where the minimum is taken over all vertices g such that (g, l) is a valid pair (or $g = l - 1$) and the angle $(g, l, j) \leqslant 180°$.

Theorem A.2. *The minimum-length convex decomposition of P into at most s subpolygons (for every $1 \leqslant s \leqslant 2r_p$) can be computed in $O(m^2 r_P^4)$ time.*

Proof. We use the arguments from the proof of Theorem A.1. We now compute $O(m^2 r_P^3)$ entries. The monotonicity condition described above still holds, i.e., for a fixed quadruple s, i, k, l, the value of g increases monotonically with j. So each entry can be computed in $O(r_P)$ amortized time since $s \leqslant 2r_P$, giving a total of $O(m^2 r_P^4)$ time. □

Theorem A.3. *A decomposition D_{PQ} of the polygons P and Q that minimizes the mixed function $c(D_{PQ})$ can be computed in time $O(m^2 r_P^4 + n^2 r_Q^4)$.*

Proof. Using the above algorithm we can compute $\widehat{c}(P, a) = f(a, 1, m)$ for P for every $a \leqslant 2r_P$ in $O(m^2 r_P^4)$ time and $\widehat{c}(Q, b) = f(b, 1, n)$ for Q for every $b \leqslant 2r_Q$ in $O(n^2 r_Q^4)$. We need an additional $O(r_P r_Q)$ time to compute c^*, which is subsumed by the other factors of the running time. □

References

[1] P.K. Agarwal, M. Sharir, Arrangements, in: J.-R. Sack, J. Urrutia (Eds.), Handbook of Computational Geometry, Elsevier Science/North-Holland, Amsterdam, 1999, pp. 49–119.

[2] B. Aronov, S. Fortune, Approximating minimum-weight triangulations in three dimensions, Discrete Comput. Geom. 21 (4) (1999) 527–549.

[3] C. Bajaj, M.-S. Kim, Generation of configuration space obstacles: the case of moving algebraic curves, Algorithmica 4 (1989) 155–172.

[4] M. Bern, Triangulations, in: J.E. Goodman, J. O'Rourke (Eds.), Handbook of Discrete and Computational Geometry, CRC Press, Boca Raton, FL, 1997, pp. 413–428.

[5] B. Chazelle, D.P. Dobkin, Optimal convex decompositions, in: G.T. Toussaint (Ed.), Computational Geometry, North-Holland, Amsterdam, 1985, pp. 63–133.

[6] K. Daniels, V. Milenkovic, Limited gaps, in: Proc. 6th Canad. Conf. Comput. Geom., 1994, pp. 225–230.

[7] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry: Algorithms and Applications, Springer, Berlin, 1997.

[8] G. Elber, M.-S. Kim, Offsets, Sweeps and Minkowski Sums, Special Issue of Comput. Aided Design 31 (1999).

[9] D. Eppstein, Approximating the minimum weight Steiner triangulation, Discrete Comput. Geom. 11 (1994) 163–191.

[10] A. Fabri, G. Giezeman, L. Kettner, S. Schirra, S. Schönherr, On the design of CGAL, the Computational Geometry Algorithms Library, Software – Practice Exper. 30 (2000) 1167–1202.

[11] E. Flato, Robust and efficient construction of planar Minkowski sums, Master's thesis, Department of Computer Science, Tel-Aviv University, http://www.cs.tau.ac.il/~flato, 2000.

[12] E. Flato, D. Halperin, Robust and efficient construction of planar Minkowski sums, in: Abstracts 16th European Workshop Comput. Geom., Eilat, 2000, pp. 85–88.

[13] E. Flato, D. Halperin, I. Hanniel, O. Nechushtan, The design and implementation of planar maps in CGAL, in: J. Vitter, C. Zaroliagis (Eds.), Proceedings of the 3rd Workshop on Algorithm Engineering, Lecture Notes in Computer Science, Vol. 1148, Springer, 1999, pp. 154–168. To appear in The ACM Journal of Experimental Algorithmics.

[14] D. Halperin, Arrangements, in: J.E. Goodman, J. O'Rourke (Eds.), Handbook of Discrete and Computational Geometry, CRC Press, Boca Raton, FL, 1997, pp. 389–412.

[15] D. Halperin, J.-C. Latombe, R.H. Wilson, A general framework for assembly planning: The motion space approach, Algorithmica 26 (2000) 577–601.

[16] I. Hanniel, The design and implementation of planar arrangements of curves in CGAL, Master's thesis, Department of Computer Science, Tel-Aviv University, 2000.

[17] G. Kant, H.L. Bodlaender, Triangulating planar graphs while minimizing the maximum degree, in: Proc. 3rd Scand. Workshop Algorithm Theory, Lecture Notes in Computer Science, Vol. 621, Springer, 1992, pp. 258–271.

[18] A. Kaul, R. Farouki, Computing Minkowski sums of plane curves, Internat. J. Comput. Geom. Appl. 5 (1995) 413–432.

[19] A. Kaul, M.A. O'Connor, V. Srinivasan, Computing Minkowski sums of regular polygons, in: Proc. 3rd Canad. Conf. Comput. Geom., 1991, pp. 74–77.

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

[21] J.M. Keil, Decomposing a polygon into simpler components, SIAM J. Comput. 14 (1985) 799–817.

[22] J.M. Keil, J.-R. Sack, Minimum decompositions of polygonal objects, in: G.T. Toussaint (Ed.), Computational Geometry, North-Holland, Amsterdam, 1985, pp. 197–216.

[23] J.M. Keil, J. Snoeyink, On the time bound for convex decomposition of simple polygons, in: Proc. 10th Canad. Conf. Comput. Geom., 1998.
[24] M. Keil, Polygon decomposition, in: J.-R. Sack, J. Urrutia (Eds.), Handbook of Computational Geometry, Elsevier Science/North-Holland, Amsterdam, 1999.
[25] J.-C. Latombe, Robot Motion Planning, Kluwer Academic Publishers, Boston, 1991.
[26] I.-K. Lee, M.-S. Kim, G. Elber, Polynomial/rational approximation of Minkowski sum boundary curves, Graphical Models Image Process. 60 (2) (1998) 136–165.
[27] D. Leven, M. Sharir, Planning a purely translational motion for a convex object in two-dimensional space using generalized Voronoi diagrams, Discrete Comput. Geom. 2 (1987) 9–31.
[28] K. Melhorn, S. Näher, The LEDA Platform of Combinatorial and Geometric Computing, Cambridge University Press, 1999.
[29] K. Mehlhorn, S. Näher, C. Uhrig, M. Seel, The LEDA User Manual, Version 4.0, Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany, http://www.mpi-sb.mpg.de/LEDA/leda.html, 1999.
[30] V. Milenkovic, K. Daniels, Z. Li, Placement and compaction of nonconvex polygons for clothing manufacture, in: Proc. 4th Canad. Conf. Comput. Geom., 1992, pp. 236–243.
[31] K. Mulmuley, Computational Geometry: An Introduction through Randomized Algorithms, Prentice-Hall, Englewood Cliffs, NJ, 1994.
[32] L. Santaló, Integral Probability and Geometric Probability, Encyclopedia of Mathematics and Its Applications, Vol. 1, Addison-Wesley, Reading, MA, 1979.
[33] M. Sharir, P.K. Agarwal, Davenport–Schinzel Sequences and Their Geometric Applications, Cambridge University Press, New York, 1995.
[34] The CGAL User Manual, Version 2.0, http://www.cgal.org, 1999.