Computing a Single Cell in the Union of two Simple Polygons

Mark de Berg, Olivier Devillers, Katrin Dobrindt, Otfried Schwarzkopf

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Computing a single cell in the union of two simple polygons

Mark de Berg, Olivier Devillers, Katrin Dobrindt, Otfried Schwarzkopf

N° 2626

Juillet 1995

PROGRAMME 4

ISSN 0249-6399

INRIA
SOPHIA ANTIPOLIS

Computing a single cell in the union of two simple polygons

Mark de Berg*, Olivier Devillers**, Katrin Dobrindt***, Otfried Schwarzkopf****

Programme 4 — Robotique, image et vision
Projet Prisme

Rapport de recherche n°2626 — Juillet 1995 — 10 pages

Abstract: This note presents a non trivial combination of two techniques previously used with randomized incremental algorithms: the lazy cleaning scheme [dBDS94] to maintain structures with 'non local' definition and the $O(n \log^* n)$ acceleration when some additional information about the data is known [Sei91, CCT92, Dev92]. Authors assume that the reader is somehow familiar with this techniques.

If, we are interested in computing a single face in the intersection of two simple polygons, the two previous techniques can be used. The lazy approach allows to compute a single face in an arrangement of line segments in expected time $O(n\alpha(n)\log n)$ but this technique do not exploit the organization of the segments in simple polygons. The accelerated framework compute the whole intersection of the two simple polygon in $O(n \log^* n + A)$ where $A = O(n^2)$ is the number of intersection points, and then the relevant connected component can be extracted easily in $O(n)$ time. The two techniques can be combined to reach an algorithm whose expected time complexity is $O(n \log^{\star 2} n)$ which improve previous bounds above.

(We denote by $\alpha(n)$ the pseudo-inverse of Ackermann function ; α grows to infinity, but excessively slow, for $n \leq 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \Big\} 2^{65536}$ $\alpha(n) \leq 3$. We denote by $\log^{(i)} n = \overbrace{\log\log\ldots\log}^{i \text{ times}} n$; $\log^\star n = \inf\{k; \log^{(k)} n \leq 1\}$ and $(\log^\star n)^2$ by $\log^{\star 2} n$. Thus for any imaginable data set $n \leq 2^{65532}$ and thus $\log^\star n \leq 5$.)

Key-words: randomized algorithms, computational geometry, arrangement, trapezoidal map

(Résumé : tsvp)

Research of M. de Berg, K. Dobrindt and O. Schwarzkopf are supported by the Dutch Organization for Scientific Research (NWO). This work was done while O. Devillers was visiting Utrecht University and was also supported in part by ESPRIT Basic Research Action 6546 (PROMotion).

* Vakgroep Informatica, Universiteit Utrecht, Postbus 80.089, 3508 TB Utrecht, the Netherlands. E-mail: markdb@cs.ruu.nl.

** INRIA, B.P.93, 06902 Sophia-Antipolis cedex (France), Phone: +33 93 65 77 63, E-mail: Olivier.Devillers@sophia.inria.fr.

*** Vakgroep Informatica, Utrecht. Current address is Microsoft, Parc Marépolis, 644, Chemin du Puy, 06600 Antibes, France. E-mail: katrind@microsoft.com.

**** Vakgroep Informatica, Utrecht. Current address is Dept. of Computer Science, Postech, San 31, Hyoja-dong, Pohang, Kyungbuk 790-784, Korea. E-mail: otfried@vision.postech.ac.kr.

Unité de recherche INRIA Sophia-Antipolis
2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex (France)
Téléphone : (33) 93 65 77 77 – Télécopie : (33) 93 65 77 65

Calcul d'une cellule de l'intersection de deux polygones simples

Résumé : Cette note présente une possibilité de combinaison entre deux techniques déjà appliquées aux algorithmes randomisés : le nettoyage paresseux [dBDS94] pour la mise à jour de structures à définition non locale et l'accélération en $O(n \log^\star n)$ d'algorithmes utilisant des informations supplémentaires sur les données [Sei91, CCT92, Dev92]. Nous supposons le lecteur au courant de ces méthodes.

Si l'on désire calculer une cellule dans l'intersection de deux polygones simples, les deux techniques précédentes peuvent être utilisées. L'approche paresseuse nécéssite un temps moyen $O(n\alpha(n)\log n)$ sans utiliser le fait que les segments forment des polygones simples. La technique accélérée calcule l'intersection des deux polygones en temps $O(n \log^\star n + A)$ où A est le nombre de points d'intersections, la cellule recherchée peut facilement être extraite en temps linéaire. Les deux techniques peuvent être combinées pour obtenir un algorithme de complexité $O(n \log^{\star 2} n)$ améliorant les bornes précédentes. ($\alpha(n)$ est la pseudo-inverse de la fonction d'Ackermann α est une fonction à croissance très lente pour $n \leq 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}$ $\alpha(n) \leq 3$. Si $\log^{(i)} n = \overbrace{\log\log\ldots\log}^{i \text{ fois}} n$; $\log^\star n = \inf\{k; \log^{(k)} n \leq 1\}$ et $(\log^\star n)^2$ désigne $\log^{\star 2} n$. Tout ensemble de données concevable ayant une taille $n \leq 2^{65532}$ dans la pratique $\log^\star n \leq 5$.)

Mots-clé : algorithmes randomisés, géométrie algorithmique, arrangement, carte des trapézes

1 Incremental randomized algorithms

Randomized incremental construction is now an extremely versatile tool in the field of computational geometry [CS89, Mul93, BDS+92]. Here a geometric structure induced by a set S of certain geometric objects is computed by adding the objects in S in random order, meanwhile maintaining the structure induced by the current subset. In the following we summarize the main ideas and results in our context.

Consider a set S of n line segments in the plane. The *trapezoidation* induced by this set is a subdivision of the plane into *trapezoids* and is obtained by extending vertical segments from every endpoint of a segment or an intersection point of two segments upwards and downwards—see Figure 1. The randomized incremental algorithm for computing the trapezoidation maintains a data-structure, the so-called *history graph* [BDS+92, GKS92]. The history graph for an insertion sequence $s_1, s_2, \ldots, s_n$ of the segments of S is a directed acyclic graph. Its nodes are the trapezoids that have been created during the incremental construction. The trapezoids of the current trapezoidation are leaf nodes in the history graph. When a new segment s_r is added to the structure, the trapezoids of the current trapezoidation that are intersected by s_r are *located* by a partial traversal of the history graph. The history graph is then *updated* as follows. Each leaf node corresponding to an intersected trapezoid is split by s_r into at most four new trapezoids. These new trapezoids are added to the history graph as leaves below the intersected trapezoid (which is now no longer a leaf).

If the segments of S are inserted in random order, the leaf nodes of the history intersected by a new segment can be found in $O(\log n)$ expected time, and the history graph can be updated in expected $O(1 + A/n)$ time, where A is total number of intersection points. Thus the expected time complexity of computing the trapezoidation induced by a set of n line segments with A intersection points is $O(A + n \log n)$.

2 Lazy randomized incremental algorithms

While the entire arrangement induced by a set of n line segments might have quadratic combinatorial complexity, the combinatorial complexity of any of its face is only $O(n\alpha(n))$ [GSS89]. In the following we are interested in computing a single face in the arrangement of line segments, say the face containing the origin, instead of the entire arrangement. Actually we compute the trapezoidation of this face.

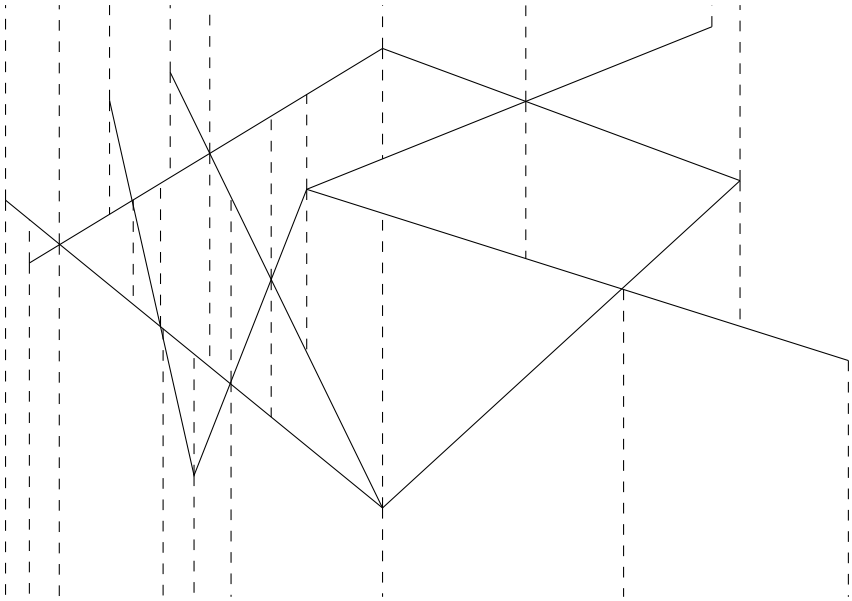

Figure 1: Example of trapezoidal map

However, single faces in arrangements do not fit into the usual framework of randomized incremental construction, because it cannot be decided locally whether a trapezoid belongs to the current structure [dBDS94]. More precisely, a new inserted segment may cut off parts of the current face by separating them from the origin. It is difficult to determine the trapezoids that are cut off (which depends on the complete configuration of the segments). *Lazy* randomized incremental algorithms are suited for computing, for instance, single faces in arrangements. When inserting a segment they use the usual algorithm of the preceding section and do not attempt to identify and discard the parts of the face which are cut off. Instead they simply keep everything around, including parts that lie no longer in the current single face. This way they would, of course, end up constructing the full arrangement of segments. Therefore the structure is cleaned after inserting the 2^i-th segment, $1 \leq i \leq \log n$. To perform these clean-up steps the current structure is traversed using the neighborhood relations, and the parts outside the single face are marked. These "outside" trapezoids remain as leaves in the history graph, but need no further refinement.

The expected size of structure induced by r segments that is maintained by the algorithm is $O(r\alpha(r))$, and thus is asymptotically not larger than the face. Between the clean-ups in step 2^i and 2^{i+1}, $1 \leq i \leq \log n$, the history graph can be updated in $O(2^i\alpha(2^i))$ expected time. The clean-up in step 2^i takes time proportional to the number of trapezoids in current structure and thus $O(2^i\alpha(2^i))$. The expected time to compute the face of an arrangement of n segments containing the origin is thus $O(n\alpha(n)\log n)$.

3 Accelerated randomized incremental algorithms

If we have additional information on the input data, such as the segments form a simple polygon, this can be exploited to accelerate the algorithm [Sei91, CCT92, Dev92]. In the standard algorithm, the time complexity for inserting a new segment can be subdivided in a location part that takes logarithmic expected time and an output sensitive update part. The basic idea of the accelerated algorithms is that they do not always traverse the whole history graph in the location step, but stop from time to time to compute intermediate location information which helps with searches later on.

The trapezoidation of a simple polygon is computed by constructing the history graph as usual, up to some step r. Then a *conflict graph* storing all the intersections between the not yet inserted segments and the current trapezoidation is computed.

This is done by traversing the trapezoidation following the simple polygon using the neighborhood relations. Then insertions in the history graph are continued, but the location step starts directly at the conflict graph and traverses only the parts of the history graph below the trapezoids of the trapezoidation of the first r segments which are intersected by the new inserted segment.

The conflict graph at step r can be computed in output sensitive time. Since this gives us the intersections between the n-th segment and the trapezoidal map of the first r segments, the location time is reduced to $O(\log n/r)$. By choosing the steps where a conflict graph is computed carefully, namely $\frac{n}{\log^{(h)} n}$, $0 \leq h \leq \log^\star n$, the final running time is $O(n \log^\star n)$.

This technique allows to compute the intersection of m simple polygons with in total n vertices and A intersection points in $O(A + m \log n + n \log^\star n)$. It works similarly to the one polygone algorithm, but during the conflict graph computation, one vertex of each polygon must localized the history graph, which yields to an overall cost of $O(\log n)$ time for each polygon.

4 A single face in the intersection of two polygons

At first glance it should be possible to combine the two previous techniques to compute a single face of the intersection of two simple polygons with in total n vertices in $O(n\alpha(n) \log^\star n)$. (Any face of the intersection of two simple polygons is a simple polygon and has $O(n)$ vertices—see Figure 2.) But this does not work out so easily. The main problem is that if we want to compute a conflict graph at stage r by a tracing one polygon through the trapezoidation following one polygon, there may be $\Omega(n \log r)$ intersections points between this polygon and the "outside" trapezoids. This is illustrated in Figure 3. But we know by the moment theorem that the number of intersections between not yet inserted segments and trapezoids of the face at step r is only $O(n\alpha(r))$!

Fortunately we can cope with this problem by computing separately the intersection of the two polygons with the current face C_r and then tracing the polygons only inside C_r. The algorithm is thus as follows.

Algorithm *ComputeSingleFace*()
1. Let $s_1, s_2, \ldots, s_n$ be a random permutation of S;
2. Generate the initial conflict graph $\mathcal{G}_1$ for $s_2, \ldots, s_n$ and the trapezoids created by s_1;
3. $last\_cleanup := 1$; $last\_log := n$;

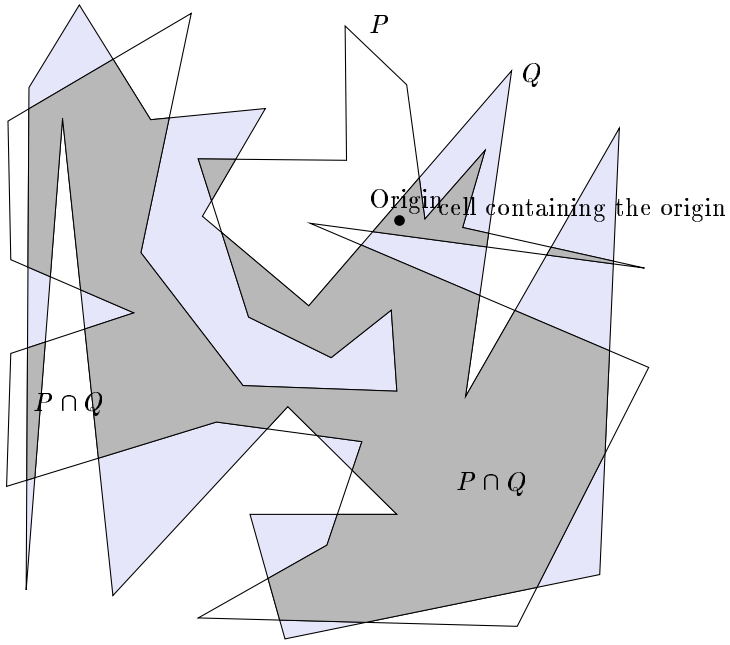

Figure 2: Example of intersection of two simple polygons

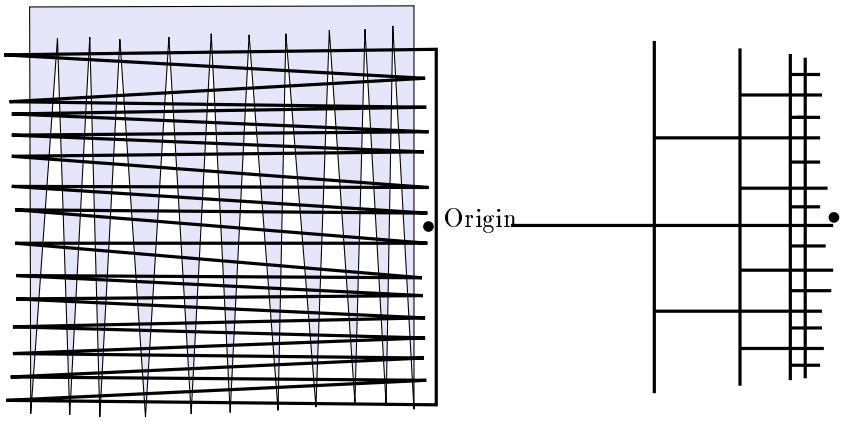

Figure 3: The number of conflicts may be $\Omega(n \log r)$.
The left part of the figure describe the two polygons and the position of the origin. The right part is the "average" shape of the trapezoidal map at some stage r, then the overlap of the shaded polygon with this map has linear size, but the overlap of the map with the other polygon produces a expected number $n \log r$ intersection points even if they mainly concern "outside" trapezoids. The number of intersection whith "inside" trapezoids is ony linear.

4. **for** $r = 2$ **to** n
5. **do** locate s_r in the history graph starting at the conflict graph $\mathcal{G}_{\frac{n}{last\_log}}$;
6. update the history graph;
7. **if** $r = 2 \cdot last\_cleanup$
8. **then** perform a clean-up step; $last\_cleanup := r$;
9. **if** $r = \frac{n}{\log(last\_log)}$
10. **then** $last\_log := \log(last\_log)$;
11. perform a clean-up step to obtain C_r;
12. compute intersections points between C_r and the two simple polygons using twice the standard accelerated algorithm;
13. compute $\mathcal{G}_{\frac{n}{last\_log}}$ by tracing the relevant parts of the two polygons through the trapezoidation of C_r;

The location of rth segment in Step 5 costs $O(r\alpha(r)\log r/last\_log)$ which sums for all n segments to $O(n\alpha(n)\log^\star n)$ and the update of the history graph in Step 6 is done in $O(n\alpha(n))$ time during the whole algorithm. The costs of all clean-up steps in Step 8 are $O(n\alpha(n))$. Step 11 takes $O(r\alpha(r))$ time. At Step 12 the intersections points between C_r and the two simple polygons are computed using twice the standard accelerated algorithm in $O(A + n'\log^\star n')$ expected time, with $A = O(n\alpha(r))$ and $n' = O(n + r\alpha(r)) = O(n)$. Thus Step 12 needs $O(r\alpha(r) + n\log^\star n)$. In Step 13 the conflict graph can be computed in $O(n\alpha(r)$ time. Since the Steps 9–13 are executed $\log^\star n$ times, $r < n$ and $\alpha(n) < \log^\star n$, the total expected time complexity of *ComputeSingleFace* is $O(n\alpha(n)\log^\star n + \log^\star n(n\alpha(n) + n\log^\star n)$ which reduces to $O(n\log^{\star 2} n)$.

5 Open problems

We would obtain an $O(n\alpha(n)\log^\star n)$ expected running time complexity if the construction of a conflict graph could be done in linear time. However, designing such algorithm is not easy, since the original polygons cannot be traced in the current and complete subdivision (the cost of tracing it through the "outside" trapezoids may be $\Omega(n\log n)$).

It is not clear that the face defined by a sample of the r edges of two simple polygon actually has complexity $\theta(r\alpha(r))$, since the final result has only linear size.

An interesting open problem the case of m polygons with in total n vertices. This problem has applications to path planning in an environment of m polygons of total complexity n. A direct generalization of the above work yields a $O(mn\log^{\star 2} n)$

algorithm which is only interesting for very small m. The lower bound for that problem is $\Omega(n \log m)$.

References

[BDS+92] J.-D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec. Applications of random sampling to on-line algorithms in computational geometry. *Discrete Comput. Geom.*, 8:51–71, 1992.

[CCT92] K. L. Clarkson, R. Cole, and R. E. Tarjan. Randomized parallel algorithms for trapezoidal diagrams. *Internat. J. Comput. Geom. Appl.*, 2(2):117–133, 1992.

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[dBDS94] M. de Berg, K. Dobrindt, and O. Schwarzkopf. On lazy randomized incremental construction. In *Proc. 26th Annu. ACM Sympos. Theory Comput.*, pages 105–114, 1994.

[Dev92] O. Devillers. Randomization yields simple $O(n \log^* n)$ algorithms for difficult $\Omega(n)$ problems. *Internat. J. Comput. Geom. Appl.*, 2(1):97–111, 1992.

[GKS92] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[GSS89] L. J. Guibas, M. Sharir, and S. Sifrony. On the general motion planning problem with two degrees of freedom. *Discrete Comput. Geom.*, 4:491–521, 1989.

[Mul93] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, New York, 1993.

[Sei91] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1:51–64, 1991.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399