Efficient Construction of the Union of Geometric Objects

Eti Ezra Dan Halperin Micha Sharir

School of Computer Science
Tel Aviv University
{estere,danha,micha}@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.

Computing the union, by constructing the arrangement 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 \( \text{SSUM-hard} \) problem [4]. 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.

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 [9] (a similar algorithm is also presented by Agarwal and Har-peled [7]).

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 by one, in their order in \( D \). In the \( i \)th iteration, we compute the partial union \( \bigcup_{j=1}^{i-1} \Delta_j \). This is accomplished by finding the intersection points of the boundary of the next triangle \( \Delta_i \) with the boundary of the preceding union \( \bigcup_{j=1}^{i-1} \Delta_j \) and by removing all features of the union that lie inside \( \Delta_i \). For further details concerning possible implementations of these insertion steps, see [1, 9]. In our study we ignore these details because the measure we use for the cost of the algorithms is the number of vertices that they generate (some of which may not appear on the boundary of the union), and the set of these vertices depends only on the random permutation \( D \). Note however that the actual expected cost of the algorithm (in the unit cost model) depends on this number in a more subtle way than we use here [1, 9]. The justification for this approach 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 \( A(T) \) denote the arrangement of \( T \), and let \( L \) denote the number of vertices of \( A(T) \) that are intersections of
triangle boundaries and have depth $i$, for $1 \leq i \leq n - 2$ (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 $\mathbb{E}(T) = \sum_{i=1}^{n-2} \frac{1}{2^i} L_i$; we refer to this sum as Mulumley's theta series. The factor $\frac{1}{2^i}$ 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 based on Mulumley's theta series [1]. 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, then their union has only linear or near-linear complexity [6, 7], 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. Section 3 describes experimental results that compare the performance of our algorithm and of the RIC, showing that our algorithm performs significantly better in practice.

2. CONSTRUCTING THE UNION: DISJOINT COVER 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 $(\Delta_1, \ldots, \Delta_n)$. Define $S_{\Delta_j} = V \setminus \bigcup_{i < j} \Delta_i$, namely, the set of vertices in the interior of $\Delta_j$ 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 $S_{\Delta_j}$. Note that $\{S_{\Delta_i}\}$ 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 later), we proceed as follows. Suppose we have already chosen $(\Delta_1, \ldots, \Delta_{j-1})$ to be inserted. For each remaining triangle $\Delta_j$ we set (temporarily) $S_{\Delta_j}$ to be the set of all vertices of $V$ in the interior of $\Delta$ that are not covered by $\bigcup_{i < j} \Delta_i$. We compute the corresponding weights $W(\Delta)$ of all the remaining $\Delta$'s, and set $\Delta_{j+1}$ 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 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 $S_{\Delta}$ 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 $\varepsilon$-approximations (see, e.g., [10]) is not directly applicable to argue that $S_{\Delta} \cap R$ is a good approximation of $S_{\Delta}$ because the portion of the plane over which $S_{\Delta}$ is estimated at the $j$-th step, namely, $\Delta \setminus \bigcup_{i < j} \Delta_i$, 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 that this heuristic performs very well in practice—see Section 3.

The above discussion is summarized in the following lemma. (Proofs are omitted in this abstract.)

**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 $O(n|R| \log n)$ time.

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 $\kappa$ intersection points at positive depth. Then the number of positive-depth vertices generated by the ideal DC algorithm is at most $O(n^{2/3} \kappa^{1/2} M^{1/3})$, where $M$ is the expected number of positive-depth vertices generated by the RIC algorithm.

Note that when $M$ and $\kappa$ are $O(n^2)$, both algorithms generate the same 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, this bound seems rather pessimistic, and we believe that it can be improved (as is strongly suggested by our experimental results).

We remark that there exist (rather pathological) examples in which $\kappa \ll n^2$ and the DC algorithm performs
<table>
<thead>
<tr>
<th>input name</th>
<th>description</th>
<th>figure</th>
</tr>
</thead>
<tbody>
<tr>
<td>regular</td>
<td>arbitrary triangles randomly placed inside a square</td>
<td>Figure 1</td>
</tr>
<tr>
<td>fat</td>
<td>equilateral triangles randomly placed inside a square</td>
<td>Figure 1</td>
</tr>
<tr>
<td>fat_with_grid</td>
<td>a grid-like pattern partially covered by many random fat triangles; half of the triangles form the grid and the other half are the fat triangles</td>
<td>Figure 2</td>
</tr>
</tbody>
</table>

Table 1: The different data sets

considerably worse than the RIC algorithm. 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.

3. EXPERIMENTAL RESULTS

In this section we present experimental results comparing the RIC and the DC algorithms. We start by describing the input sets and the implementation 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 $|R|$ times. Another operation is testing whether a vertex lies inside a triangle, which we do $|R|$ times. What is the best size of $R$ in practice 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 described in Table 1 and in Figures 1 and 2. For each type of input, we have experimented with a varying number of triangles, up to 800 per run. The complexity of the union of fat is almost linear in the number of triangles (see Figure 1), while the union of fat_with_grid can have a superlinear number of holes. In addition to the three types of input presented here, we performed other tests on different data sets which are not reported in this abstract. In all our experiments the DC algorithm performs better than the RIC.

3.2 Implementation Details

Our implementation of the union algorithms is based on the CGAL (version 2.3) and LEDA (version 4.3) libraries. The implementation employs the CGAL maps and arrangements packages [3, 5], and uses exact arithmetic. We use LEDA's rational kernel which employs a floating point filter in computing predicates [8]. 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 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 to the partial union constructed is performed by the insertion of the three edges defining the triangle, one at a time. Each insertion of an edge $e$ to 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 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 [3]. Next, we traverse the zone of $e$. 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 in the present union (each such portion is delimited by a pair of consecutive intersection points).

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, and removing all features that lie inside the union, and have not yet been removed. The removal stage is performed by traversing all edges in the current structure, and checking whether each such edge lies inside the union (in constant time per edge, due to extra information that should be maintained in the edges). In practice, the time consumed by traversing all edges is negligible.
3.3 Results

We present experimental results of applying both 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 the yardstick we use for measuring and comparing the performance of the algorithms.

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 size of \( R \) varies between a constant, a logarithmic factor in the number of input triangles, a linear factor, and \( R \) being the full set \( V \). The results are presented in Figures 3 through 5.

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., in the case of the \( fat \) the size of \( R \) does not have a significant effect.

\(^1\)Recall that this may fail to hold in some pathological examples, when \( |V| \ll n^2 \).

input, even if we use much smaller samples \( R \), for example, samples of size linear in the input size, then we save the construction of over 1000 vertices (out of about 2000) during the incremental stage when the input consists of 190 triangles (Figure 5). The saving hardly changes when we use \( R = V \).

For the \( fat \) input (see Figure 4), the improvement is less significant than the improvement obtained for the other input sets. Notice that the amount of work the RIC algorithm performs for fat triangles is always close to linear.\(^2\) Hence improving such small amount of work is more difficult than improving the amount for regular triangles.\(^3\)

For the \( fat \) the RIC algorithm performs poorly since the intersection points of the grid are shallow on the average.

We remark that the number of vertices \( |V| \) in some of our examples is huge (reaching roughly half a million for the

\(^2\)The proof is given in the full paper.

\(^3\)Note that we do not know the effect of the randomization in the choice of the regular input on the number of positive-depth vertices constructed by one algorithm or by the other.
4. REFERENCES


