An Algorithm to Compute the Minkowski Sum Outer-face of Two Simple Polygons

G.D. Ramkumar*

Department of Computer Science
Stanford University
Stanford, CA 94305, USA.
E-mail: *ramkumar@cs.stanford.edu*

Abstract

We use the concept of the convolution of two polygons as a tool for computing the outer-face of their Minkowski sum. Our method consists of traversing each cycle of the convolution, detecting self-intersections, and snipping-off the loops thus created. In order to detect self-intersections, we adapt the geodesic triangulation ray-shooting data structure to answer ray-shooting queries on a *dynamic* polygonal line of size n, in $O(\log^2 n)$ amortized time. We find the outer-face of the Minkowski sum of two simple polygons of size m and n respectively in time $O((k+(m+n)\sqrt{l})\log^2(m+n))$ where k is the size of the convolution and l is the number of cycles in the convolution. Although k can be $O(mn)$ in the worst case, it is much less for realistic input instances. If one of the polygons is convex, then the convolution has only one cycle, and the algorithm runs in $O(k\log^2(m+n))$ time. Existing methods to find the Minkowski sum outer-face rely on general algorithms to compute a single face in an arrangement of k line segments which take $O(k\log k\alpha(k))$ time. Our algorithm exploits a new insight into the relationship between convolution and Minkowski sum and, though asymptotically slower, has practical advantages for realistic polygon data.

*Supported by NSF Grant #s NSF-CCR-921-5219, NSF-IRI-930-6544

Computational Geometry'96, Philadelphia PA, USA
© 1996 ACM 0-89791-804-5/96/05..$3.50

1 Introduction

Minkowski sums arise commonly in robotics under the name of configuration space obstacles [9], when the robot motion is restricted to translation. The Minkowski sum $Q^N \oplus P$ of a Robot Q reflected through the origin with obstacle P is the the set of placements of Q with respect to P which cause the two polygons to intersect. The *outer-face* of the Minkowski sum is the infinite component of the complement of the Minkowski sum and represents the set of *free* placements of Q that can be reached by translation from infinity. In most robotics applications interior cavities of the Minkowski sum are uninteresting parts of the free space — what really matters is the region bounded by the outer-face. Given two simple polygons P and Q, we present in this paper an algorithm to compute the outer-face of the Minkowski sum of P and Q using the convolution of the two polygons made into tracings as the key tool.

Planar tracings and their convolutions were introduced as generalizations of ordinary polygons by Guibas, Ramshaw, and Stolfi [6] as part of the *kinetic framework* for computational geometry. One of the consequences of that framework is that convolution and Minkowski sum are closely related. In this paper, we show how to use convolution to find the outer-face of the Minkowski sum in time $O((k+(m+n)\sqrt{l})\log^2(m+n))$, where m and n denote the sizes of P and Q respectively, k is the size of the convolution $P * Q$, and l is the number of cycles in the convolution tracing (we assume $m \leq n$, and of course $l \leq k$). Note that the convolution is always of size $O(mn)$, while the Minkowski sum can be of size $\Theta(m^2n^2)$. We use an algorithm presented in [6] to compute the convolution of planar polygonal tracings in optimal time $O(m + n + k)$. The outer-face of the Minkowski sum can be computed by applying the general algorithm of [3] to compute a single face in an arrangement of k line segments in time $O(k\alpha(k)\log k)$. It is shown in [7] that the complexity of a single face in the Minkowski sum can in fact be $\Theta(mn\alpha(mn))$ in the worst case.

Unlike this approach, which only needs the edges of the convolution as a set of line segments, our algorithm exploits the way the segments are linked together into cycles in the convolution. The key idea is to detect self-intersections and excise the loops thus formed in the convolution. To detect self-intersections, we provide a novel scheme for performing ray-shooting against a dynamic polygonal line, using the geodesic triangulation structure of [2]. Our scheme relies on dividing the polygonal line into blocks of contiguous segments and storing a geodesic triangulation for each block. Ray-shooting queries are answered in amortized time $O(\log^2 k)$, where k is the size of the polygonal line. The problem of dynamic ray-shooting has been studied in a more general setting in [4], where the authors obtain a worst-case time bound of $O(\log^2(n))$ for ray-shooting against a dynamic planar subdivision of size n. Our technique only works for polygonal lines, but it is simpler and sufficient for our needs. Our outer-face algorithm, though again slightly inferior in asymptotic complexity to the single face method, is easy to implement and helps illuminate some new aspects of the connection between convolution and the Minkowski sum.

This paper is organized as follows. Section 2 introduces basic terms and outlines the relationship between convolution, fiber product, and Minkowski sum. In Section 3 we describe how to traverse a cycle of the convolution, detecting self-intersections, excising loops, and computing its outer boundary. In section 4, we discuss how to connect multiple cycles of the convolution together and traverse them in sequence using the algorithm of Section 3 to find the outer-face of the Minkowski sum. Section 5 concludes the paper.

2 Convolution and Minkowski sum

We recall briefly some terms introduced in the kinetic framework [5]. A *state* is a pair consisting of a position $\dot{s}$ in the plane and a direction $\vec{s}$. A *move* is a set of states with constant direction and position varying along a line segment parallel to the direction. A *turn* is a set of states with constant position and direction varying along an arc of the circle of directions. A *polygonal trip* consists of a continuous sequence of moves and turns; the trip is closed if it starts and ends at the same state. A *polygonal tracing* is a collection of closed polygonal trips. We think of each loop of a tracing as being traversed by a car which always faces in the direction of the state it is currently in. The winding number of a point with respect to a tracing has the traditional meaning.

Given a simple polygon P we denote as $\hat{P}$ the tracing corresponding to a counterclockwise traversal of its boundary with the car always moving forward (for details, see [5]). Let P, Q be two simple polygons with corresponding tracings $\hat{P}$ and $\hat{Q}$. The convolution of $\hat{P}$ and $\hat{Q}$ is defined as follows. If p and q are states in $\hat{P}$ and $\hat{Q}$ such that $\vec{p} = \vec{q}$, then the state $c = (\dot{p} + \dot{q}, \vec{p})$ is a state in the convolution. In other words, the convolution is a set of vector sums of states in $\hat{P}$ and $\hat{Q}$ where the two cars face in the same direction. A convenient way to visualize the convolution is to depict it as the *fiber product* FP(P,Q) defined as follows. Lay out the perimeter of P on the horizontal axis and the perimeter of Q on the vertical axis, and plot all pairs of boundary points that produce a state in the convolution (note that this representation collapses information about matching states in turns). A vertical edge of the fiber product corresponds to a convolution move derived from $\hat{Q}$ while a horizontal edge is derived from an edge of $\hat{P}$ (see Figure 1).

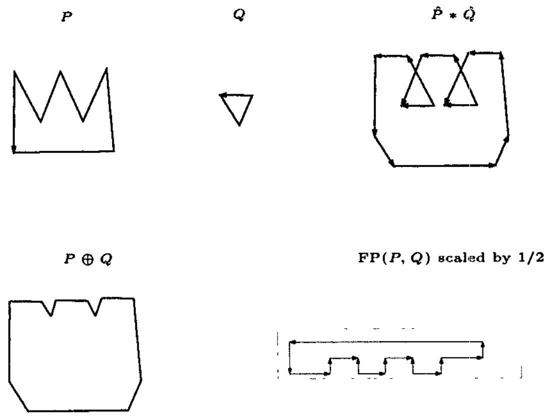

Figure 1: The convolution, Minkowski sum, and fiber product of two simple polygons.

The Minkowski sum $P \oplus Q$ of polygons P and Q consists of pair-wise vector sums of all interior or boundary points of P and Q. Note that this includes all pairs used in the definition of the convolution. From the fundamental theorem on convolutions [5], it follows that the winding number of a point x with respect to $\hat{P} * \hat{Q}$ is the number of connected components in the intersection of P^N translated by x and Q. The Minkowski sum $P \oplus Q$ is thus the non-zero winding number region of $\hat{P} * \hat{Q}$. From this it follows that the convolution pairs cover the boundary of the Minkowski sum; other pairs must be inside or can be on the boundary, but they cannot be outside. On the other hand, edges of the convolution can be interior, wholly or in part, to the Minkowski sum, e.g., when they separate regions of non-zero winding number.

A key advantage of the convolution is that it is stored as a collection of edge cycles without regard to how these cycles might intersect themselves or each other. This is the reason why the convolution is $O(mn)$ in size regardless of the geometry of the input polygons. The Minkowski sum, on the other hand, can be combinatorially much larger, since intersections of different convolution edges that are on the Minkowski sum boundary must be explicitly stored. A worst-case example of a Minkowski sum with complexity $\Theta(m^2n^2)$ is provided in [8].

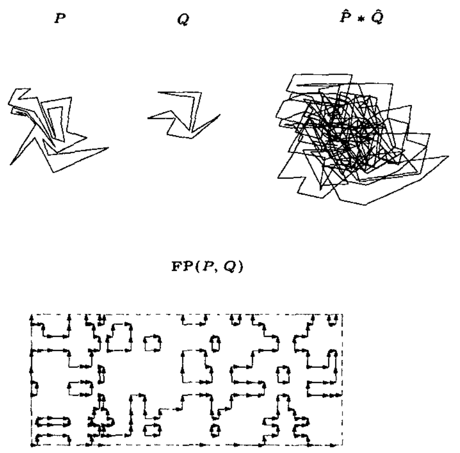

Figure 2: A larger example of the convolution and the fiber product (the fiber product is scaled to fit within the page).

2.1 Cycles of the convolution

The convolution consists of a number of closed polygonal trips that we call *cycles*. They may overlap in the plane, producing intersections. In the fiber product, however, the cycles are spatially separated out. This is because a pair of boundary features of P and Q can only produce a single feature in the convolution. This property makes the fiber product a convenient tool for visualizing the convolution, without the clutter introduced by its actual embedding in the plane. Figure 3 shows two simple polygons which produce a convolution containing two cycles. Each cycle contributes to the outer-face of the Minkowski sum, requiring our algorithm to consider both in order to find the outer-face. Figure 2 shows a large example of a convolution with cycles separated out in the corresponding fiber product.

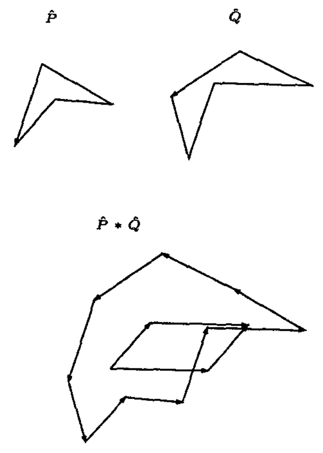

Figure 3: The convolution of these two polygons contains two cycles and each cycle contributes to the outer-face of the Minkowski sum.

Lemma 2.1 *If $\hat{Q}$ is a convex polygonal tracing and $\hat{P}$ is a simple polygonal tracing, the convolution $\hat{P} * \hat{Q}$ consists of a single cycle.*

Proof: Consider any pair of matching states, say (p_1, q_1), (p_2, q_2) in the convolution. Move the P car from p_1 continuously along P (in either direction) till p_2 is reached. Since Q is convex, the Q car always match any arbitrary (left or right) turn the P car makes, simply by moving (forwards or backwards) along Q from its current state. Thus any two states on the convolution can be connected and therefore there can be only one cycle.

2.2 Computing the convolution

Guibas and Seidel [6] show how to compute the convolution of two simple polygonal tracings $\hat{P}$ and $\hat{Q}$ in optimal time $O(m + n + k)$, where m, n, and k are the sizes of $\hat{P}, \hat{Q}$, and $\hat{P} * \hat{Q}$ respectively. They compute the set of moves and turns comprising the convolution and sort them using radix sort to find the order in which they are traversed in the convolution. We recall below an alternative way to find the traversal order using signs assigned to turns.

In the kinetic framework of [5] it is shown how to assign *signs* to moves and turns of tracings so that the convolution derives consistent signs for its moves and turns (positive moves are forward and positive turns are to the left). The sign of a move of the convolution is then the product of the signs of the move and turn that paired to cause it. For counterclockwise simple polygonal tracings, all moves are positive, left turns are positive, and right turns are negative. Hence the convolution of a move with a left turn is a positive (i.e. forward) move and that with a right turn is a negative (i.e. backward) move. It is then possible to traverse the convolution starting from any move, going forward or backward depending on its sign, and so on.

Since we need to compute the outer-face, we start the traversal with a convolution vertex that contributes to the outer-face. We do this by choosing to start with a vertex that is extreme in some direction: for instance the convolution of the two left-most turns of $\hat{P}$ and $\hat{Q}$. Let us now see how the traversal of a single cycle of the convolution is performed.

3 Outer boundary of a convolution cycle

In this section, we focus on the computation of the outer boundary of one cycle of the convolution, and show that the outer boundary of a cycle composed of k moves can be computed in time $O(k \log^2 k)$. The algorithm works as follows: starting from an outer-face vertex, we grow a simple polygonal chain, one move at a time, keeping a ray-shooting data structure that allows us to detect self-intersections. When a self-intersection occurs, we add the intersection point as a new vertex, and then "pinch off" and excise the edge cycle thus formed. We describe the algorithm in more detail below and show that it correctly computes the outer boundary of a cycle. We then show how the geodesic triangulation data structure of [2] can be adapted for efficient dynamic ray-shooting towards a growing simple polygonal chain.

3.1 Description of the algorithm

The algorithms works as follows: Start from an outer vertex and grow the chain one edge at a time. The new endpoint of the next edge added becomes the current active end of the chain where the next edge will go. For each edge being added, we shoot *backwards* from the free endpoint of the new edge towards the active vertex of the polygonal chain, to see if the edge intersects the polygonal chain grown so far – see Figure 4. If it does not, the edge is accepted and added to the chain. If it does, we break both the new edge and the polygonal chain at this first intersection found. Because we shot backwards, we are guaranteed to find the last intersection between the chain and the edge (if we were to imagine the new edge growing out of then active vertex). The part of the original chain between the intersection and the active vertex is removed (this is the cycle being pinched off), and then the remaining fragment of the new edge is added to give us a new active vertex.

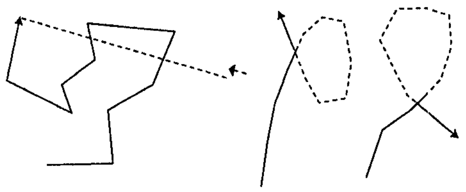

Figure 4: Snap shots of the running of the algorithm: left picture depicts the ray-shooting, and the middle and right pictures show excised clockwise and counterclockwise loops in dotted lines.

3.2 Convolution loops

The proposed algorithm relies on an important observation, which justifies the pinching off: if the polygonal line starts from a point that lies on the outer face of the Minkowski sum, then no edge (or part of an edge) in a loop can lie in the outer face. For the argument below, recall that we have oriented all our polygons in a counterclockwise manner.

Lemma 3.1 *Let π be a path following a cycle of the convolution and starting from a point O known to lie on the outer face. If π self-intersects, then no point on the loop thus created can be on the outer face of that cycle.*

Proof: Clockwise loops are completely enclosed within the convolution because the interior of the convolution is always to the left of a convolution edge according to the way the edge is traversed in going around the cycle. This means that the region enclosed by the clockwise loop is completely within the outer-face of the Minkowski sum.

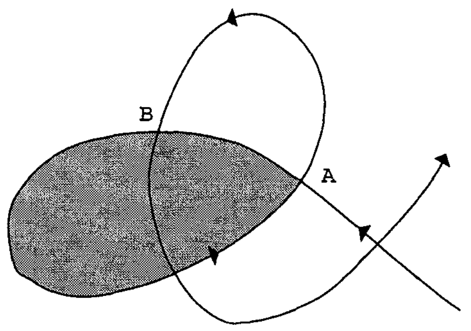

Figure 5: A counterclockwise loop of the convolution with the excised loop shown in grey.

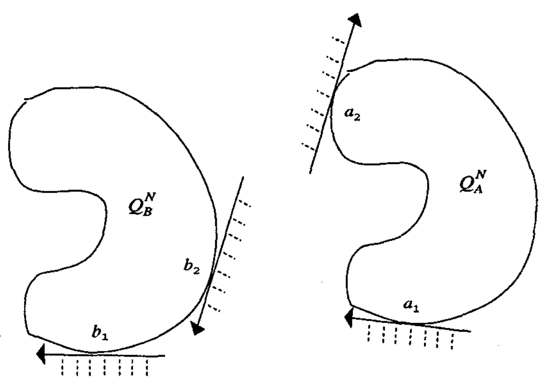

Figure 6: Placements Q_A^N and Q_B^N of Q^N corresponding to intersections A and B of Figure 5.

Let us now consider a counterclockwise loop ℓ created by the first self-intersection of π at the point A. We can assume that A itself is on the outer face, for if π had already entered inside the convolution by crossing in at an earlier point X, then X itself defines another loop whose set of edges includes those of ℓ and to which our argument below can be applied: if no portion of that cycle can be on the outer face, *a fortiori* the same applies to ℓ.

Now around A the loop cannot contribute to the outer-face, for it separates regions of winding number 1 and 2 respectively – see Figure 5. So the only way for ℓ to appear on the outer face is for π to cross again into ℓ so as to bring the lowest of these winding numbers down to 0. Let the such a crossing be B. By reasoning as above, we can assume that B is on the outer face as well (if not, keep going till such a B is found). Points A and B are *double points* in the convolution: each of them denotes a placement of Q^N that causes it to touch the boundary of P in two places. Let us denote these placements as Q_A^N and Q_B^N respectively, and the tangenecy points as a_1, a_2 and b_1, b_2 respectively. The portion σ of the boundary of P of that gener-

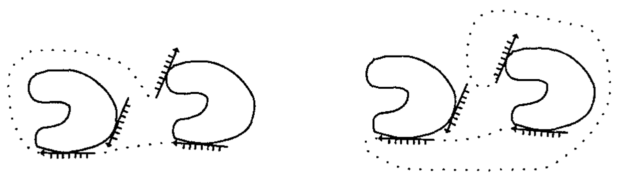

Figure 7: A counterclockwise loop cannot be on the outer-face.

ates the portion of π seen so far visits these points in the order a_1, b_1, a_2, b_2, as depicted in Figure 6. Since the cycle is counterclockwise, the interior of P must locally always be as depicted in Figure 6. Note also that, since A and B are on the outer face by assumption, the boundary of P cannot enter the interior of Q_A^N of Q_B^N.

But now we have a topological impossibility: there is no way to connect the points a_1, b_1, a_2, b_2 into a cycle in the order given, without either self intersections or going through the regions Q_A^N or Q_B^N. Some of the possibilities are shown in Figure 7. But this contradicts the assumption that there is a simple counterclockwise oriented polygon P passing through these points and avoiding regions Q_A^N and Q_B^N.

It is worth noting that the proof relies on the input tracings representing simple polygons with only forward moves. This is the only case which is of relevance to robotics.

3.3 Dynamic ray-shooting structure

A geodesic triangulation based ray-shooting data structure for a simple polygon in the plane was presented in [2]. The data structure is computed in linear time and supports ray-shooting queries in $O(\log k)$ time where k is the size of the polygon. This technique can be extended for polygonal lines by computing geodesic triangulations that stay to the left and to the right of the lines respectively. Ray-shooting queries can be made to the left as well as the right of the polygonal line.

The ray-shooting data structure must handle updates of the polygonal line at one end. In order to efficiently process these updates we divide the polygonal line into contiguous blocks of segments and store a static ray-shooting structure for each block. The number of blocks is chosen to be $O(\log k)$, where k is the size of the polygonal line. The ray-shooting cost summed over all the blocks is $O(\log^2 k)$. We now show how to choose the lengths of the blocks such that the amortized cost of updates of the polygonal line is minimized.

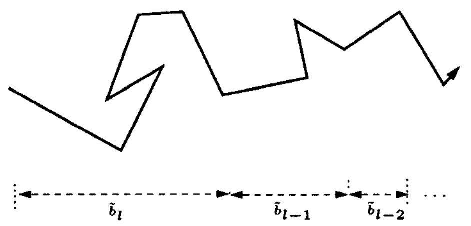

Figure 8: Dividing the polygonal line into blocks of segments.

Suppose the current length of the polygonal line is j. Let $\tilde{j} = (\tilde{b}_\lambda, \ldots, \tilde{b}_0)$ be a *redundant* binary representation of j: where each $\tilde{b}_i$ is 0, 1, or 2, and $j = \Sigma(\tilde{b}_i 2^i)$. The polygonal line is divided into blocks of length $\tilde{b}_i 2^i$ with the active end stored in the smallest block. Since a bit is allowed to be 2, a number can have redundant representations: for instance 5 can be either 101 or 21. Addition involves carry propagation and subtraction involves borrow propagation using the following table.

Operation	0	1	2	10	20
Increment	1	2	10	-	-
Decrement	-	0	1	01	11

For example incrementing 1222 yields 2111 and decrementing 200 yields 111. Let us now look at what happens when an insert (or delete) happens at the active end of the polygonal line of length j. The current representation $\tilde{j}$ is incremented (or decremented) and for each bit position $\tilde{b}_i$ that changes the corresponding block is reconstructed or destroyed. Since the length of this block is $\tilde{b}_i 2^i$, the cost of changing bit position $\tilde{b}_i$ is $O(2^i)$. We now show that the amortized cost of updates is $O(\log k)$.

3.4 Amortized analysis of the redundant binary counter

We perform an analysis of the redundant binary counter in an abstract setting in which the cost of flipping bit i is $O(2^i)$. Let us assume that the total number of increment and decrement operations is k (bounded by the size of the convolution), so that the number of bits in the counter is bounded by $\lambda = \lceil \log k \rceil$. We show that the amortized cost of each operation on the counter is $O(\log k)$.

For a representation $\tilde{j} = (\tilde{b}_\lambda, \ldots, \tilde{b}_0)$ of j, the value of the counter at time t, we associate a potential for each bit position ℓ:

$$\phi_\ell(t) = \left| \sum_{0 \le i < \ell} \tilde{b}_i 2^i - 2^\ell \right|$$

An increment that flips $\nu + 1$ bits causes the following change in representation:

$$(\ldots, \tilde{b}_\nu, 2, \ldots, 2) \longrightarrow (\ldots, \tilde{b}_\nu + 1, 1, \ldots, 1)$$

Before the increment, we have, for all $\ell \le \nu$,

$$\begin{aligned} \phi_\ell(t) &= \left| \sum_{0 \le i < \ell} 2 \cdot 2^i - 2^\ell \right| \\ &= 2^\ell - 2 \end{aligned}$$

After the increment, the potential at the same position becomes:

$$\begin{aligned} \phi_\ell(t+1) &= \left| \sum_{0 \le i < \ell} 1 \cdot 2^i - 2^\ell \right| \\ &= -1 \end{aligned}$$

All potentials associated with positions higher than ν simply increase or decrease by 1. Thus the difference in the total potential $\phi = \sum_i \phi_i$ is:

$$\begin{aligned} \phi(t+1) - \phi(t) &= \sum_{\nu < i \le \lambda} \phi_i(t+1) - \phi_i(t) \\ &\quad + \sum_{0 \le \ell \le \nu} \phi_\ell(t+1) - \phi_\ell(t) \\ &<= \lambda - \nu + \sum_{0 \le \ell \le \nu} (-2^\ell + 1) \\ &= \lambda - 2^{\nu+1} + 1 \end{aligned}$$

The cost of flipping ν bits is $O(2^\nu)$. The change of potential pays for all but λ of the cost. Hence amortized cost of each operation is $\lambda + 2$. The decrement operation, if it flips $\nu + 1$ bits, causes the following change:

$$(\ldots, \tilde{b}_\nu, 0, \ldots, 0) \longrightarrow (\ldots, \tilde{b}_\nu - 1, 1, \ldots, 1)$$

and the exact same analysis carries through.

Although the total potential function can be negative, it reaches its minimum when the counter is all 2 or all 0: namely the potential is $-2^{\lambda+2} = \Theta(k)$, adding only a constant cost per operation. Thus, the redundant counter dynamizes the geodesic triangulation data structure with an amortized cost of $O(\log k)$ per operation.

4 Connecting multiple cycles

The algorithm of Section 3 hinges on ray-shooting against a *connected* polygonal line and no obvious extension seems possible to within a comparable time bound when multiple disjoint cycles can be present. In order to apply our technique on multiple cycles of the convolution we attempt to connect these cycles together and form a single cycle. If the paths connecting the cycles are known to lie inside the Minkowski sum, then the outer-face will remain unchanged when we apply the loop excising process to this composite cycle. The running time of the algorithm thus depends on the total length of this path, namely the size of the convolution plus the length of the connecting paths.

Let us now see how to connect the cycles of the convolution visualized on the fiber product. If the input polygons P and Q are of sizes m and n the fiber product $\mathrm{FP}(P, Q)$ is on a rectangle of size $m \times n$. A grid point in the rectangle is a pair of vertices (p, q) of P and Q, and corresponds to their vector sum $(p + q)$. By definition, $(p + q)$ lies in the Minkowski sum of P and Q. Thus, a path connecting two points in the fiber product corresponds to a path in the Minkowski sum. Connecting cycles of the fiber product hence amounts to connecting cycles of the convolution using paths that lie in the Minkowski sum.

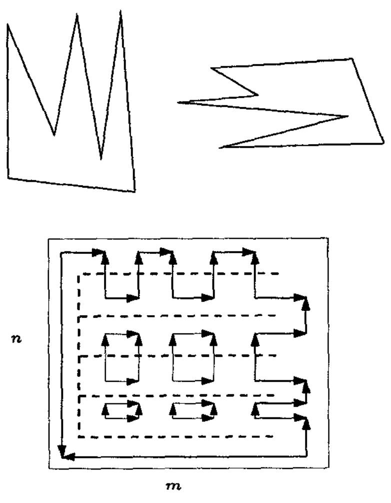

Figure 9: Shows two polygons and their convolution fiber product and a superimposed comb to connect the cycles.

Let us now see how to connect cycles of the fiber product. Suppose there are l cycles in the fiber product. Overlay a comb with $\sqrt{l}$ horizontal teeth and a vertical spacing of $\frac{n}{\sqrt{l}}$, as in Figure 9. Now connect each cycle of the fiber product to its nearest tooth using a vertical segment. This construction connects together all the cycles. Let us now compute the length of the connecting paths. There are $\sqrt{l}$ teeth in the comb each of length m yielding a total comb length of $m\sqrt{l}$. Each of the l cycles can be connected to a tooth with a vertical segment smaller than $\frac{n}{\sqrt{l}}$, yielding a total length of at most $n\sqrt{l}$. The final length of all the added paths is therefore $(m + n)\sqrt{l}$. It can be shown that this is optimal to within a constant factor for the length of connecting paths that lie on the fiber product. In particular, if the fiber product cycles are arranged in a regular $\sqrt{l} \times \sqrt{l}$ subgrid of the $m \times n$ fiber product grid, $(m + n)\sqrt{l}$ is a lower bound for the length of any set of interconnecting paths. Using these connecting paths (and possibly doubling them over), we can now easily form a single cycle whose edges include all the original edges of the convolution cycles.

We can now compute the total running time of our loop-cutting algorithm for a convolution with multiple cycles. The length of the polygonal line traversed is $O(k + (m + n)\sqrt{l})$. Amortized cost of ray-shooting and updating the polygonal line is $O(\log^2(k + (m + n)\sqrt{l}))$ which simplifies to $O(\log^2(m + n))$ since $k \leq mn$. Hence the total cost of finding the outer-face of the Minkowski sum is $O((k + (m + n)\sqrt{l})\log^2(m + n))$.

Note that this bound remains roughly quadratic even when $l = \Theta(k) = \Theta(mn)$. Though a quadratic number of cycles is possible, we expect that in most practical situations l will be quite small.

5 Conclusion

In this paper we have presented a method to use the convolution of two simple polygons to find the outer-face of their Minkowski sum in time $O((k + (m + n)\sqrt{l})\log^2(m + n))$, where m, n are the sizes of the polygons, k is the size of their convolution, and l is the number of cycles in the convolution. Three key ingredients went into our algorithm. We adapted the geodesic triangulation data structure for ray-shooting queries against a polygonal line of size k to work in amortized time $O(\log^2 k)$ in a dynamic context. If one of the input polygons is convex, then the convolution has a single cycle. We proved a topological lemma that allows us to computer the outer face by traversing this cycle and discarding any loops formed by self-intersections; the loops are discovered using the dynamic ray shooting structure and the total running time is $O(k \log^2(m+n))$. In the general case, our algorithm uses an additional technique to connect the cycles of the convolution into a single cycle using paths inside the convolution of total length $O((m+n)\sqrt{l})$, and then applies the single cycle technique.

An interesting open problem here is whether the cycles can be connected using paths of length $O(m+n+l)$. Our method only considers paths in the fiber product, which correspond to pairs of points on the boundary of P and Q. For such pairs the lower bound example mentioned earlier applies, but it may be possible to do better if we allow pairing of points *inside* P and Q. Another problem raised is whether ray-shooting queries against a dynamic polygonal line of length k can be performed in $O(\log k)$ time.

A theory of polyhedral tracings and their convolution is introduced in [1]. An interesting question there is whether the relationship between the convolution and the Minkowski sum explored in this paper can be extended to three dimensions, and whether a similar algorithm can be formulated to compute the outer-face of the Minkowski sum.

Acknowledgments

I would like to thank my advisor Leonidas Guibas for his contributions to these results, and my colleague Julien Basch for many helpful discussions.

References

[1] J. Basch, L.J. Guibas, G.D. Ramkumar, and L. Ramshaw. Polyhedral tracings and their convolution. *Submitted.*

[2] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. In *Proc. 18th Internat. Colloq. Automata Lang. Program.*, volume 510 of *Lecture Notes in Computer Science*, pages 661–673. Springer-Verlag, 1991.

[3] B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir, and J. Snoeyink. Computing a face in an arrangement of line segments and related problems. *SIAM J. Comput.*, 22:1286–1302, 1993.

[4] M. T. Goodrich and R. Tamassia. Dynamic ray shooting and shortest paths via balanced geodesic triangulations. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 318–327, 1993.

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

[6] L. J. Guibas and R. Seidel. Computing convolutions by reciprocal search. *Discrete Comput. Geom.*, 2:175–193, 1987.

[7] Sariel Har-Peled, Timothy M. Chan, Boris Aronov, Dan Halperin, and Jack Snoeyink. The complexity of a single face of a Minkowski sum. In *Proc. 7th Canad. Conf. Comput. Geom.*, pages 91–96, 1995.

[8] 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.

[9] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. Comput.*, C-32:108–120, 1983.