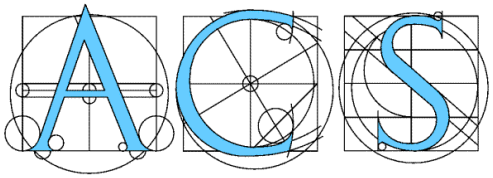


ACS

Algorithms for Complex Shapes
with Certified Numerics and Topology

Exact and Efficient Construction of Planar Minkowski Sums using the Convolution Method

Ron Wein

ACS Technical Report No.: ACS-TR-241300-04

Part of deliverable:	WP-I/D3
Site:	TAU
Month:	24

Project co-funded by the European Commission within FP6 (2002–2006)
under contract nr. IST-006413

Abstract

The Minkowski sum of two sets $A, B \in \mathbf{R}^d$, denoted $A \oplus B$, is defined as $\{a + b \mid a \in A, b \in B\}$. We describe an efficient and robust implementation for the construction of Minkowski sums of polygons in $\mathbf{R}^2$ using the *convolution* of the polygon boundaries. This method allows for faster computation of the sum of non-convex polygons in comparison to the widely-used methods for Minkowski-sum computation that decompose the input polygons into convex sub-polygons and compute the union of the pairwise sums of these convex sub-polygons.

1 Introduction

Given two sets $A, B \in \mathbb{R}^d$, their *Minkowski sum*, denoted by $A \oplus B$, is the set $\{a + b \mid a \in A, b \in B\}$. Minkowski sums are used in many applications, such as motion planning and computer-aided design and manufacturing. In this paper we focus on an important sub-class of the planar Minkowski-sum computation problem: computing the sum of two simple polygons.

If P and Q are simple planar polygons having m and n vertices respectively, then $P \oplus Q$ is a subset of the arrangement of $O(mn)$ line segments, where each segment is the Minkowski sum of an edge of P with a vertex of Q, or vice-versa. The size of the sum is therefore bounded by $O(m^2n^2)$, and this bound is tight [11]. However, if both P and Q are convex, then $P \oplus Q$ is a convex polygon with at most $m + n$ vertices, and can be computed in $O(m + n)$ time (see, e.g., [3, Chap. 13]). If only P is convex, the Minkowski sum of P and Q is bounded by $O(mn)$ [12], and this bound is tight as well.

As we mentioned above, computing the Minkowski sum of two convex polygons can be performed in linear time using a simple procedure that can be easily implemented in software. The prevailing method for computing the sum of two non-convex polygons P and Q, is therefore based on *convex decomposition*: we decompose P into convex sub-polygons $P_1, \ldots, P_k$ and Q into convex sub-polygons $Q_1, \ldots, Q_\ell$, obtain the Minkowski sum of each pair of sub-polygons and compute the union of the $k\ell$ pairwise sub-sums. Namely, we calculate $P \oplus Q = \bigcup_{i,j} (P_i \oplus Q_j)$. Flato [4] (see also [1]), implemented a software package for computing Minkowski sums of planar polygons in an exact manner, based on the decomposition method. He implemented about a dozen different polygon-decomposition strategies and several methods for the union computation, and conducted thorough experiments to determine the optimal decomposition and union strategies. Flato's code is robust and produces exact results. It is based on CGAL Version 2.0,[1] and uses exact rational arithmetic. This is the first implementation capable of handling degenerate inputs, and the only one that correctly identifies low-dimensional elements of the Minkowski sum, such as antennas or isolated vertices (see more details in Sec. 3.2). The LEDA library [13] also contains functions for robust Minkowski-sum computation based on convex polygon decomposition that use exact rational arithmetic.[2] However, these

[1] See CGAL's homepage at `http://www.cgal.org`.

[2] For more details and a detailed on-line documentation, see:
`http://www.algorithmic-solutions.info/leda_guide/geo_algs/minkowski.html`.

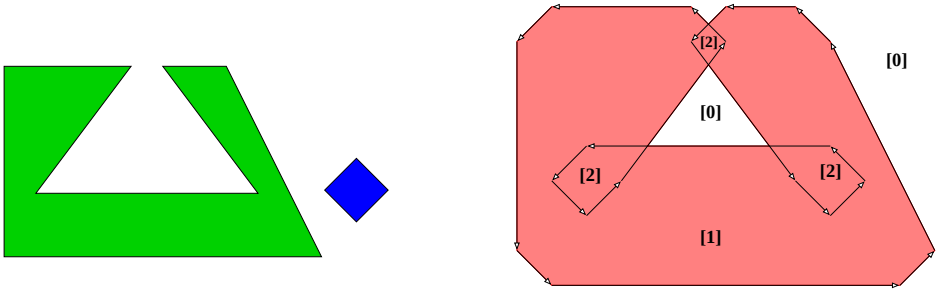


Figure 1: Computing the convolution of a convex polygon and a non-convex polygon (left). The convolution consists of a single self-intersecting cycle, drawn as a sequence of arrows (right). The winding number associated with each face of the arrangement induced by the segments forming the cycle appears in brackets. The Minkowski sum of the two polygons is shaded.

functions are limited to performing *regularized* Minkowski-sum computations, which eliminate low-dimensional features of the output.

Another approach to computing the Minkowski sum of two polygons is calculating the *convolution* of the boundaries of P and Q [7, 8]. Ramkumar [15] used this approach to devise an efficient algorithm for computing the outer boundary of the Minkowski sum of two polygons.[3] To the best of our knowledge, our code is the first implementation of software for robust and exact computation of Minkowski sums that is based on the convolution method. As our experiments show, using the convolution method we construct intermediate geometric entities that are more compact than the ones constructed using the decomposition method. Consequently, we are able to obtain faster running times.

The rest of this paper is organized as follows: In Sec. 2 we review the definition of polygon convolution and develop the notation that will be used throughout the paper. In Sec. 3 we give the details of our implementation of the Minkowski-sum algorithm. We present experimental results in Sec. 4, and give some concluding remarks in Sec. 5.

2 Preliminaries

Guibas *et al.* [7] introduced the concept of convolutions of general *planar tracings*, giving a special attention to *polygonal tracings*, which are composed of a series of interleaved *moves* (translations in a fixed direction) and *turns* (rotations at a fixed location).

Given two polygons, P with vertices $(p_0, \ldots, p_{m-1})$ and Q with vertices $(q_0, \ldots, q_{n-1})$, we make a move by traversing a polygon edge $\overrightarrow{p_i p_{i+1}}$, and make a turn be rotating on a polygon vertex p_i from the direction of $\overrightarrow{p_{i-1} p_i}$ to the of direction $\overrightarrow{p_i p_{i+1}}$.[4] Without loss of generality, we can assume that both polygons

[3] The complexity of this boundary is $O(mn \cdot \alpha(n))$, where $\alpha(\cdot)$ is the functional inverse of Ackermann's function [9].

[4] Throughout this paper, when we increment or decrement an index of a vertex, we do so

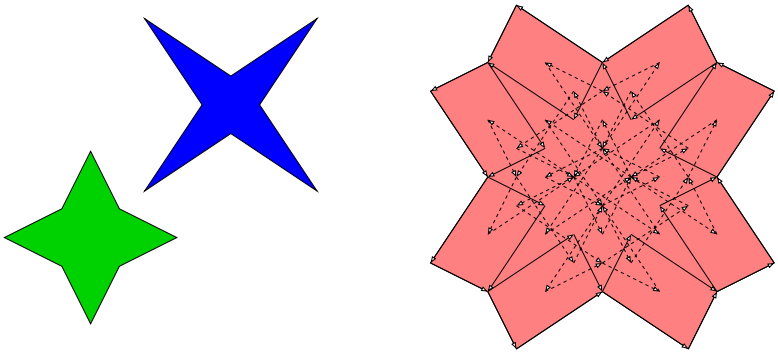


Figure 2: Computing the convolution of two non-convex octagons (left). The convolution consists of two cycles (right), one (solid arrows) comprises 32 line segments while the other (dashed arrows) contains 48 line segments, non of which lies on the boundary of the Minkowski sum (shaded).

are counterclockwise oriented. The *convolution* of these two polygons, denoted $P * Q$, is a collection of line segments of the form $\overrightarrow{(p_i + q_j)(p_{i+1} + q_j)}$, where the vector $\overrightarrow{p_i p_{i+1}}$ lies between $\overrightarrow{q_{j-1} q_j}$ and $\overrightarrow{q_j q_{j+1}}$,[5] and — symmetrically — of segments of the form $\overrightarrow{(p_i + q_j)(p_i + q_{j+1})}$, where the vector $\overrightarrow{q_j q_{j+1}}$ lies between $\overrightarrow{p_{i-1} p_i}$ and $\overrightarrow{p_i p_{i+1}}$. We can label the convolution segment as $\langle (i, i+1), j \rangle$ in the former case or $\langle i, (j, j+1) \rangle$ in the latter case. From the definition, it is clear that $P * Q$ contains at most $O(mn)$ line segments.

The segments of the convolution form a number of closed (not necessarily simple) polygonal curves called *convolution cycles*. The Minkowski sum $P \oplus Q$ is the set of points having a non-zero winding number with respect to these cycles (see, e.g., [14, Chap. 7] for the topological definition of the *winding number* and some examples). See Fig. 1 for an illustration.

In case both input polygons P and Q are convex, their convolution is a convex polygonal tracing. If only one polygon (say P) is convex, then $P * Q$ still contains a single cycle, which may not be simple (see [15] for a proof), as illustrated in Fig. 1. If both P and Q are non-convex, the convolution may comprise several cycles, and in order to compute the Minkowski sum of the polygons one has to consider the set of points having a non-zero winding number with respect to all cycles (see Fig. 2 for an illustration).

3 Implementation Details

Given two simple polygons P and Q having m and n vertices, respectively, we compute their Minkowski sum in three steps: first we compute the cycles of the convolution $P * Q$, then we construct the planar arrangement induced by the

modulo the size of the polygon. Indeed, $\overrightarrow{p_{m-1} p_0}$ is also a valid polygon edge.

[5] We say that a vector $\vec{v}$ lies between two vectors $\vec{u}$ and $\vec{w}$, if we reach $\vec{v}$ strictly before reaching $\vec{w}$ if we move all three vectors to the origin and rotate $\vec{u}$ counterclockwise. Note that this also covers the case where $\vec{u}$ has the same direction as $\vec{v}$.

ComputeConvolutionCycle $(P, i_0;\ Q, j_0)$

let $\mathcal{C} \longleftarrow \varnothing$.
let $i \longleftarrow i_0$. **let** $j \longleftarrow j_0$.
let $s \longleftarrow (p_i + q_j)$.
do:
  let *inc_P* $\longleftarrow$ IsBetweenCounterclockwise $(\overrightarrow{p_i, p_{i+1}};\ \overrightarrow{q_{j-1}q_j}, \overrightarrow{q_j q_{j+1}})$.
  let *inc_Q* $\longleftarrow$ IsBetweenCounterclockwise $(\overrightarrow{q_j, q_{j+1}};\ \overrightarrow{p_{i-1}p_i}, \overrightarrow{p_i p_{i+1}})$.
  if *inc_P* = True, **then**:
    let $t \longleftarrow (p_{i+1} + q_j)$.
    Push the segment $\vec{st}$, labelled $\langle (i, i+1), j \rangle$, into $\mathcal{C}$.
    let $s \longleftarrow t$.
    let $i \longleftarrow i + 1$ (modulo the size of P).
  if *inc_Q* = True, **then**:
    let $t \longleftarrow (p_i + q_{j+1})$.
    Push the segment $\vec{st}$, labelled $\langle i, (j, j+1) \rangle$, into $\mathcal{C}$.
    let $s \longleftarrow t$.
    let $j \longleftarrow j + 1$ (modulo the size of Q).
while $i \neq i_0$ **or** $j \neq j_0$.
return $\mathcal{C}$.

segments that form the convolution cycles, and finally we extract the Minkowski sum from this arrangement.

3.1 Computing the Convolution Cycles

Guibas and Seidel [8] show how to compute the convolution cycles of two polygons in optimal $O(m + n + K)$ time, where $K = |P * Q|$. However, they make some general-position assumptions on the polygons (e.g., an edge of P cannot have the same direction as an edge in Q) and cannot handle degenerate inputs. In this section we present a simple and robust algorithm, whose asymptotic running time is $O(m + n + \min\{m_r n, n_r m\} + K)$, where m_r and n_r are the number of reflex vertices in P and Q, respectively. As our experiments show, the running time of the construction of the convolution cycles is in practice negligible with respect to the overall Minkowski-sum computation.

We start by describing a simple procedure that constructs a single convolution cycle $\mathcal{C}$ of two polygons $P = (p_0, \ldots p_{m-1})$ and $Q = (q_0, \ldots q_{n-1})$, starting from two vertices p_{i_0} and q_{j_0}, such that $p_{i_0} + q_{j_0}$ is a vertex on the cycle $\mathcal{C}$. See the pseudo-code listing of the procedure ComputeConvolutionCycle. Observe that in the main loop of the procedure we add at least one convolution segment each iteration. However, in degenerate situations, namely when $\overrightarrow{p_i, p_{i+1}}$ and $\overrightarrow{q_j, q_{j+1}}$ have the same direction, we add two segments to the cycle in a single iteration.

We next describe how to locate the pairs of indices i_0 and j_0 needed for the procedure above. We start by locating the bottommost vertex of P (the minimal vertex with respect to a yx-lexicographical order) and the bottommost vertex of Q. Assume, without loss of generality, that these are the vertices p_0 and q_0. These vertices are not reflex, and it is clear that either $\overrightarrow{p_0 p_1}$ lies between the edges around q_0, or vice-versa. We can therefore compute a convolution cycle starting from this pair of vertices.

If either of the polygons is convex (that is, $m_r = n_r = 0$), then the convolution consists of a single cycle, and we are done. Otherwise, we traverse the reflex vertices of Q (we assume, without loss of generality that $n_r m < m_r n$), and for each such vertex q_{j_0} we go over all vertices of P and try to locate a vertex p_{i_0}, such that $\overrightarrow{p_{i_0}p_{i_0+1}}$ lies between $\overrightarrow{q_{j_0-1}q_{j_0}}$ and $\overrightarrow{q_{j_0}q_{j_0+1}}$. When we locate such a vertex pair and such that the label $\langle (i_0, i_0+1), j_0 \rangle$ has not been used (for this purpose we maintain an auxiliary set of *used labels*; this set can be represented using a hash table, such that each access to the set takes $O(1)$ time on average), we invoke the procedure above to compute an additional convolution cycle.

3.2 Obtaining the Minkowski Sum from the Convolution Cycles

Flato [4, App. A.1] describes a simple algorithm for computing the union of a set of simple polygons having counterclockwise orientation. He constructs the arrangement of the directed segments that correspond to the polygon edges (referred to as the *boundary segments*), namely the subdivision they induce on the plane into one unbounded face and several bounded faces. The arrangement is represented using a doubly-connected edge-list (DCEL for short; see, e.g., [3, Chap. 2]), such that it is easy to perform a breadth-first search traversal of all arrangement faces, starting from the unbounded face (whose winding number is of course 0), and compute the winding number of each face. The same arguments used for proving the correctness of the polygon-union algorithm in [4, App. A.1] also hold for the case of convolution cycles. We can therefore construct the arrangement of all directed segments constituting the convolution cycles, compute the winding numbers of the arrangement faces and output the union of the faces with a positive winding number. Consider for example Fig. 1, where we correctly identify the hole in the Minkowski sum, as it is represented by a face whose winding number is 0.

So far we have described an algorithm that computes the regularized Minkowski sum $P \hat{\oplus} Q$, namely the closure of the interior of $P \oplus Q$, as all 1-dimensional or 0-dimensional features of the sum (*antennas* and *isolated vertices*, respectively) are disregarded. In this case, the output is given as a polygon that represents the outer boundary of the sum and an additional, possibly empty, set of polygons that represents the holes inside this polygon. This representation is sufficient for many applications, but for other applications, such as motion planning and assembly planning, low-dimensional features may play an important role as they represent *tight passages*. Consider the example depicted in Fig. 3, where we wish to move a star-shaped polygon in a house, allowing translations only. The interior of the Minkowski sum in this case consists of all forbidden placements for the star center, such that each point on the boundary of the sum corresponds to a *semi-free* placement of the star, where it touches the walls but does not penetrate them.

Our software is also capable of outputting a planar arrangement that captures all features of the Minkowski sum of the input polygons.[6] Locating the

[6] Unlike previous CGAL versions, Version 3.2 of CGAL's arrangement package supports isolated vertices, so we can have 0-dimensional features in the output as well.

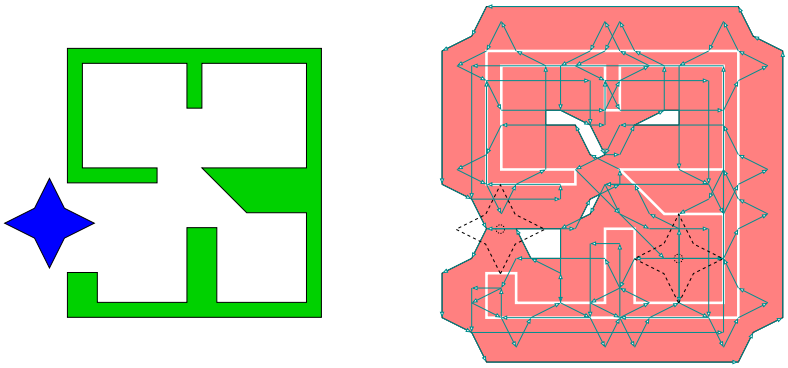


Figure 3: A house plan and a star-shaped polygon (left). The Minkowski sum of the two polygons (right) consists of low-dimensional features. For clarity, two copies of the star are drawn using a dashed line with their center positioned on these features. The left copy is located on an *antenna* on the Minkowski-sum boundary, such that the star can move along this antenna while touching the walls of the house. The right copy is located on an *isolated vertex*, which designates a location where the star can be placed without being able to move.

low-dimensional features incurs no asymptotic run-time penalty; see [4] for the full details of the algorithm including a proof of correctness.

4 Experimental Results

We have adapted parts of Flato's software [4] to comply with the interface of the latest version of CGAL (Version 3.2). As the original software consists of several thousands of lines of code, we did not convert all polygon-decomposition strategies; instead, we use the three convex-decomposition algorithm bundled with the Polygon Partitioning package of the CGAL public release. In addition, we added the small-side angle-bisector decomposition strategy (see below). In his experiments, Flato reports this strategy as the one that yields the fastest running times for the overall Minkowski-sum computation process (see also [1]). We ran our experiments with all four strategies listed below.

Optimal (Opt) The dynamic-programming algorithm of Greene [6] for computing an optimal decomposition of a polygon into a minimal number of convex sub-polygons. The main drawback of this strategy is that it runs in $O(n^4)$ time and requires $O(n^3)$ space in the worst case, where n is the number of vertices in the input polygon.

Hertel–Mehlhorn (HM) The approximation algorithm suggested by Hertel and Mehlhorn [10], which triangulates the input polygon and proceeds by throwing away unnecessary triangulation edges. This algorithm requires $O(n)$ time and space and guarantees that the number of sub-polygons it generates is not more than four times the optimum.

Greene (Gre.) Greene's approximation algorithm [6] which computes a con-

vex decomposition of the polygon based on its partitioning into y-monotone polygons. This algorithm runs in $O(n \log n)$ time and $O(n)$ space, and has the same approximation guarantee as Hertel and Mehlhorn's algorithm.

Small-side angle-bisector (SSAB) A heuristic improvement to the angle-bisector decomposition method suggested by Chazelle and Dobkin [2]. It starts by examining each pair of reflex vertices in the input polygon such that the entire interior of the diagonal connecting these vertices is contained in the polygon. Out of all available pairs, it selects p_i and p_j, such that the number of reflex vertices from p_i to p_j (or from p_j to p_i) is minimal. The polygon is split by the diagonal $p_i p_j$ and the process continues recursively on both resulting sub-polygons. In case it is not possible to eliminate two reflex vertices at once any more, each reflex vertex is eliminated by an angle bisector emanating from it. The entire process takes $O(n^2)$ time.

In the original implementation, the intersections between the angle bisectors and the polygon edges induce *Steiner points* in the decomposed sub-polygons. We have slightly modified the algorithm, such that instead of eliminating a reflex vertex p_i using an angle bisector, we look for another vertex p_{j^*}, such that $p_i p_{j^*}$ is contained in the polygon, and such that the ratio between the two angles $\measuredangle p_{i-1} p_i p_{j^*}$ and $\measuredangle p_{j^*} p_i p_{i+1}$ that $p_i p_{j^*}$ induces is as close to 1 as possible. Our experiments show that this modified approach yields very good decompositions, while avoiding the introduction of Steiner points, which may lead to more complex computations.

The original Minkowski-sum software was based on the arrangement package of CGAL Version 2.0, which supported only the *incremental* construction of arrangements, inserting curves one at a time. In the current CGAL version, it is possible to construct arrangements *aggregately*, so all boundary segments are inserted together using a sweep-line algorithm. Constructing an arrangement aggregately is asymptotically more efficient for line segments that sparsely intersect, as happens in our case. This theoretical argument is also backed up by experiments that show that constructing an arrangement of line segments aggregately is usually one order of magnitude faster than constructing it incrementally. We have therefore implemented an aggregated version of the union algorithm, as described in Sec. 3.2, and did not try the incremental union algorithm, as described in [4, Chap. 3].

We have conducted a large number of tests with various input sets. Here we report on eight representative input sets containing polygon pairs, most of them taken from [1] (see Figs 4 and 5 for illustrations). All data sets are available on-line at `http://www.cs.tau.ac.il/~wein/software/`:

Chain: A chain-shaped polygon with 82 vertices (37 of them are reflex), and a star-shaped polygon with 30 vertices (6 reflex).

Stars: Two star-shaped polygons, each containing 40 vertices (14 reflex).

Comb: A comb-shaped polygon containing 53 vertices (24 reflex) and a convex polygon with 22 vertices. This input set introduces the worst-case

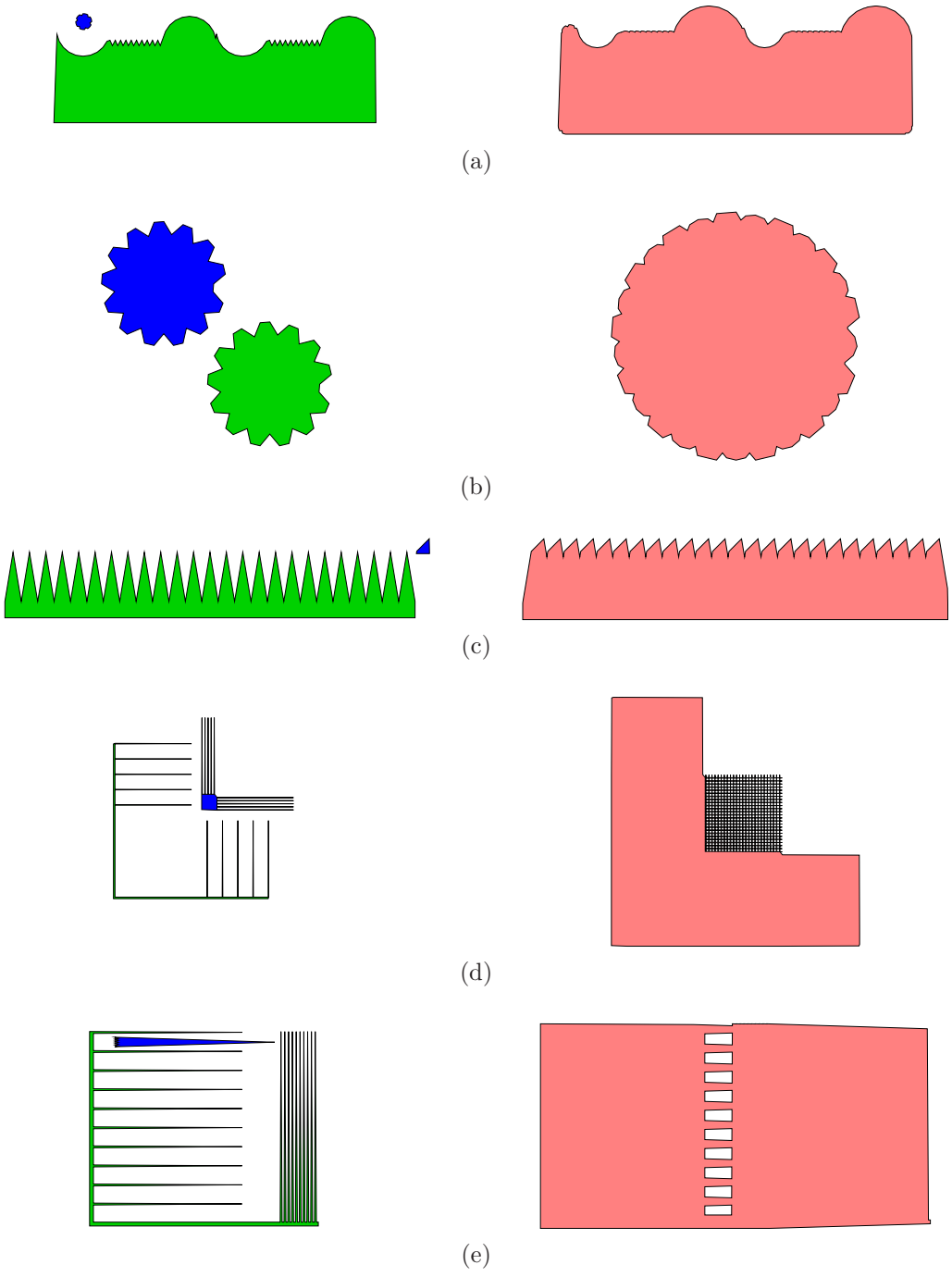
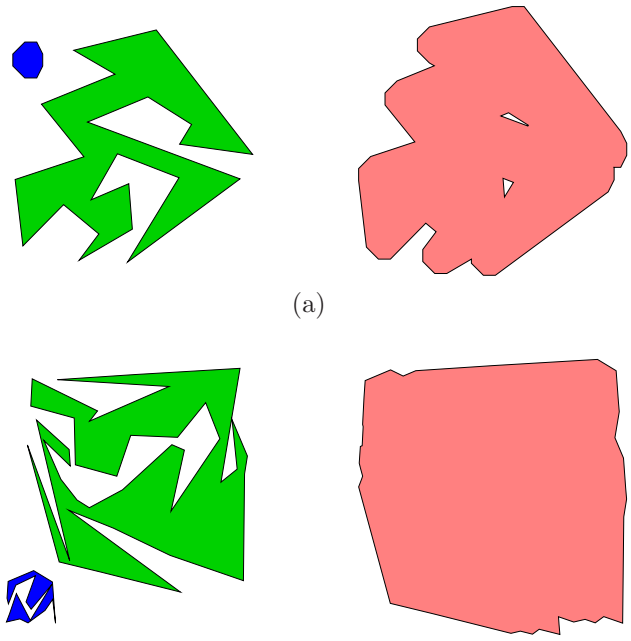


Figure 4: Samples of input polygons (left) and their Minkowski sums (right): (a) chain; (b) stars; (c) comb; (d) fork; (e) knife.



(a)

(b)

Figure 5: Samples of input polygons (left) and their Minkowski sums (right): (a) cavity; (b) random.

complexity for the sum of a convex and a non-convex polygon.

Fork: The well-known example for a Minkowski sum of size $O(m^2n^2)$ [11]. The large "fork" consists of 34 vertices (19 reflex) and the smaller one has 31 vertices (18 reflex).

Cavity: A random-looking polygon with 22 vertices (10 reflex) and a small convex octagon, chosen to fit some of the cavities in the larger polygon.

Random: Two random-looking polygons, with 40 and 20 vertices (19 and 8 reflex vertices, respectively).

Knife: A large polygon with 64 vertices (40 reflex) and a small polygon with 12 vertices (5 reflex), that can fit into the cavities on the larger polygon.

Country: The map of Israel, represented as a polygon with 50 vertices (24 reflex), and a smaller polygon with 30 vertices (8 reflex).

Table 1 summarizes the performance of the Minkowski-sum computations for the selected input sets, using the polygon-decomposition method. We give the running times, as obtained on a Pentium IV 3 GHz machine with 2 Gb of RAM, and averaged over 100 executions for each decomposition strategy. We also indicate — for the strategy that achieved the fastest running time on each input set — the numbers of sub-polygons k and ℓ in the decompositions of the two input polygons, the total number S of segments in all $k\ell$ Minkowski sums,

Table 1: Running times (measured in milliseconds) for the Minkowski-sum computation using various decomposition strategies. The running time of the for the fastest strategy in each case is shown in bold, and its construction statistics are also given.

				Decomp.	Arrangement size			Running time			
Input set	k	ℓ	S	time	$\|V\|$	$\|E\|$	$\|F\|$	Opt.	HM	Gre.	SSAB
chain	35	7	2520	123	7239	13627	6390	**390**	424	578	444
stars	17	17	2446	13	12955	25624	12671	488	744	867	**477**
comb	26	1	650	4	671	769	100	32	58	**17**	37
fork	12	11	1048	6	6827	13377	6552	287	524	1220	**260**
cavity	14	1	160	1	167	244	79	8	7	**6**	8
random	20	10	1540	15	8209	16310	8103	**234**	349	376	320
knife	22	6	1108	9	7721	15150	7431	249	370	1630	**232**
country	16	8	1344	8	5126	9772	4648	338	798	410	**188**

and the size of the arrangement (number of vertices, edges and faces) induced by the boundary segments.

It should be mentioned that the running times stated in Table 1 are sometimes two orders of magnitude faster than the ones reported in [1]. This can be partly attributed to the fact that we used a faster machine in our experiments,[7] and mostly due to the improvements in the new version of CGAL's arrangement package, which now handles intersections of line segments more efficiently [5] due to numerous improvements, such as the reduction of the number of geometric operations and predicates the arrangement-construction algorithms invoke [16]. Moreover, we use the predefined CGAL kernel, which uses interval arithmetic to filter exact computations with rational numbers, and helps reducing the running times even further. The exact rational number-type we use is provided by Gnu's multi-precision library (GMP Version 4.1).

Table 2 summarizes the performance of the Minkowski-sum computations for the selected input sets using the convolution method. We indicate the numbers of convolution cycles N_c, the total number K of convolution segments and the size of the induced arrangement. We also include a column that states the running time using the fastest decomposition scheme in each case (as given in Table 1) and the running times of the Minkowski-sum function provided by LEDA (Version 4.4) on the various input sets. We used the exact rational kernel of LEDA, which also employs arithmetic filtering to speed up the computations with an exact rational number-type.

In almost all cases, the convolution methods yields faster running times than the fastest decomposition scheme (recall that LEDA also employs the polygon-decomposition method). This is attributed to the fact that the convolution method usually induces a smaller arrangement as its intermediate construct. As the total running-time is clearly dominated by the arrangement-construction time, the convolution method may achieve a speed-up of up to a factor of 5 in

[7] Flato reports using a 500 MHz Pentium III machine.

Table 2: Running times (measured in milliseconds) and construction statistics for the Minkowski-sum computation using the convolution method.

Input set	N_c	K	Conv. time	Arrangement size			Total time	Best decomp.	Using LEDA
				$\|V\|$	$\|E\|$	$\|F\|$			
chain	1	1452	7	2077	2868	793	**69**	390	463
stars	2	1200	7	3225	5482	2259	**92**	477	332
comb	1	603	7	627	651	26	**17**	17	97
fork	1	1266	5	11203	22063	10862	**521**	260	266
cavity	1	110	1	135	161	28	**4**	6	21
random	1	580	3	2589	4698	2111	**62**	234	229
knife	1	876	5	5075	9749	4676	**184**	232	175
country	2	1050	5	1940	3064	1126	**61**	188	275

some cases. Moreover, the memory requirements for storing the intermediate arrangement are also considerably smaller.

The convolution method has another important advantage. As we learn from Table 1, the choice of a decomposition strategy may have drastic effects on the running time of the Minkowski-sum computation, and the best strategy can only be found by experimenting; when using the convolution method, no such experiments are needed.

The *fork* input set is the only example that exhibits slower running times when using the convolution method. This is due to the fact that the input polygons are decomposed very nicely by the SSAB decomposition-strategy, separating each of the fork prongs into a single thin rectangle, such that the Minkowski sums of the sub-polygons are nearly pairwise disjoint. On the other hand, as we have many pairs of parallel edges in the input polygons (note that all edges of the two forks are axes-parallel), the convolution cycle in this case contains many redundant loops that incur the greater run-time overhead.

5 Conclusions

We present an efficient implementation for computing Minkowski sums of simple polygons using the convolution method. This is the first software implementation of a robust algorithm based on the polygon-convolution method. It uses exact computation, and can handle inputs with many degeneracies, yielding topologically correct results. As our experiments show, the convolution method is superior to the polygon-decomposition method on almost all input sets, and improves the running times by a factor of 2–5. We intend to include our software in the next public release of CGAL (the forthcoming Version 3.3).

References

[1] P. K. Agarwal, E. Flato, and D. Halperin. Polygon decomposition for efficient construction of Minkowski sums. *Computational Geometry: Theory and Applications*, **21**:39–61, 2002.

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

[3] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition, 2000.

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

[5] E. Fogel, R. Wein, and D. Halperin. Code flexibility and program efficiency by genericity: Improving CGAL's arrangements. In *Proc. 12th Europ. Sympos. Alg.*, volume **3221** of *LNCS*, pages 664–676, 2004.

[6] D. H. Greene. The decomposition of polygons into convex parts. In F. P. Preparata, editor, *Computational Geometry*, volume **1** of *Adv. Comput. Res.*, pages 235–259. JAI Press, Greenwich, Conn., 1983.

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

[8] L. J. Guibas and R. Seidel. Computing convolutions by reciprocal search. *Discrete and Computational Geometry*, **2**:175–193, 1987.

[9] S. Har-Peled, T. M. Chan, B. Aronov, D. Halperin, and J. Snoeyink. The complexity of a single face of a Minkowski sum. In *Proc. 7th Canad. Conf. Comput. Geom.*, pages 91–96, 1995.

[10] S. Hertel and K. Mehlhorn. Fast triangulation of simple polygons. In *Proc. 4th Internat. Conf. Found. Comput. Theory*, volume **158** of *LNCS*, pages 207–218. Springer-Verlag, 1983.

[11] 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, 1991.

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

[13] K. Mehlhorn and S. Näher. LEDA*: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 2000.

[14] T. Needham. *Visual Complex Analysis*. Oxford University Press, Oxford, UK, 1997.

[15] G. D. Ramkumar. An algorithm to compute the Minkowski sum outerface of two simple polygons. In *Proc. 12th Sympos. Comput. Geom.*, pages 234–241, 1996.

[16] R. Wein, E. Fogel, B. Zukerman, and D. Halperin. Advanced programming techniques applied to CGAL's arrangement package. In *Proc. Wrkshp. Library-Centric Software Design*, 2005. `http://lcsd05.cs.tamu.edu/#program`.