Speeding Up the Incremental Construction of the Union of Geometric Objects in Practice*

Eti Ezra   Dan Halperin   Micha Sharir

School of Computer Science
Tel Aviv University
{estere,danha,michas}@post.tau.ac.il

Abstract

We present a new incremental algorithm for constructing the union of n triangles in the plane. In our experiments, the new algorithm, which we call the Disjoint-Cover (DC) algorithm, performs significantly better than the standard randomized incremental construction (RIC) of the union. Our algorithm is rather hard to analyze rigorously, but we provide an initial such analysis, which yields an upper bound on its performance that is expressed in terms of the expected cost of the RIC algorithm. Our approach and analysis generalize verbatim to the construction of the union of other objects in the plane, and, with slight modifications, to three dimensions. We present experiments with a software implementation of our algorithm using the CGAL library of geometric algorithms.

1 Introduction

Computing the union of n triangles in the plane is a fundamental problem in computational geometry with many applications. For example, this problem arises in the construction of the forbidden portions of the configuration space in certain robot motion planning problems, and in hidden surface removal for visibility problems in three dimensions [9, 12].

*Work reported in this paper has been supported in part by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces), 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. Micha Sharir has also been supported by NSF Grants CCR-97-32101 and CCR-00-98246, and by a grant from the U.S.-Israeli Binational Science Foundation.

Computing the union, by constructing the *arrangement* of the triangles (namely, the subdivision of the plane into vertices, edges, and faces induced by the boundary segments of the triangles), may result in a solution which is too slow in practice. This is because it is likely that most vertices of the arrangement lie in the interior of the union, so computing them is wasteful. Naturally, one would like to have an output-sensitive algorithm. However, such an algorithm is unlikely to exist: Even the problem of deciding whether the union of a given set of triangles in the plane covers another given triangle is a *3SUM-hard* problem [6]. The best known solutions for problems from this family require $\Theta(n^2)$ time in the worst case, even though the size of the output may be only linear or even constant.

In what follows, we ignore the actual vertices of the triangles themselves (whose number is only $3n$), and use the term *vertices* (of the arrangement of the input triangles) to refer to intersection points between edges of the triangles.

1.1 Randomized Incremental Construction

We compare our new algorithm to a randomized incremental algorithm (RIC) for constructing the union, which is *quasi output sensitive*, and which is a variant of a similar algorithm due to Mulmuley [12] (a similar algorithm is also presented by Agarwal and Har-Peled [1]).

Given a set T of n triangles in the plane, the RIC algorithm computes their union as follows: We compute a random permutation $D := (\Delta_1, \ldots, \Delta_n)$ of T, and insert the triangles one at a time, in their order in D. In the i'th iteration, we compute the partial union $\bigcup_{j=1}^{i} \Delta_j$. This is accomplished by finding the intersection points of the boundary of the next triangle Δ_i with the boundary of the preceding union $\bigcup_{j<i} \Delta_j$, and by removing all features that lie inside the union $\bigcup_{j\leq i} \Delta_j$. For further details concerning possible implementations of these insertion steps, see [1, 12]. Our DC algorithm also computes the union incrementally, by inserting the triangles one at a time, and it differs from the RIC algorithm in the order in which the triangles are inserted. In our study we use a twofold approach to measuring the cost of the algorithms. Our first cost measure is the number of vertices that the algorithms generate (some of which may not appear on the boundary of the union), and the set of these vertices depends only on the insertion permutation D. This cost measure allows us to ignore details of the implementation of the algorithms. However, the actual expected cost of the algorithms (in the unit cost model) depends on the set of triangles in a more subtle way [1, 12]. We discuss these aspects when presenting the implementation details for the algorithms, where we use, and experiment with, a second measure for the cost of the algorithms. The justification to using the number of generated vertices as our main cost measure comes from our experimental observations, and is discussed in more details below.

The crucial parameter is thus the (expected) number of intersections between triangle boundaries created during this process. Define the *depth* $d(v)$ of a vertex v to be the number of triangles of T that contain v in their interior. Vertices at depth 0 are the vertices of the union, and they have to be constructed by any algorithm that computes the union. We are thus only interested in the *residual cost* of the algorithm, defined as the (expected) number of positive-depth vertices that the algorithm constructs.

Let $\mathcal{A}(T)$ denote the arrangement of T, and let L_i denote the number of vertices of $\mathcal{A}(T)$ that are intersections of triangle boundaries and have depth i, for $1 \le i \le n-2$ (as already mentioned, the vertices of the triangles themselves are ignored in the analysis). Then the expected number of vertices at positive depth constructed by the RIC algorithm is $\theta(\mathcal{A}(T)) = \sum_{i=1}^{n-2} \frac{2}{(i+1)(i+2)} \cdot L_i$; we refer to this sum as *Mulmuley's theta series* [12]. (Conforming to earlier notation [12], we also denote this sum by $\theta_0(\mathcal{A}(T))$.) The factor $\frac{2}{(i+1)(i+2)}$ expresses the probability that a vertex v having depth i will be constructed; indeed, v is constructed if and only if the two triangles that create it appear in D before the i triangles that cover it.

1.2 Related Work

Agarwal and Har-Peled gave a randomized incremental algorithm for constructing the union of n triangles in the plane, whose analysis is based on Mulmuley's theta series [1]. The expected cost of their algorithm is proportional to $\sum_{i=1}^{n-2} \frac{L_i}{(i+1)}$, which we denote by $\theta_1(\mathcal{A}(T))$. An earlier variant, due to Mulmuley [12], constructs partial unions for hidden surface removal, with the same asymptotic bound on the expected running time. If the given triangles are *fat* (every angle of each triangle is at least some constant positive angle), or arise in the union of Minkowski sums of a fixed convex polygon with a set of pairwise disjoint convex polygons (which is the problem one faces in translational motion planning of a convex polygon), then their union has only linear or near-linear complexity [8, 10], and more efficient algorithms, based on either deterministic divide-and-conquer, or on randomized incremental construction, can be devised, and are presented in the above-cited papers.

1.3 Our Results

We present an incremental algorithm for constructing the boundary of the union. The algorithm, which we call the *Disjoint Cover* (DC) algorithm, inserts the triangles one by one in some order. Each insertion is performed exactly as in the RIC algorithm. The difference is in the order in which we process the triangles. The intuition behind our approach is that the random order used in the RIC construction makes sure that deep vertices of the arrangement are very unlikely to be constructed; however, shallow vertices have rather high probability of being created. A typical bad situation is when there exist triangles that cover many shallow vertices. If we could force these triangles to be inserted first, they would have eliminated many vertices that will be constructed under a random insertion order. This is exactly what the new algorithm is trying to achieve.

In Section 2 we present our algorithm and state a theoretical upper bound that expresses the residual cost of our algorithm in terms of the residual cost of the RIC, and also present a lower bound that shows, in certain rather pathological situations, the DC algorithm may perform more poorly than the RIC algorithm. Section 3 describes experimental results that compare the number of positive-depth vertices constructed by our algorithm and by the RIC algorithm, showing our algorithm performs significantly better in practice. Section 4 describes our implementation for the DC and the RIC algorithms, and presents experimental results concerning the actual running times of our algorithms. In this section we also define a new estimator, based on our implementation, for measuring the performance of each of the two algorithms. We give concluding remarks and suggestions for further research in Section 5.

2 The Disjoint Cover Algorithm

2.1 Description of the Algorithm

Define the *weight* $w(v)$ of a vertex v (at positive depth) to be $\frac{1}{d(v)}$. We denote by V^+ the set of vertices of the arrangement $\mathcal{A}(T)$ at positive depth (considering, as above, only intersection points of the triangle boundaries). Suppose that the insertion order of the DC algorithm (to be described shortly) is $(\Delta_1, \ldots, \Delta_n)$. Regard the triangles of T as open triangles. Define $S_{\Delta_j} = V^+ \cap \left(\Delta_j \setminus \bigcup_{i<j} \Delta_i\right)$, namely, the set of vertices in the interior of Δ_j that are not covered by the interior of any previously inserted triangle. The *weight* $W(\Delta_j)$, for $j = 1, \ldots, n$, is then defined to be the sum of the weights of the vertices in S_{Δ_j}. Note that $\{S_\Delta\}_\Delta$ is a partition of V^+ into pairwise disjoint sets.

The DC algorithm chooses an insertion order that aims to maximize the sequence $(W(\Delta_1), \ldots, W(\Delta_n))$ in lexicographical order. In an ideal setting (which is too expensive to implement, and which will therefore be modified shortly), we proceed as follows. Suppose we have already chosen $(\Delta_1, \ldots, \Delta_j)$ to be inserted. For each remaining triangle Δ, we set (temporarily) S_Δ to be the set of all vertices of V^+ in the interior of Δ that are not covered by $\bigcup_{i \le j} \Delta_i$. We compute the corresponding weights $W(\Delta)$ of all the remaining Δ's, and set Δ_{j+1} to be the triangle with the maximum weight. We proceed in this manner until all triangles have been chosen.

The problem with this approach is that it requires knowledge of all the vertices of $\mathcal{A}(T)$, which in general is too expensive to compute. Instead, we consider a smaller subset R. We fix some parameter r, select r random pairs of triangles from T, construct and collect the intersection points, if any, of the boundaries of each pair. We now estimate each set S_Δ by the corresponding set $S_\Delta \cap R$, which is computed using only the vertices in R, and consequently estimate $W(\Delta)$ by the sum of weights of vertices in $S_\Delta \cap R$. At present, this simplification should be viewed as purely heuristic—the theory of random sampling and ε-approximations (see, e.g., [13]) is not directly applicable to argue that $S_\Delta \cap R$ is a good approximation of S_Δ, because the portion of the plane over which S_Δ is estimated at the $(j+1)$-st step, namely, $\Delta \setminus \bigcup_{i \le j} \Delta_i$, may not have constant complexity, which is a (sufficient) condition that is usually needed to be assumed in order to facilitate the application of the random sampling theory. We hope and plan to set this heuristic on solid theoretical footing. Nevertheless, our experimental results indicate this heuristic performs very well in practice—see Section 3.

In order to compute the insertion order of the DC algorithm efficiently, we maintain the points of R in a list $\mathcal{L}$. We also keep for each triangle in T the list of points of R contained in it together with cross pointers between the points in this list and the corresponding points in $\mathcal{L}$. We omit the straightforward details of how we update these lists and weights of the triangles, and summarize the cost of this part of the DC algorithm in the following lemma.

Lemma 1 *Given a set T of n triangles and a vertex set R as above, the construction of the insertion order by the DC algorithm takes $O(n|R|)$ time.*

2.2 An Upper Bound

The following theorem relates the residual cost of the DC algorithm (in its ideal setting) to that of the RIC algorithm:

Theorem 1 *Let T be a collection of n triangles with κ intersection points at positive depth. Then the number of positive-depth vertices generated by the ideal DC algorithm is at most $O(n^{2/3}\kappa^{1/3}M^{1/3})$, where M is the expected number of positive-depth vertices generated by the RIC algorithm.*

Proof: Let $V_0 = V^+$ denote the set of vertices of $\mathcal{A}(T)$ at positive depth. Recall that the *depth* $d(v)$ of a vertex v is the number of triangles of T that contain v in their interior, and the *weight* $w(v)$ of a vertex v (at positive depth) is defined to be $1/d(v)$. Put $\kappa = |V_0|$, and

$$D_0 = \sum_{v \in V_0} d(v)w(v) = \kappa.$$

We can rewrite D_0 as (recall that the triangles are assumed to be open)

$$D_0 = \sum_{\Delta \in T}\sum_{v \in \Delta} w(v) = \sum_{\Delta \in T} W(\Delta).$$

Hence, there exists a triangle whose weight is at least $D_0/n = \kappa/n$. Thus, the weight W_1 of the first triangle Δ_1 that is inserted by the DC algorithm satisfies

$$W_1 \geq \frac{D_0}{n}.$$

Erasing the vertices in the interior of Δ_1 and discarding Δ_1 from T, we note that the depth of any remaining vertex has not changed. Denoting by V_1 the set of remaining vertices, and putting $\kappa_1 = |V_1|$, we have

$$D_1 = \sum_{v \in V_1} d(v)w(v) = \kappa_1.$$

Repeating the above argument, we see that the weight W_2 of the second triangle to be inserted by the DC algorithm satisfies

$$W_2 \geq \frac{D_1}{n-1},$$

and, erasing the vertices that Δ_2 covers in its interior, we are left with a subset V_2 of vertices, of size κ_2, that satisfies

$$D_2 \equiv \sum_{v \in V_2} d(v)w(v) = \kappa_2.$$

We keep iterating this analysis step. At the j-th step, the weight W_j of the j-th triangle Δ_j that the DC algorithm inserts satisfies

$$W_j \geq \frac{D_{j-1}}{n-j+1},$$

where $D_{j-1} = \kappa_{j-1}$ is the number of vertices that have not been covered by the first $j-1$ triangles.

We continue the process as long as $W_j \geq \frac{D_0}{nt}$, for some parameter t that we will fix shortly. Suppose that we stop after inserting q triangles. This means that, at the $(q+1)$-st step, we have

$$\frac{D_q}{n-q} \leq W_{q+1} < \frac{D_0}{nt},$$

implying that

$$\kappa_q = D_q < \frac{(n-q)D_0}{nt} < \frac{D_0}{t} = \frac{\kappa}{t}.$$

In other words, the number of vertices of $\mathcal{A}(T)$ at positive depth that have not been covered by the first q triangles inserted by the DC algorithm is at most κ/t. As the algorithm continues from this point, it may, in the worst case, generate all these vertices, and we make no attempt to analyze its performance, from this point on, in any finer manner.

In addition, the first q triangles that the algorithm inserts can generate, among themselves, at most $6q(q-1)/2 < 3q^2$ vertices. Our next goal is to estimate q. We have

$$W^* \equiv \sum_{i=1}^{q} W_i = \sum_{v \in V_0 \setminus V_{q+1}} \frac{1}{d(v)},$$

and since $W_1 \geq W_2 \geq \cdots \geq W_q$, we have $qW_q \leq W^*$. On the other hand, by assumption, $W_q \geq D_0/(nt) = \kappa/(nt)$. Hence,

$$q \leq \frac{W^* nt}{\kappa}.$$

Using the Cauchy-Schwarz inequality, and letting $V^* = V_0 \setminus V_{q+1}$ denote the set of positive-depth vertices covered by the first q triangles, we have

$$q^2 \leq \left(\frac{nt}{\kappa}\right)^2 \cdot \left(\sum_{v \in V^*} \frac{1}{d(v)}\right)^2 \leq \tag{1}$$

$$\frac{n^2t^2}{\kappa^2} \cdot \left(\sum_{v \in V^*} \frac{1}{d^2(v)} \right) \cdot \kappa \le \frac{n^2t^2}{\kappa} \cdot \left(\sum_{v \in V_0} \frac{1}{d^2(v)} \right) =$$

$$O(\frac{n^2t^2}{\kappa} \cdot M).$$

In other words, the (residual) cost of the DC algorithm, that is, the number of vertices at positive depth that it generates, is at most

$$O(\frac{n^2t^2M}{\kappa} + \frac{\kappa}{t}).$$

Choose

$$t = \frac{K^{2/3}}{n^{2/3}M^{1/3}},$$

to obtain that the cost of the DC algorithm is at most $O(n^{2/3}\kappa^{1/3}M^{1/3})$. This completes the proof of the theorem. □

Note that when M and κ are $\Theta(n^2)$, both algorithms generate the same quadratic number of vertices asymptotically. If either of these two parameters is strictly subquadratic, then the DC algorithm will produce a strictly subquadratic number of vertices at positive depth. However, the upper bound of Theorem 1 seems rather pessimistic, and we believe it can be improved (as is strongly suggested by our experimental results).

2.3 A Lower Bound

We next observe that there exist (rather pathological) examples in which $\kappa \ll n^2$ and the DC algorithm performs considerably worse than the RIC algorithm. This lower bound is exemplified in Theorem 2. No such examples are known for $\kappa = \Theta(n^2)$, and we conjecture that in this case the residual cost of the DC algorithm is at worst comparable with that of the RIC (and is likely to be smaller in practice).

Theorem 2 *Let n be an arbitrarily large integer, and let κ be any integer satisfying $cn^{3/2} \le \kappa \le n^2/c$, where c is a sufficiently large constant. There exists a collection T of n triangles with κ intersections between their edges at positive depth, so that the expected number of positive-depth vertices generated by the RIC algorithm is $O(\max\{n, \kappa^3/n^4\})$, and the number of positive-depth vertices generated by the DC algorithm is $\Omega(\kappa^2/n^2)$. The ratio between the latter and the former bounds is maximized when $\kappa = n^{5/3}$, in which case the bounds are $O(n)$ and $\Omega(n^{4/3})$, respectively.*

Proof: The lower bound construction is depicted in Figure 1.

The triangles in the figure are arranged as follows. We use two parameters k and t, such that $k \ll n^{1/2} \ll t$ and $kt = \Theta(n)$. We also put $a = \beta t$, where $\beta < 1$ is a sufficiently small positive constant (the various constants of proportionality will be determined later). There are $2a$ triangles of type A, a of which are vertical (with bases

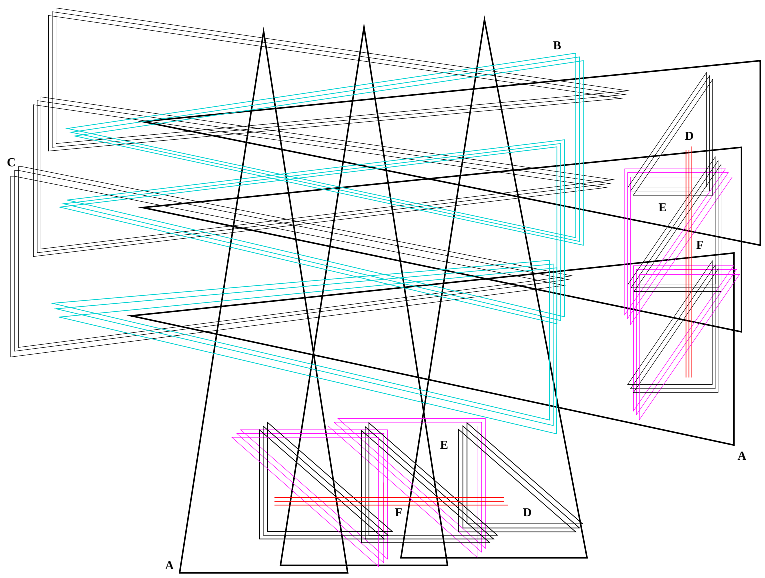


Figure 1: The construction in Theorem 2. The holes of the grid created by the triangles of type A are covered by the horizontal triangles of types B and C. The triangles of types D, E and F appear within the bottom portions of the vertical triangles of type A, and a copy of their configuration rotated by 90 degrees, also appears within the right portions of the horizontal triangles of type A.

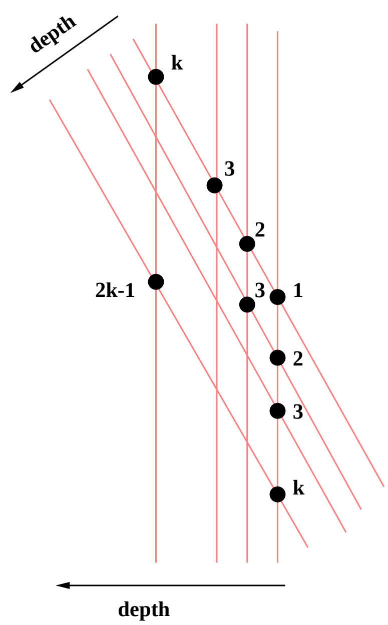


Figure 2: Two stacks of triangles of types D or E forming a grid. The labels of the intersection points denote their depth.

at the bottom) and a horizontal (with bases on the right). We also have ak horizontal triangles of type B, which are arranged in a stacks, each consisting of k triangles, obtained by very small translations of one of them, in the manner shown in the figure. Similarly, there are ak horizontal oppositely-oriented triangles of type C, which are also arranged in a similar stacks, each consisting of k triangles. In addition, there are two sets of triangles of types D, E, and F, where one set is arranged within the bottom portions of the vertical A-triangles, and the second set is a copy of the first set, rotated by 90° and shifted to be placed within the right portions of the horizontal A-triangles. Specifically, the bottom D-, E- and F-triangles are arranged as follows. There are ak triangles of type D and ak triangles of type E; the triangles of each type are arranged in a stacks of k triangles each, and the horizontal shifts between the stacks are the same as those between the vertical A-triangles. The stacks are arranged in the manner shown in the figure. Finally, we have t long and skinny horizontal triangles of type F, each shown in the figure as a horizontal line segment.

The total number of triangles is

$$n = 2a + 2ak + 4ak + 2t = 2t(1 + \beta + 6\beta k),$$

which is the precise relationship between t and k.

The idea of the proof is to show that, with an appropriate choice of k, t, and β, the triangles with the largest weight are essentially the A-triangles, and that this property persists after inserting many A-triangles. This will cause the DC algorithm to insert the A-triangles before any other triangle, thereby creating partial unions whose complexity is quadratic in the number of A-triangles, that is, $\Omega(t^2)$. On the other hand, the RIC algorithm treats the triangles in a more uniform manner, and consequently does better. The analysis of the RIC algorithm is simpler, and follows from routine estimation of the corresponding theta-series.

A-triangles. We first derive a lower bound for the weight of the A-triangles. The following analysis holds for all A-triangles except for the leftmost and rightmost vertical triangles and for the lowest and highest horizontal triangles. Let τ be any non-extreme vertical A-triangle (the case of horizontal triangles is fully symmetric). For our lower bound, we consider only the intersection points that lie inside τ along the edges of the F-triangles. As we traverse such an edge e from left to right within τ, we pass through five stacks of edges of triangles of types D and E. As follows from the illustration in the figure, the depth of the moving point starts at k, increases to $2k-1$, decreases back to k, increases once again to $2k-1$, decreases back to k, and finally increases back to $2k-1$. It follows that the total weight of all the vertices encountered along a single F-edge is

$$5\left(\frac{1}{k} + \cdots + \frac{1}{2k-1}\right) \geq 5\int_k^{2k} \frac{dx}{x} = 5\ln 2.$$

Each F-triangle has two edges that are traced this way, so the total weight of an A-triangle (except for the extreme ones) is at least

$$2t \cdot 5\ln 2 = 10t\ln 2.$$

Notice that this lower bound on the weight of an A-triangle τ continues to hold also after other A-triangles (or B- or C-triangles) have been inserted, as long as we have not yet inserted any of the two neighboring A-triangles preceding and succeeding τ in its sequence (of vertical or horizontal A-triangles).

D- and E-triangles. The weight of a D-triangle or an E-triangle is estimated in a similar manner, except that now we seek an *upper bound* on the weight. Let τ be a D-triangle (the case of an E-triangle is handled in a fully analogous manner). The vertices inside τ fall into the following categories:

(i) Intersections between edges of F-triangles and edges of D- and E-triangles. There are four stacks of these vertices along each F-edge, two 'inner' ones involving E-edges and two 'outer' ones involving D-edges. Each inner stack consists of k vertices, whose depth varies between k and $2k-1$. The total weight of these vertices, over all F-edges is thus at most

$$2t \cdot 2 \int_{k-1}^{2k-1} \frac{dx}{x} = 4t \ln \frac{2k-1}{k-1} < 4t \ln(2+\varepsilon_0),$$

for any $\varepsilon_0 > 0$, provided that k is sufficiently large.

Concerning the outer stacks, suppose that τ is the j-th triangle from the left in its stack of D-triangles. Then the left outer stack along the F-edge consists of $k-j$ vertices, whose depth varies between $k+j$ and $2k-1$, and the right outer stack consists of $j-1$ vertices, whose depth varies between $2k-1$ and $2k-j+1$. Arguing as above, the total weight of these vertices, over all F-edges, is at most

$$2t \cdot \left[\int_{k+j-1}^{2k-1} \frac{dx}{x} + \int_{2k-j}^{2k-1} \frac{dx}{x} \right] = 2t \ln \frac{(2k-1)^2}{(k+j-1)(2k-j)}.$$

This expression is maximized when $j = 1$ or $j = k$, yielding a bound of at most

$$2t \ln \frac{2k-1}{k} < 2t \ln 2.$$

Hence, the total weight of the vertices in the present category is at most $6t \ln(2+\varepsilon_0)$, for any $\varepsilon_0 > 0$.

(ii) Intersections between edges of D- and E-triangles and edges of other D- and E-triangles. These points form a constant number of (slanted) grids within τ, each of size at most $k \times k$, so that the depth of a point keeps increasing by 1 as we trace any row or column of the grid, and the minimum depth in such a grid is at least 1. It follows that the number of points in such a grid at depth i is at most i, implying that the total weight of all the points in the grid is $O(k)$. Hence, the total weight of vertices in the present category is $O(k)$.

(iii) Intersections between edges of A-triangles and edges of D- and E-triangles. There are only $O(k)$ such points, so their total weight is at most $O(k)$.

(iv) Intersections between edges of A-triangles and edges of F-triangles. There are $O(t)$ such points, but the depth of each of them is at least k, implying that their total weight is at most $O(t/k)$.

To recap, we have shown that the weight of a D-triangle or of an E-triangle is at most $6t \ln(2 + \varepsilon_0) + O(k + t/k)$, which, with an appropriate choice of parameters, is smaller than the lower bound on the weights of the A-triangles.

B- and C-triangles. Next we estimate the weight of the B- and C-triangles. Let τ be a B-triangle (C-triangles are handled in a fully analogous manner). There are three types of vertices that lie inside τ:

(i) Intersections between edges of B- and C-triangles and edges of other B- and C-triangles. These vertices are arranged in a constant number of grids, where each grid is of size at most $k \times k$. Arguing as in the analysis of D-triangles, the total weight of these vertices is at most $O(k)$.

(ii) Intersections between edges of A-triangles and edges of B- and C-triangles. There are $O(ak)$ such vertices, and along each A-edge they are arranged in a constant number of stacks, each consisting of k vertices, whose depth increases monotonically. Except for the C-triangles in their own highest stack, and for B-triangles in their own lowest stack, the depth of any vertex in a stack along an A-edge is at least k, for a total weight of at most $O(1)$. Hence, assuming τ is not one of these extreme triangles, the overall weight of vertices in this category is $O(a)$.

(iii) Intersections between edges of A-triangles and edges of other A-triangles. There are only $O(a)$ such intersections within τ, and the depth of each of them is at least k. Hence, the total weight of these vertices is at most $O(a/k)$.

To recap, we have shown that the weight of a B-triangle or of a C-triangle, except for those extreme ones noted above, is at most $O(a + k)$, which, with an appropriate choice of parameters (especially the constant of proportionality β), is smaller than the lower bound on the weights of the A-triangles.

F-triangles. Finally, the weight of any F-triangle is 0, since it does not contain any vertices in its interior.

The performance of the DC algorithm. Consider now the behavior of the DC algorithm on this input. Except for the extreme B- and C-triangles, the algorithm will first insert $\Theta(a)$ horizontal and $\Theta(a)$ vertical A-triangles. In fact, at least $a/3$ A-triangles from each group will be inserted first: If, say, fewer than $a/3$ vertical A-triangles have been inserted, there must exist a vertical A-triangle τ, so that neither τ nor any of its two neighbors have been inserted; in this case the weight of τ is still at least $10t \ln 2$, so τ will be inserted before any of the lower-weight non-A triangles. This implies that the DC algorithm will create partial unions of complexity $\Omega(a^2) = \Omega(t^2)$.

The performance of the RIC algorithm. On the other hand, the expected cost of the RIC algorithm is smaller, and is bounded as follows:

(i) There are $16akt = O(kt^2) = O(nt)$ vertices along the F-edges. Each of these vertices is at depth at least k, so the expected number of such vertices that the RIC algorithm generates is

$$O(kt^2/k^2 = O(t^2/k) = O(t^3/n).$$

(ii) There are $\Theta(a^2) = \Theta(t^2)$ intersections between A-edges. Except for $O(a)$ of them, they all lie at depth at least k, so the expected number of such vertices that the RIC algorithm generates is

$$O(a + a^2/k^2) = O(t + t^2/k^2) = O(t + t^4/n^2).$$

(iii) There are $\Theta(a^2k) = O(kt^2) = O(nt)$ intersections between A-edges and B- and C-edges. Except for $O(ak)$ of them, they all lie at depth at least k, so the expected number of such vertices that the RIC algorithm generates is

$$O(ak + a^2k/k^2) = O(tk + t^2/k) = O(n + t^3/n).$$

(iv) There are $\Theta(ak^2) = \Theta(tk^2) = \Theta(n^2/t)$ intersections between B- and C-edges and other B- and C-edges, and between D- and E-edges and other D- and E-edges. These vertices are arranged along the edges in stacks of size $\leq k$, and the expected number of vertices in a single stack that are generated by the RIC algorithm is at most proportional to $\sum_{i=1}^{k} 1/i^2 = O(1)$. It follows that the expected number of generated vertices in this category is $\Theta(kt) = \Theta(n)$.

Altogether, the expected number of vertices that the RIC algorithm generates is thus $O(n + t^3/n)$.

Note that the number κ of positive-depth vertices of the arrangement of the triangles in the construction is dominated by the number of those vertices along the A-edges and the F-edges. Hence, $\kappa = \Theta(t^2k) = \Theta(nt)$. Since we require that $t \gg n^{1/2}$, the construction yields arrangements with $\kappa \gg n^{3/2}$. Note that κ cannot be chosen real close to the maximum value $\Theta(n^2)$, because that would require t to be close to n, and thus k to be too small a constant. This in turn could make the term $O(t/k)$ in the estimation of the weights of the D- and E-triangles too large, thereby preventing us from separating these weights from those of the A-triangles. Nevertheless, we can still choose κ to be as large as γn^2, for γ a constant, provided that we keep γ sufficiently small.

In summary, the construction yields a set of n triangles with κ vertices at positive depth, for any $n^{3/2} \ll \kappa \ll n^2$, so that the DC algorithm generates $\Omega(\kappa^2/n^2)$ of these vertices, while the RIC algorithm generates an expected number of $\Theta(n+\kappa^3/n^4)$ vertices, which is significantly smaller when $\kappa \ll n^2$. The analysis of the special case $\kappa = \Theta(n^{5/3})$ is straightforward. This completes the proof of Theorem 2. $\square$

Note that Theorem 1 yields in this case the bound

$$O(n^{2/3}\kappa^{1/3}(\kappa^3/n^4)^{1/3}) = O(\kappa^{4/3}/n^{2/3})$$

on the number of positive-depth vertices generated by the DC algorithm. This bound gets closer to the lower bound $\Omega(\kappa^2/n^2)$ of Theorem 2 as κ gets closer to n^2.

3 Experimental Results I: Number of Positive Depth Vertices

In this section we present experimental results comparing the number of positive-depth vertices constructed by the RIC and by the DC algorithms. We start by describing the input sets and then display the results and comment on them.

The motivation to devise the DC algorithm is practical. We wish to precede the incremental construction of the union with a simple and fast procedure that will speed up the more heavy-duty incremental stage. The incremental stage uses rather involved data structures for representing the topology of the partially constructed union and for searching in it. In comparison, the preprocessing stage of the DC algorithm, where we compute the order of insertion, uses very simple operations. The most expensive operation, since we use exact arithmetic (see below), is the construction of a vertex, which we do $O(|R|)$ time. The only other nontrivial operation is testing whether a vertex lies inside a triangle, which we do $O(n|R|)$ times. In practice, the best size of R is the subject of on-going investigation, which has to determine the optimal trade-off between the preprocessing cost and the degree of approximation of the true triangle weights (which may affect the insertion order). In our experiments the preprocessing time is negligible compared with the time of the incremental construction, even when $|R|$ is linear in the number of input triangles.

3.1 Input Sets

The input data that we used is depicted in Figures 3 and 4. It consists of the following sets:

- **regular** (Figure 3(a)): Arbitrary triangles (each generated from a random triple of vertices) randomly placed inside a square.
- **fat** (Figure 3(b)): Equilateral triangles of a fixed size randomly placed inside a square.
- **fat_with_grid** (Figure 3(c)): A grid-like pattern fully covered by many random fat triangles; half of the triangles form the grid and the other half are the fat triangles.
- **star-shaped_robot_and_obstacles** (Figure 4(a-b)): The triangulated Minkowski sums of a star-shaped robot and triangular obstacles. The resulting arrangement contains star-shaped sets, each such set intersects its adjacent sets in a superlinear number of points.
- **L-shaped_robot_and_obstacles** (Figure 4(c-d)): The triangulated Minkowski sums of an L-shaped robot and rectangular obstacles. A fifth of the rectangular obstacles are long and narrow rectangles, and the rest of them are small squares. The resulting arrangement contains L-shaped sets rotated by 180 degrees.

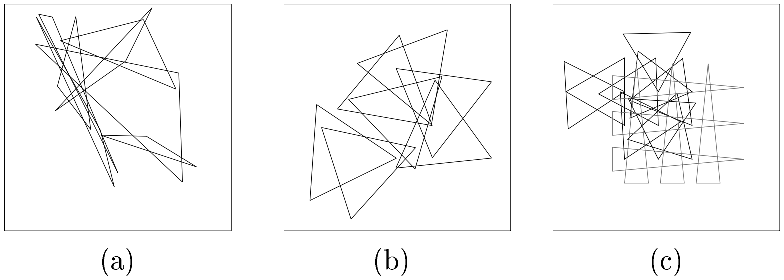


Figure 3: (a) Regular input, (b) Fat input and (c) Fat_with_grid input.

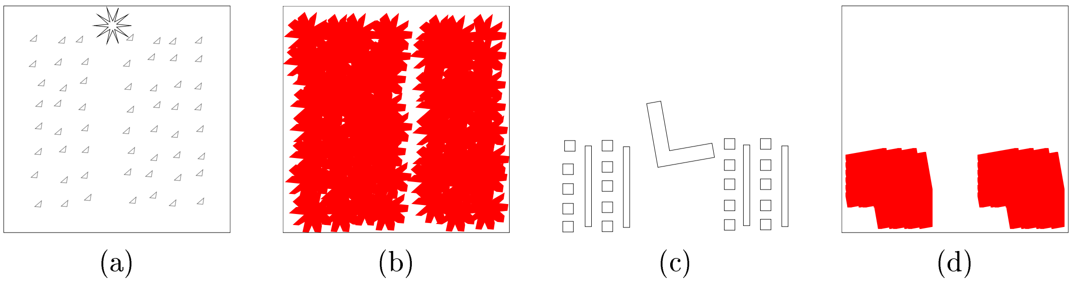


Figure 4: (a) A star-shape robot and triangular obstacles, and (b) the Minkowski sum of the robot (rotated by 180 degrees) with each of the obstacles. (c) An L-shape robot and rectangular obstacles, and (d) the Minkowski sum of the robot (rotated by 180 degrees) with each of the obstacles.

For each type of input, we have experimented with a varying number of triangles, up to 800 per run, except for the *star-shaped_robot_and_obstacles* and the *L-shaped_robot_and_obstacles* data sets, in which we used a fixed set of size 2832 and 840, respectively. For each specific input set, we ran each algorithm five times, and the results reported below are the averages over these running times. The complexity of the union of the *fat* input is almost linear in the number of triangles (see Figure 3), while the union of the *fat_with_grid* input can have a superlinear number of holes. The *star-shaped_robot_and_obstacles* input (resp. *L-shaped_robot_and_obstacles* input) constitute the configuration-space obstacle [9], arising in translational motion planning of a star-shaped (resp. L-shaped) polygon moving amid disjoint triangular (resp. rectangular) obstacles which are placed inside a square. See Figure 4. Using the Minkowski sums package developed on top of the CGAL library [4] (see below), we computed the Minkowski sum of the star-shaped (resp. L-shaped) robot with each of the obstacles and triangulated each resulting sum. Then we collected all such triangles to form our data set.

3.2 Results

We present experimental results of applying both the DC and the RIC algorithms to each of the data sets described in Section 3.1. We present the number of positive-depth vertices created by each of these algorithms, which, as discussed above, is our first yardstick for measuring and comparing the performance of the algorithms.

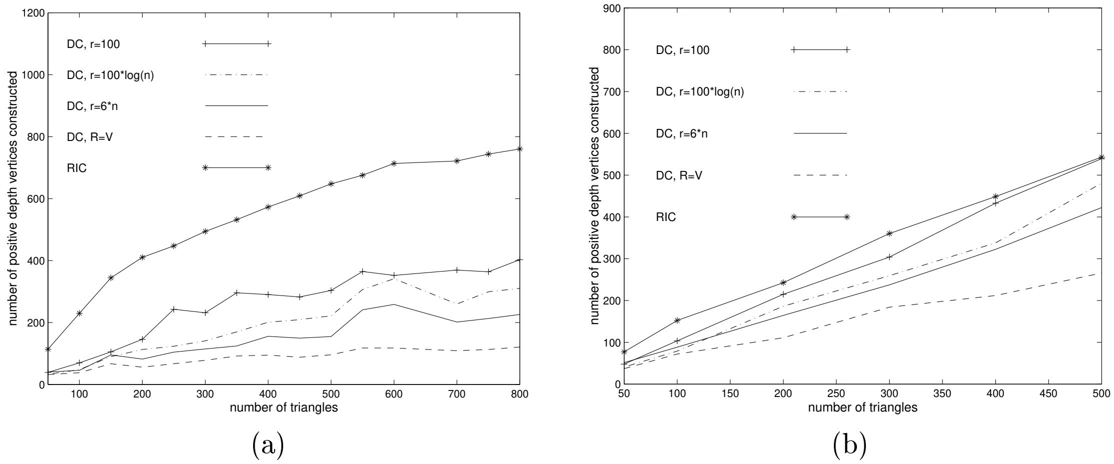


Figure 5: (a) Average number of positive-depth vertices created for the *regular* input, and (b) for the *fat* input; r denotes the number of pairs of triangles used to construct the sample R.

In all our experiments the DC algorithm performs better, and in several cases significantly better, than the RIC. As mentioned above, determining the right size of R in practice is a subject of on-going investigation. For each input type we show five graphs. Besides the graph for the RIC algorithm, we present graphs for the DC algorithm where the (maximal) size of R varies between a constant, a logarithmic term in the number of input triangles, a linear term, and R being the full set V. The results are presented in Figures 5 and 6 and in Table 1. Note that the results reported in this section are independent of implementation details; they only depend on the insertion order (permutation) determined by the DC and the RIC algorithms.

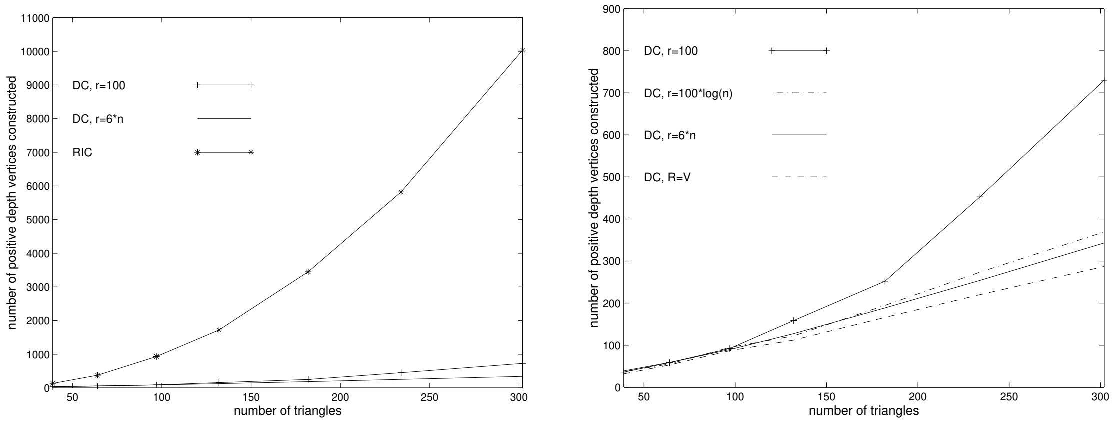
In all the graphs we see that if we take the whole set V into account in computing the insertion order then the savings in the union construction stage are big. In general, in all our experiments, the DC algorithm performs better than the RIC,[1] and the performance improves as the size of R increases. In some cases, e.g., for the *fat_with_grid* input, even if we use much smaller samples R, e.g., samples of size linear in the input size, then we save the construction of over 9700 vertices (out of about 10040) during the incremental stage when the input consists of 302 triangles (Figure 6).

For the *fat* input (see Figure 5(b)), there is still improvement, but it is less significant than the improvement obtained for the other input sets. It can be shown that the amount of work that the RIC algorithm performs for fat triangles is always close to linear.[2] Hence, improving such small costs is more difficult (and less of an issue) than improving the costs for regular triangles.

For the *fat_with_grid* input (Figure 6) the RIC algorithm performs poorly since the intersection points of the grid tend to be shallow on the average.

[1] Recall that this may fail to hold in some pathological examples, where $|V| \ll n^2$, as in Theorem 2.

[2] The proof is a routine exercise in random sampling, and is therefore omitted; it follows from the fact that the union of any subset of the triangles has near-linear complexity.



(a) (b)

Figure 6: Average number of positive-depth vertices created for the *fat_with_grid* input. Since the differences between the number of positive-depth vertices constructed by the DC algorithm and by the RIC algorithm are highly significant for this input, we compare in (a) the RIC algorithm to the DC algorithm only for $r = 100$ and $r = 6n$, and in (b) we zoom in on the number of positive-depth vertices constructed by the DC algorithm for all different random sample sizes.

algorithm	**number of positive-depth vertices created, n=2832**
DC, $R = V$	328
DC, $r = 6n$	943
DC, $r = 100 \cdot \log n$	1379
DC, $r = 100$	1880
RIC	2037.92

algorithm	**number of positive-depth vertices created, n=840**
DC, $R = V$	303
DC, $r = 6n$	554.5
DC, $r = 100 \cdot \log n$	654.5
DC, $r = 100$	1070.34
RIC	1423.74

Table 1: The average number of positive-depth vertices constructed for the *star-shaped_robot_and_obstacles* input (left) and for the *L-shaped_robot_and_obstacles* input (right).

For the *star-shaped_robot_and_obstacles* and the *L-shaped_robot_and_obstacles* inputs (Table 1), we get an improvement comparable with that obtained for the *regular* input. For the *star-shaped_robot_and_obstacles* input, notice that since every star-shaped set intersects its adjacent sets in a superlinear overall number of vertices, most of the arrangement vertices will not be shallow, and hence these vertices are less likely to be generated by the RIC algorithm. For the *L-shaped_robot_and_obstacles* input (Table 1 (right)), the saving in the number of positive-depth vertices generated by the DC algorithm is significant even when R is much smaller than V. This is because the triangles created by the long and narrow rectangular obstacles cover most of the vertices of the arrangement induced by the Minkowski sums of the robot (rotated by 180 degrees) and the obstacles. Hence, even when choosing a random subset smaller than V, the DC algorithm tends to insert first most of these triangles into the union with high likelihood.

We remark that the number of vertices $|V|$ in some of our examples is huge (reaching roughly half a million for the *regular* input with 800 triangles), rendering the construction of the union by first computing the entire underlying arrangement unacceptable (when using exact arithmetic). Hence, our experimenting with $R = V$ is not intended (for such inputs) as a viable implementation, and is used only for measuring and calibrating performance.

4 Experimental Results II: Running Times

In this section we present our implementation and experimental results comparing the running times of the RIC and the DC algorithms. We use the same input sets as in Section 3.1.

In the experiments reported in Section 3 we focused on the number of positive-depth vertices constructed by the RIC and the DC algorithms. This number is a pure estimator in the sense that it is an (expected) lower bound on the time complexity of any implementation of the DC or RIC algorithms, independent of the actual implementation. However, any implementation requires additional operations for constructing the union boundary. Such operations are needed for updating the structure representing the union during the construction. For example, the RIC algorithm can be implemented using trapezoidal decomposition of the complement of the union constructed so far, and corresponding conflict lists between trapezoids and crossing triangles [1, 12]. The expected running time of this implementation is $O(n \log n + \theta_1(\mathcal{A}(T)))$, where $\theta_1(\mathcal{A}(T)) = \sum_{i=0}^{n-2} \frac{1}{(i+1)} \cdot L_i$. The function $\theta_1(\cdot)$ expresses the expected overall number of conflicts over a random permutation $D := (\Delta_1, \ldots, \Delta_n)$ of the input triangles [1]. Exactly the same implementation can be applied to the DC algorithm.

In our implementation, which is more practically oriented, we do not maintain trapezoidal decompositions (although this is implemented in the CGAL library), due to the large overhead in their maintenance. Instead, we maintain the union and its complement as a collection of undecomposed faces. In this representation, the additional operations that are needed are point location of vertices of newly-inserted triangles, and traversal of their edges through the current union. In this section, we introduce another estimator for the running time of our algorithms, which is based on the total cost of the edge traversals—see below. Our experimental results compare this new estimator for each run of the DC algorithm and the RIC algorithm, and show the connection between this value and the actual running time of our algorithms.

4.1 Implementation Details

Our implementation of the union algorithms is based on the CGAL (version 2.3) and LEDA (version 4.3) libraries. Our package works with Linux (gcc 2.95 compiler). The tests were performed on a Pentium-III PC machine having two processors, 1GHz each, with 2Gb RAM memory. The implementation employs the CGAL maps and arrangements packages [5, 7], and uses *exact arithmetic*. We use LEDA's rational kernel which employs a floating point filter in computing predicates [11]. Note however that the construction of new vertices does not benefit from filters and the coordinates of intersection points are computed to unlimited precision (namely, with as much rational precision as required).

More specifically, the DC and RIC algorithms use the CGAL *Planar_map_with_intersections* class. The union constructed by each of the two algorithms is stored in a Doubly Connected Edge List (DCEL for short) [2]. Every insertion of a triangle into the partially constructed union is performed by the insertion of the three edges defining the triangle, one at a time. Each insertion of an edge e into the DCEL is done in the following manner: First we locate one of the endpoints of e in the DCEL. The point-location operation is performed by "walking" backwards from infinity along the zone of a vertical ray emanating from the query point. The walk starts at the unbounded face and progresses towards the query point, as described in [5]. Next, we find the intersection points of e with the boundary of the current union, by traversing its zone in the union, in the same manner as in the initial ray tracing step. Each time we discover an intersection along e, we create a new vertex in the DCEL and insert into the structure the portions of e that do not lie inside the present union (each such portion is delimited by a pair of consecutive intersection points). Note that, in practice, only one point location per each new triangle is required.

After the new triangle has been inserted, we remove all features that lie inside the union, and have not been previously removed. The removal stage is performed by traversing all edges in the current structure in a depth-first manner starting at the faces incident to the edges of the newly inserted triangle, and checking whether each such edge lies inside the union. In practice, the time consumed by traversing all edges in the clean-up stage is negligible, since we do not perform any heavy-duty geometrical operations when checking whether an edge lies inside the union.

In the RIC algorithm, we first randomly permute all the triangles in the input set and then construct the union boundary incrementally by adding one triangle at a time, as described above.

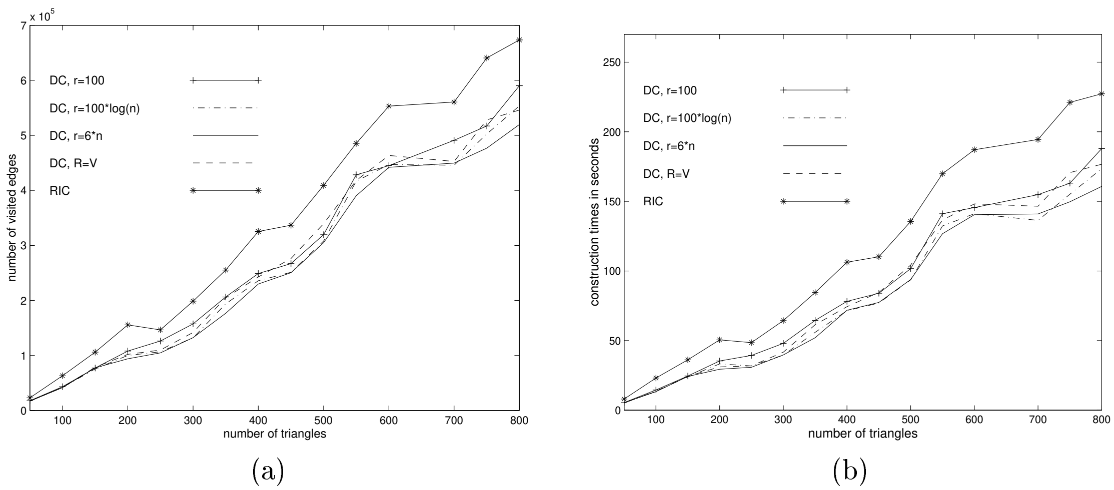


Figure 7: (a) The (average) total number of traversed edges for the *regular* input, and (b) the running times for constructing the union, excluding preprocessing time of the DC algorithm.

The preprocessing stage of the DC algorithm that produces the ordering of the triangles constructs and uses a random subset of the vertices of the underlying arrangement of the triangles, in the manner explained in Section 2. After obtaining the insertion order with respect to the above random subset, the algorithm is implemented in the same way as the RIC algorithm, as just outlined. We emphasize that we have not made at this stage a serious attempt to optimize our implementation, which can definitely be improved with more care. Nevertheless, since we apply the same implementation to both the DC and the RIC algorithms, the relative running times, as well as the edges traversal estimators, provide a good comparison between the two algorithms.

4.2 Results

In our implementation, the running time of the algorithm is dominated by the total number of edges traversed when inserting the triangles into the union constructed so far. In the experiments reported in this section, we measure the total number of visited edges by each of the two algorithms. We also compare running times and show that there is a high correlation between the number of traversed edges and the time undertaken for constructing the union by each of the two algorithms.

The results are presented in Figures 7 through 9 and in Tables 3 and 4.

In all graphs we see that the DC algorithm performs better than the RIC algorithm. However, (i) the improvement is not as significant as in the number of generated positive-depth vertices, and (ii) the performance does not significantly improve as the size of the sample R increases. The explanation to these phenomena is: First, both algorithms need to construct the vertices of the union, and this tends to partially hide the savings due to constructing fewer positive-depth vertices. Second, the edge traversal cost may be high

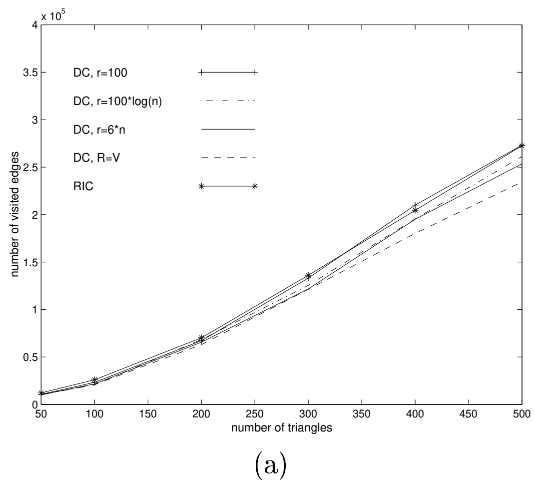


Figure 8: (a) The (average) total number of traversed edges for the *fat* input, and (b) the running times for constructing the union. The running time includes the preprocessing time when using the DC algorithm.

when the combinatorial complexity of the visited faces (of the union and its complement) is large. For instance, for the *regular* input the smallest number of traversed edges is obtained when $r = 6n$ (Figure 7(a)). The total number of all traversed edges in this case, for 800 input triangles, is roughly 520000, compared with 674000 when running the RIC algorithm. Conceivably, this cost can be significantly reduced with more careful implementation.

Figure 7(b) displays the running times when constructing the union for the regular input sets. We did not count in this experiment the preprocessing time of the DC algorithm. With this omission, we have almost identical relative performance when compared with the number of traversed edges presented in Figure 7(a). This indicates that the running time of our algorithms (excluding preprocessing in the case of the DC algorithm), under the implementation described at Section 4.1, is dominated by the total number of traversed edges during the union construction. We got similar results for the other four data sets.

For the *fat* input (Figures 8), the DC algorithm only slightly improves the performance. Recall that the RIC algorithm has good performance on the *fat* input, and improving its performance is more difficult (and less feasible, and less important anyway) in this case. Notice that, even when including the preprocessing time of the DC algorithm (Figure 8(b)), the DC algorithm is still slightly faster than the RIC algorithm, excluding the case in which $R = V$, where the preprocessing time consumes a significant part of the running time (as noted already, this case is not intended as a real execution strategy).

For the *fat_with_grid* input (Figure 9) the DC algorithm performs significantly better than the RIC algorithm. In this case the RIC algorithm traverses faces with a large number of holes on average. The DC algorithm first inserts the fat triangles, and hence traverses faces with a smaller number of holes. Choosing a random sample of size

algorithm	number of total edges traversed, n=2832	union construction time (including preprocessing) in seconds
DC, $R = V$	2466699	2579.47
DC, $r = 6n$	2867058.67	824.55
DC, $r = 100 \cdot \log n$	3243469	941.88
DC, $r = 100$	3646946.5	1074.48
RIC	3696264.67	1088.02

Table 2: The (average) total number of traversed edges and union construction time (including preprocessing time) for the *star-shaped_robot_and_obstacles* input.

algorithm	number of total edges traversed, n=840	union construction time (including preprocessing) in seconds
DC, $R = V$	778700	359.06
DC, $r = 6n$	819588	207.35
DC, $r = 100 \cdot \log n$	850242	216.06
DC, $r = 100$	986724.67	253.89
RIC	1113375.2	294.01

Table 3: The (average) total number of traversed edges and union construction time (including preprocessing time) for the *L-shaped_robot_and_obstacles* input.

linear or logarithmic in the input size yields almost the same performance as choosing $R = V$ in this case. The explanation to this phenomenon is similar to that given in Section 3.2, where we got a similar number of positive-depth vertices constructed by the DC algorithm, for a random sample of linear size and for $R = V$.

For the *star-shaped_robot_and_obstacles* and the *L-shaped_robot_and_obstacles* inputs, the DC algorithm performs better than the RIC algorithm (see Tables 2 and 3). For the *L-shaped_robot_and_obstacles* input, for all random samples smaller than V, the improvement in the performance of the DC algorithm relative to the that of the RIC algorithm is slightly more significant when compared with the *star-shaped_robot_and_obstacles* input. The explanation to this behavior is similar to that given in Section 3.2. For both data sets the total number of traversed edges decreases as the size of the random sample grows, and the best running time is achieved when taking $r = 6n$.

In most of our experiments, the preprocessing time of the DC algorithm, when taking $R = V$, consumed most of the running time of the union construction (Table 4).

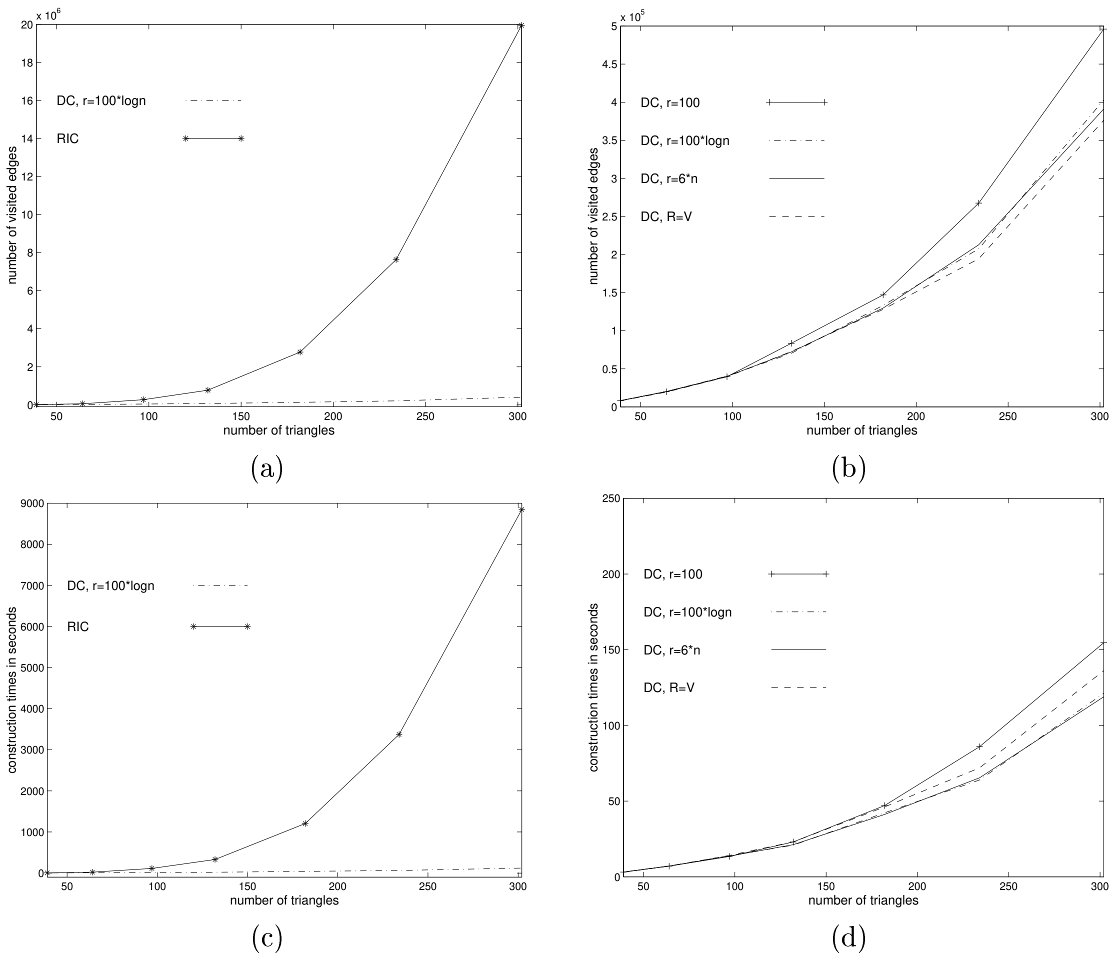


Figure 9: (a)-(b) The (average) total number of traversed edges for the *fat_with_grid* input, and (c)-(d) the running times, including preprocessing time. Since the differences between the performance of the DC algorithm and the RIC algorithm are highly significant for this input, the two left-hand side figures (a) and (c) compare the RIC algorithm to the DC algorithm only with $r = 100 \cdot \log n$. In the two right-hand side figures (b) and (d) we zoom in on the number of traversed edges and running times of the DC algorithm for all different random sample sizes.

Data set	$R = V$	$r = 6n$	$r = 100 \cdot \log n$	$r = 100$
regular, n=400	82.1825, 148.84	4.85, 67.32	2.18, 69.6	0.48, 70.93
fat, n=500	58.03, 118.71	2.52, 69.29	1.05, 70.12	0.29, 72.48
fat_with_grid, n=302	28.26, 136.15	2.38, 118.88	1.48, 121.16	0.36, 154.59
star_robot_with_obstacles, n=2832	1918.23, 2579.47	18.04, 824.55	5.02, 941.88	2.6, 1074.48
L_robot_with_obstacles, n=840	178.25, 359.06	5.01, 207.35	2.78, 216.06	0.48, 253.89

Table 4: The preprocessing time (left-hand side number in each cell) and the entire union construction time (including preprocessing, right-hand side number in each cell) for all five data sets, for the DC algorithm with the specified input sizes.

However, already by decreasing R to be linear in the number of triangles (even though the bound of Lemma 1 on the preprocessing cost is $O(n^2)$) we get a significant improvement in the preprocessing time. For 400 triangles from the *regular* input set, we have that, in all cases, excluding the case $R = V$, the ratio between the preprocessing time and the entire union construction time is less than 1:15. For the *fat*, *fat_with_grid*, *star-shaped_robot_and_obstacles* and *L-shaped_robot_and_obstacles* data sets, this ratio is roughly 1:28, 1:70, 1:45 and 1:40, respectively, when taking $r = 6n$.

Note that in all our experiments, when taking $r = 6n$, the total number of edges traversed by the DC algorithm is very close to that number when taking $R = V$.

5 Conclusions

The experiments reported above demonstrate the practical advantages of the DC algorithm relative to the RIC algorithm. Our results show that the number of positive-depth vertices constructed by the RIC algorithm is larger (and, in many cases, significantly larger) than the number of such vertices constructed by the DC algorithm, for all five kinds of input.

We note that the DC algorithm in its ideal setting (with $R = V$) is *deterministic*. In practice we use randomness, but only to estimate the weights of triangles. After doing so, the insertion order is still computed deterministically (although it is a random variable).

The DC algorithm can be generalized for other geometric objects in the plane, and also can be extended to higher dimensions. Since the calculation of the disjoint cover of each such object deals mostly with counting the number of vertices contained in the interior of that object, the order of insertion can be easily calculated if we are provided with suitable primitives for calculating intersection points and for testing for inclusion of points inside geometric objects. We are currently in the process applying our algorithm to the union of ellipses and of lenses formed by pairs of intersecting disks.

The simplest extension to three dimensions concerns the construction of the union of tetrahedra. If we concentrate on our first performance measure, the number of positive-depth vertices generated by the algorithms, it is easy to extend the DC algorithm and its analysis to this 3-dimensional setup. In particular, we provide in Appendix A an extension of Theorem 1 to three dimensions.

We note that after the original submission of this paper, we managed to construct a set of triangles whose arrangement contains $\Theta(n^2)$ vertices, and the residual cost of the DC and the RIC algorithms are $\Omega(n^{4/3})$ and $O(n)$, respectively. This example is a generalization of the lower bound construction presented in Section 2.3. Note that Theorem 1 yields the upper bound $O(n^{5/3})$ on the residual cost of the DC algorithm in this case, so there still exists a gap between this upper bound and the best known lower bound. Note also that this construction strengthens Theorem 2, since it does not establish any gap in the cost of the two algorithms when $\kappa = \Theta(n^2)$. For further research we propose to improve the upper bound on the residual cost of the DC algorithm, or alternatively, show a tighter lower bound.

The experimental results under our implementation show that the DC algorithm performs better than the RIC algorithm. The two sessions of experiments, in which the first measures only the number of generated positive-depth vertices, and the second measures the number of traversed edges and running times, point to the connection between these two performance parameters. In general, as the number of generated positive-depth vertices decreases, the performance improves. However, it is not guaranteed that fewer positive-depth vertices constructed by an algorithm always lead to a similar improvement in the performance of the algorithm. For instance, the experiments with the *regular* input set show a significant improvement in the number of positive-depth vertices constructed by the DC algorithm, compared with the RIC algorithm. However, there is a smaller improvement in the actual performance of the algorithm. This difference is explained by our implementation of the construction of the union, in which the cost of inserting a triangle is largely dominated by the number of edges traversed during this stage. Even when there are fewer positive-depth vertices, the performance may still suffer from having to traverse faces with large complexity. For further research, we propose to check whether this gap still exists when using an implementation with better theoretical performance, such as the one suggested in [1]. In addition, we note that, when comparing the running time of the algorithms, we measure the time undertaken for constructing vertices at depth 0 as well. These vertices are constructed by both algorithms, implying a potentially smaller difference between the performance of the two algorithms, compared with the difference obtained when measuring only positive-depth vertices constructed.

Nevertheless, through our experiments, the DC algorithm did not decrease performance relative to the RIC algorithm. In some cases, as the case of the *fat_with_grid* input, the improvement in the performance achieved by the DC algorithm was extremely significant. In other cases (the *regular* input and the two *robot_and_obstacles* inputs), the actual improvements were still significant, ranging from 20 to 30 percent. We also note that, in our experiments, the preprocessing time, undertaken for random samples that are logarithmic or even linear in the size of the input, is negligible relative to the entire union construction time, and the improvement in performance obtained in these cases was similar to the improvement we gained when running the DC algorithm in its ideal setting. This implies that, by a small amount of additional work (recall also that the preprocessing stage is very easy to implement), we obtain an algorithm that achieves better performance in practice than the RIC algorithm.

We note that the order of insertion of the triangles can be defined by other simpler estimators rather than the disjoint cover of each triangle. For example, one can define the weights of the triangles to be proportional to their area, and thus insert the larger triangles first, or alternatively, insert them in a random order where large triangles have high probability to be chosen earlier. It can easily be proved that such a heuristic may fail to beat the RIC algorithm. Even when sorting the triangles according to the number of vertices that they contain in their interior (namely, by their "full cover", rather than the disjoint cover that we have used), it can be shown that there are examples where this algorithm produces $\Omega(n^2)$ positive-depth vertices, whereas the RIC algorithm generates only an expected number of $O(n)$ positive-depth vertices. In addition, such algorithms are not guaranteed to work well in practice. For example, it was shown in [4] that inserting the triangles by their fatness (i.e., in decreasing order of their smallest angles) results in poor performance.

Finally, we have recently applied the DC algorithm to the problem of constructing efficiently (in subquadratic time) the union of n triangles when the union is determined by only a small (unknown) subset of the input triangles. In this application, we have enhanced the algorithm with more sophisticated routines that guarantee efficient worst-case performance, and have analyzed rigorously the process of sampling a subset R of the positive-depth vertices. This work is reported in [3].

References

[1] P. K. Agarwal and S. Har-Peled. Two randomized incremental algorithms for planar arrangements, with a twist. Manuscript, 2001.

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

[3] E. Ezra and M. Sharir. Output-sensitive construction of the union of triangles. Manuscript, 2003.

[4] E. Flato. Robust and efficient construction of planar Minkowski sums. Master's thesis, Dept. Comput. Sci., Tel-Aviv Univ., 2000. http://www.cs.tau.ac.il/~flato.

[5] E. Flato, D. Halperin, I. Hanniel, O. Nechushtan, and E. Ezra. The design and implementation of planar maps in CGAL. *The ACM Journal of Experimental Algorithmics*, 5, 2000. Also in LNCS Vol. 1668 (WAE '99), Springer, pp. 154–168.

[6] A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom. Theory Appl.*, 5:165–185, 1995.

[7] I. Hanniel and D. Halperin. Two-dimensional arrangements in CGAL and adaptive point location for parametric curves. In *Proc. of the 4th Workshop of Algorithm Engineering*, volume 1982 of *Lecture Notes Comput. Sci.*, pages 171–182. Springer-Verlag, 2000.

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

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

[10] J. Matoušek, J. Pach, M. Sharir, S. Sifrony, and E. Welzl. Fat triangles determine linearly many holes. *SIAM J. Comput.*, 23:154–169, 1994.

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

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

[13] J. Pach and P. K. Agarwal. *Combinatorial Geometry.* John Wiley & Sons, New York, NY, 1995.

A The DC Algorithm in Three Dimensions

The DC and the RIC algorithms can be extended to three dimensions, where the task is to construct the union of n simplices in $\mathbb{R}^3$. The extension of both algorithms is fairly easy, at least on the conceptual level. As in the 2-dimensional case, an initial measure of the performance of the algorithms, which is independent of any implementation details and which we continue to refer to as the *residual cost*, is in terms of the number of positive-depth vertices generated during the union construction. In this subsection we note that Theorem 1 can be extended to three dimensions, to yield a similar upper bound on the residual cost of the DC algorithm in terms of the residual cost of the RIC algorithm.

The residual cost M of the RIC algorithm is $O(\sum_{v \in V_0} \frac{1}{d^3(v)})$. The initial part of the analysis in Theorem 1 carries verbatim to the 3-dimensional case. In particular, the definitions of, and relationships between D_j and W_j remain the same. The first q simplices that the algorithm inserts can generate among themselves at most $20q(q-1)(q-2)/6 \leq 4q^3$ vertices.[3] Hence, using Hölder's inequality, the inequality (1) in the

[3] The constant 20 arises as a consequence of Euler's formula, applied to the intersection polytope of three simplices.

proof of Theorem 1 becomes:

$$q^3 \leq \left(\frac{nt}{\kappa}\right)^3 \cdot \left(\sum_{v \in V^*} \frac{1}{d(v)}\right)^3 \leq$$

$$\frac{n^3 t^3}{\kappa^3} \cdot \left(\sum_{v \in V^*} \frac{1}{d^3(v)}\right) \cdot \kappa^2 \leq \frac{n^3 t^3}{\kappa} \cdot \left(\sum_{v \in V} \frac{1}{d^3(v)}\right) =$$

$$O\left(\frac{n^3 t^3}{\kappa} \cdot M\right).$$

The residual cost of the DC algorithm is thus

$$O\left(\frac{n^3 t^3 M}{\kappa} + \frac{\kappa}{t}\right).$$

We choose

$$t = \frac{\kappa^{1/2}}{n^{3/4} M^{1/4}},$$

to obtain that the residual cost of the DC algorithm is at most $O(n^{3/4} \kappa^{1/2} M^{1/4})$.