1985

Fast Parallel Algorithms for Voronoi Diagrams

Micahel T. Goodrich

Colm O'Dunlaing

Chee Yap

Report Number:
85-538

FAST PARALLEL ALGORITHMS FOR VORONOI DIAGRAMS

Michael T. Goodrich

Colm O'Dunlaing, Chee Yap
Courant Institute of Mathematical Sciences

CSD-TR-538
October 1985

Fast Parallel Algorithms for Voronoi Diagrams

Michael T. Goodrich[1]
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Colm Ó'Dúnlaing[2]
Chee Yap[2]
Courant Institute of Mathematical Sciences
New York University
251 Mercer St.
New York, NY 10012

November 1985

CSD-TR-538

[1]This work partially supported by the National Science Foundation under Grant DCR-84-51393.

[2]This work partially supported by the National Science Foundation under Grants DCR-84-01898 and DCR-84-01633.

Abstract

We present two parallel algorithms for constructing the Voronoi diagram of a set of $n > 0$ line segments in the plane:

a) The first algorithm runs in $O(\log^2 n)$ time using $O(n)$ processors. This improves the previous best results (by A. Chow and also by Aggarwal, Chazelle, Guibas, Ó'Dúnlaing and Yap) in two respects. First we improve the running time by a factor of $O(\log n)$ and second the original results allow only sets of points.

b) By using $O(n^{1+\epsilon})$ processors, for any $\epsilon > 0$, we improve the running time to $O(\log n)$. This is the fastest known algorithm using a subquadratic number of processors.

The results combine a number of techniques: a new $O(\log n)$ method for point location in certain tree-shaped Voronoi diagrams, a method of Aggarwal et al for reducing contour tracing to merging tree-shaped Voronoi diagrams, and a technique of Yap for computing the Voronoi diagrams of line segments. The computational model we use is the CREW PRAM (Concurrent-Read, Exclusive-Write Parallel RAM).

1 Introduction

Computational geometry has already proved to be an important problem area, and has applications in many areas in which a fast solution is essential [12]. Thus it is natural to search for efficient parallel solutions to these problems. There is a small but growing literature on parallel computational geometry [1,2,5,8]. These papers have addressed basic problems such as convex hulls, closest pairs, segment intersection, polygon triangulation and Voronoi diagrams.

In this paper we address the problem of constructing Voronoi diagrams in the plane. Ever since the work of Shamos, the Voronoi diagram has been recognized to be one of the most versatile geometric structures in computational geometry, being the key to the efficient solution of a host of other problems. Although the Voronoi diagram was originally defined for point sets, it has been generalized in a variety of ways. One important generalization is the Voronoi diagram of a set S of $n > 0$ line segments. We assume that the line segments in S may intersect each other only at their endpoints. This problem, which is the main concern of our paper, has applications in diverse areas such as pattern recognition [9] and robotics [15]. Note that this problem subsumes as special cases (i) the original problem for point sets, and (ii) the problem of computing the Voronoi diagram of a set of disjoint polygons. The Voronoi diagram for line segments will form a partition of the plane into star-shaped regions bounded by straight and parabolic curve segments.

The first work on this problem was Drysdale's thesis [6]. He gave an $O(nc^{(\log n)^{1/2}})$ time algorithm. Subsequently, together with D.T. Lee, this was improved to $O(n \log^2 n)$ [11]. In 1979, Kirkpatrick [9] gave a new $O(n \log n)$ method for computing the Voronoi diagram of a set of n points; Kirkpatrick indicated that his method generalizes to line segments to yield an $O(n \log n)$ algorithm. In 1984, Yap [20] gave the first detailed $O(n \log n)$ time solution to the problem. Very recently, Fortune [7] has provided another $O(n \log n)$ solution; this solution is quite surprisingly in that it is based on the sweepline paradigm. However, as we will show in this paper, the divide-and-conquer technique of Yap can be combined with our parallel merging technique to give rise to a fast parallel algorithm. The sweepline approach of Fortune seems inherently sequential.

In all our discussions of parallel algorithms, we assume the concurrent read, exclusive write (CREW) model of parallelism. A parallel algorithm for Voronoi diagrams in this model was first given in the thesis of Chow [5], who provided an $O(\log^3 n)$ time n-processor algorithm. Her solution relies on an $O(\log n)$ time n-processor sorting algorithm. At the present state of the art, using the so-called AKS algorithm [3], the implicit constant in the "$O(\log n)$" time is rather impractical. In [1] an alternative to Chow's solution is provided which has the same asymptotic time and processor bounds but which relies only on, say an $O(\log^2 n)$ time sorting algorithm. In this paper, we will improve the results of [1] in two ways: we will reduce the time complexity to $O(\log^2 n)$ and generalize the problem to allow the input to be a set of line segments.

We also provide an $O(\log n)$ time algorithm using only $O(n^{1+\epsilon})$ processors. This is the fastest known $O(\log n)$ time solution using a subquadratic number of processors.

In the remainder of this extended abstract we will outline the method for a set of points and indicate how to extend it to the case of line-segments. The geometry involved for line-segments is much more intricate and will appear in the final paper.

2 Algorithm for a Set of Points

In this section we present an algorithm for constructing the Voronoi diagram of a set of points, based on the popular divide-and-conquer paradigm. In this case we divide the set of points into two halves, recursively solve each subproblem in parallel, and then quickly merge the solutions to the subproblems. Shamos and Hoey [18] were the first to note that in merging two Voronoi diagrams it is useful to first compute the set of edges between the two subsets which must be added to construct the Voronoi diagram of the entire set. This set of edges is known as the *contour* between the two Voronoi diagrams (see Figure 1). Unfortunately, the method Shamos and Hoey gave for efficiently building the contour sequentially does not work in the parallel setting, since it is based on the "contour-tracing" paradigm. Thus, any fast parallel algorithm based on this divide-and-conquer approach must construct the contour in an entirely different manner. The algorithm which follows does just that, constructing the contour between the two subsets in $O(\log n)$ time using $O(n)$ processors.

By not immediately attacking the general case of line-segments, we isolate two key ideas: first, we exploit the fact that computing the contour can be reduced to point location in a rather special planar subdivision. The subdivision is determined by a planar graph in the form of a complete binary tree. This reduction is due to [1]. Second, we show how to construct a search structure for such a binary tree in $O(\log n)$ parallel time which can support point location queries in $O(\log n)$ sequential time. We first give a high-level description of the algorithm, and then describe the details of how each step is performed. For the sake of exposition we will assume that the input set of points is in general position. That is, no 3 points are co-linear and no 4 points are co-circular. Modifying the algorithms presented below for the more general case is straightforward.

A High-Level Description of the Algorithm VORONOI:

Input: A set S of n points in the plane.

Output: The Voronoi diagram $VOR(S)$.

Step 1. Preprocess S by sorting the points of S by x-coordinate. (This step is performed only once.)

Step 2. Divide S into two subsets S_1 and S_2 of size $n/2$ each, using a vertical cut-line, and recursively construct the Voronoi diagrams $VOR(S_1)$ and $VOR(S_2)$ and the Convex Hulls $CH(S_1)$ and $CH(S_2)$ in parallel.

Step 3. Compute the convex hull $CH(S)$ from $CH(S_1)$ and $CH(S_2)$.

Step 4. Determine all the edges of $VOR(S_1)$, and of $VOR(S_2)$, which cross the contour.

Step 5. Using the information from Steps 3 and 4, construct the contour between S_1 and S_2.

Step 6. Use the contour determined in Step 5 to construct $VOR(S)$ from $VOR(S_1)$ and $VOR(S_2)$.

Before we describe the details, we would like to review an important observation made by Aggarwal, Chazelle, Guibas, Ó'Dúnlaing, and Yap in [1], which concerns the relationship between $VOR(S_i)$ and $CH(S_i)$. Let R_e denote the region of the plane bounded by the two perpendiculars extending from the corners of the edge e on the convex hull $CH(S_i)$. Aggarwal et al observed that the intersection of $VOR(S_i)$ and R_e has the structure of a complete binary tree with all its leaves lying on the edge e, and that an interior node is always farther from e than its children are (see Figure 2). Note also that each of the regions determined by the binary tree are convex, since these regions are part of the Voronoi diagram. We exploit this convexity in the algorithm which follows to be able to construct the Voronoi diagram of n points in the plane in $O(\log^2 n)$ time using $O(n)$ processors.

The Algorithm VORONOI:

Step 1. *Preprocess S by sorting the points of S by x-coordinate.*

Step 1 is a preprocessing step which makes the dividing done in Step 2 possible in constant time.

Complexity. This step can be performed in $O(\log n)$ time using $O(n)$ processors [3,13]. Since this step is performed only once, we could also use a sub-optimal method whose constant "behind the big-Oh" is smaller (e.g., [4,17,19]), provided the algorithm runs in at most $O(\log^2 n)$ time using $O(n)$ processors.

Step 2. *Divide S into two subsets S_1 and S_2 of size $n/2$ each, using a vertical cut-line, and recursively construct the Voronoi diagram and Convex Hull for each subset in parallel.*

Complexity. If we let $T(n)$ denote the time bounds of Steps 2 through 6, and let $P(n)$ denote the processor bounds, then this step requires approximately $T(n/2)$ time and $2P(n/2)$ processors.

Step 3. *Compute $CH(S)$ from $CH(S_1)$ and $CH(S_2)$.*

It is sufficient to find the two tangent lines between $CH(S_1)$ and $CH(S_2)$.

Complexity. Using a binary search technique of Overmars and Van Leeuwen [16] (which is also described in [14]) this step can be performed in $O(\log n)$ time using one processor.

Step 4. *Determine all the edges of $VOR(S_1)$, and of $VOR(S_2)$ which cross the contour.*

Method. Note that an edge is adjacent to the contour if and only if one endpoint is closest to a point in S_1 and the other endpoint is closest to a point in S_2. We first build a data structure D_e for each R_e, $e \in CH(S_i)$, in parallel which allows us to solve the point location problem in $O(\log n)$ time.

Let R_e be some edge region, and let T be the binary tree which partitions R_e. For any node r in T, define the *triangle* at r to be the triangle formed by the node r and the leftmost and rightmost leaf descendants of r (see Figure 3). The convexity of the tree regions guarantees that none of the triangles enclosing subtrees in T intersect each other, or intersect the interior of a tree edge. The details for the construction and use of D_e follow:

Step 4.1. Preprocess the tree T and compute for each node its preorder number, the maximal preorder among all its descendants, and therefore (implicitly) the number of descendants it has.

Step 4.2. Let t be the number of nodes in T, and let r denote the root node of T. Assign t processors to T and find the deepest descendant d of r such that the subtree at d contains more than $2t/3$ nodes (this can be done in constant time).

Step 4.3. Let T_1 be the larger of the two subtrees whose roots are children of d (so T_1 has between $t/3$ and $2t/3$ nodes). Let T_2 be obtained from T by replacing T_1 by its enclosing triangle and updating the descendant information for all nodes between d and r to take account of removing T_1.

Step 4.4. Recursively process T_1 and T_2. Note that the precomputed quantities from Step 4.1 allow us to decide in constant time whether a node in T belongs to T_1 or T_2.

Comment: The data structure D_e is simply a pointer mechanism enabling one to proceed from (the triangle enclosing) T to T_1 or T_2 in one step (after determining whether the triangle enclosing T_1 contains the query point or not). Clearly D_e contains $O(\log n)$ levels, and allows one to find the tree region of R_e containing a query point in $O(\log n)$ serial time. So we can now solve the point-location problem for each point $p \in S$.

Step 4.5. Assign a processor to each endpoint p of the Voronoi edges in $VOR(S_1)$ and $VOR(S_2)$ (WLOG $p \in VOR(S_1)$), and perform a binary search to find which region R_e of $CH(S_2)$ contains p. If there is no R_e region containing p, then we are done, since p must then belong to the "wedge" between two R_e regions, hence to the Voronoi polygon of the corner vertex.

Comment: The point location problem for p is now to find which region of the binary tree T which divides R_e contains p. We call these regions *tree regions*.

Step 4.6. Use the data structure D_e to find the tree region of R_e containing p. This of course gives us the Voronoi polygon $V_2(q)$ in $VOR(S_2)$ which contains p.

Step 4.7. We can now test for each edge whether one endpoint is closest to S_1 and the other closest to S_2, or not, in constant time. This gives us the sets edges which cross the contour. Let E_1 (E_2) be the set of edges of $VOR(S_1)$ ($VOR(S_2)$) which cross the contour. Sort E_1 and E_2 in $O(\log n)$ time with $O(n)$ processors along the y-direction using parallel prefix [10] (every edge can determine in $O(1)$ time its predecessor).

Complexity. We can construct the data structure D_e for each R_e region in $O(\log n)$ time using $O(n)$ processors, since the Voronoi diagram has $O(n)$ nodes. We have already observed that we can use D_e to solve the planar location problem for each endpoint p in $O(\log n)$ serial time. Since $VOR(S_1)$ and $VOR(S_2)$ are both planar, there are $O(n)$ endpoints p. Thus Step 4 can be performed in $O(\log n)$ time using $O(n)$ processors.

Step 5. *Using the information from Steps 3 and 4, construct the contour between S_1 and S_2.*

Step 5.1. From the supporting tangents joining $CH(S_1)$ and $CH(S_2)$ we can compute the first and last (infinite-length) edges of the contour.

Comment: From Step 4 we know all the edges E_1 of $VOR(S_1)$ and the edges E_2 of $VOR(S_2)$ which intersect the contour. Note that the regions between consecutive edges correspond to Voronoi regions.

Step 5.2. Let e be the median edge in E_1. Assigning one processor to each edge f of $VOR(S_2)$, we can determine the intersection or non-intersection of e with f in constant time. If e intersects the edges $f_1, f_2, \ldots, f_k$ in this order, the processor at each f_i can in constant time determine its neighbors f_{i-1} and f_{i+1}. It is then easy to determine the vertex where e attaches to the contour.

Step 5.3. Let e' be the next edge below e in E_1, so they bound the cell of the median point in S_1 along the contour. Using the technique of step 5.2 for e' we can determine the interval of S_2-cells along the contour which interact with p. This subdivides the cells in S_2 into those which interact with S_1 above p and those which interact with S_1 below p. Thus we can reassign the processors in S_2 and recursively compute all the contour vertices.

Comment: In some cases there will be a cell in S_2 which belongs to both the upper division and the lower division. This does not affect the asymptotic number of processors used, however.

Step 5.4. Repeat the procedure of steps 5.2 and 5.3 for the edges of E_2, and sort the set of edges determined to be part of the contour by the y-coordinates of their endpoints. We can then augment this set with the edges from Step 5.1 in constant time to form the contour.

Comment: This ordering is well-defined because the contour is monotone in the y-direction.

Complexity. Since the subdivision of steps 5.2 and 5.3 can be done in constant time, the entire step can be performed in $O(\log n)$ time using $O(n)$ processors.

Step 6. *Use the contour determined in Step 5 to construct $VOR(S)$ from $VOR(S_1)$ and $VOR(S_2)$.*

Complexity. Step 6 amounts to using the contour to construct the data structure representing the Voronoi diagram of S. It is easy to see that this can be done in $O(\log n)$ time using $O(n)$ processors.

End of Algorithm VORONOI.

We summarize the above discussion in the following theorem.

Theorem: The Voronoi diagram of a set of n points in the plane can be constructed in $O(\log^2 n)$ time on a CREW PRAM using $O(n)$ processors.

In the next section we outline how to construct the Voronoi diagram for a set of line segments.

3 Algorithm for Line Segments

We briefly indicate the new complications that arise when we extend the above method to computing the Voronoi diagram of a set of line segments. The obvious method is to divide the set of segments into two subsets whose separate Voronoi diagrams can then be merged easily. This is the major obstacle facing researchers since they all seem to lead to contours that may not be connected; the best known method for computing such contours takes $O(n \log n)$ time so that the overall algorithm is $O(n \log^2 n)$, as shown by Lee and Drysdale [11]. In [20], Yap uses an unusual way of dividing the problem: choose a vertical line L that separates the endpoints of the segments into two equal halves. Whenever this vertical line may cut a segment we imagine that we have two new segments, one on each side of L. If this method is naively followed, it can lead to an $O(n^2)$ behavior. If we are careful about not computing certain parts of the Voronoi diagram "unless we have to", we can get an $O(n \log n)$ solution. We shall show that this "radical divide-and-conquer" method can be combined with the methods of the previous section.

The basic set-up following [18] is as follows. Suppose that we have an infinite region S bounded between two vertical lines. Call such a region S a *slab*. Suppose S contains some number of line-segments (none of the segments protrude outside S). Among these segments, those that stretch from the left boundary of S to the right boundary of S are called *long*. Thus the long segments divide S into quadrilateral regions (the topmost and bottommost are also regarded as quadrilaterals for this purpose). A quadrilateral is *active* if it contains any segment that is not part of its boundary. Let L be a vertical line that divides S into a right sub-slab SR and a left sub-slab SL.

Let S contain $m > 0$ line segments that are not long. Without going into the details, it suffices to say that we can obtain an $O(\log^2 n)$ time, n-processor solution to the global problem if we can compute compute the Voronoi diagram of each active quadrilateral of S in $O(\log m)$ time using m processors. Recursively, suppose that the Voronoi diagram of the active quadrilaterals of SR and SL have been computed. An active quadrilateral Q of S is composed of a contiguous series of quadrilaterals of SL, none of which are necessary active, together with a similar series of quadrilaterals of SR. We can obtain the Voronoi diagram of the union QL and QR of these two series rather simply. Assuming this, we must now compute the Voronoi diagram $VOR(Q)$ of Q from $VOR(QL)$ and $VOR(QR)$, the diagrams of QL and QR.

As in the point case, we assume that we have computed the convex hull of the segments in each quadrilateral, excluding the long segments that bound it above and below. It is not hard to show

that the Voronoi diagram that lies in the regions R_e (defined as before) forms a complete binary tree although the edges of the tree could be parabolic arcs. However, it is no longer true that each edge of the tree can only interact with the contour at most once. However, we can further subdivide these edges into at most two pieces such that the property holds. We can further show that the triangle decomposition of the planar subdivision of a tree works, and so we can determine the edges of $VOR(QL)$ that are involved in the contour C. Note that C is connected as shown in [20]. The remaining steps are substantially as before.

Theorem: The Voronoi diagram of a set of n line segments can be computed in $O(\log^2 n)$ time on a CREW PRAM using n processors.

4 An Even Faster Algorithm

It is easy to imagine situations where one would be more interested in a fast running time, even at the expense of the number of processors required to achieve that bound. Given that the convex hull of n points can be computed in $O(\log n)$ time with $O(n)$ processors [1,2], it is easy to see that $O(n^2)$ processors can compute the Voronoi diagram of n points in $O(\log n)$ time by using $O(n)$ processors to compute the Voronoi polygon for each point. In this section we show that we can achieve this same time bound with much fewer processors. Namely, we show how to compute the Voronoi diagram of n points in $O(\log n)$ time using $O(n^{1+\epsilon})$ processors on a CREW PRAM, where $\epsilon > 0$ is some constant. Again we outline the method for the case of point sets only.

We construct the Voronoi diagram of S using a more sophisticated divide-and-conquer technique than we used before. We divide S into n^ϵ subproblems of size $n^{1-\epsilon}$ each, separated by vertical cut-lines, and compute the Voronoi diagram and convex hull of each subproblem in parallel. The merge phase consists of two stages. First, we construct the contour for every pair $VOR(S_i)$ and $VOR(S_j)$, $i \neq j$. We do this by assigning $n^{1-\epsilon}$ processors to each pair (i,j), $i,j \in \{1,\ldots,\epsilon\}$, and use the methods of the previous algorithm to compute the contour between $VOR(S_i)$ and $VOR(S_j)$. Then we use all these contours to construct the Voronoi polygon $V(p)$ for each point $p \in S$, to complete the construction of $VOR(S)$. This can be done by assigning n^ϵ processors to each point $p \in S$ to find the subchain of each contour intersecting $V_i(p)$, $p \in S_i$. Since each subchain is convex in $V_i(p)$, each processor needs to compute at most 2 points of intersection, which can be done in $O(\log n)$ time. After doing this, we can easily compute the set of all contour edges constructed for S_i which intersect $V_i(p)$, call this set of edges $E(p)$. By assigning $|V_i(p)| + |E(p)|$ processors to every point $p \in S$, we can compute $V(p)$ from $V_i(p)$ and $E(p)$ in $O(\log n)$ time. This amounts to computing the intersection of all the half-planes determined by the edges of $V_i(p)$ and $E(p)$, and can be done by using a technique similar to the one used in [2] to solve the convex hull problem. Finally, we build $VOR(S)$ from the set of Voronoi polygons we just constructed. It should be clear that this

can all be done in $O(\log n)$ time using $O(n)$ processors.

We summarize the above discussion in the following theorem.

Theorem: The Voronoi diagram of n points in the plane can be constructed in $O(\log n)$ time using $O(n^{1+\epsilon})$ processors on a CREW PRAM.

5 Conclusion

In this paper we presented two algorithms for the Voronoi diagrams for a set of n line segments where the line segments may intersect only at their endpoints, and a line segment may degenerate into points. The algorithms are fairly complicated and involved a combination of several techniques. It seems to be a considerable challenge to further improve our $O(\log^2 n)$ time bound using n processors. On the other hand, there is no reason to think that they are optimal.

References

[1] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. Yap, "Parallel Computational Geometry," to appear in *Proc. 25th IEEE Symp. on Foundations of Computer Science, 1985.*

[2] M.J. Atallah and M.T. Goodrich, "Efficient Parallel Solutions to Geometric Problems," *1985 Proc. Int. Conf. on Parallel Processing*, St. Charles, IL., pp. 411–417.

[3] M. Ajtai, J. Komlós, and E. Szemerédi, Sorting in $c \log n$ parallel steps, *Combinatorica*, Vol. 3, 1983, pp. 1–19.

[4] A. Borodin and J.E. Hopcroft, "Routing, Merging, and Sorting on Parallel Models of Computation," *Jour. of Comp. and Sys. Sci.*, Vol. 30, No. 1, February 1985, pp. 130–145.

[5] A. Chow, "Parallel Algorithms for Geometric Problems," Ph.D. dissertation, Computer Science Dept., University of Illinois at Urbana-Champaign, 1980.

[6] R.L. Drysdale, III, "Generalized Voronoi Diagrams and Geometric Searching," STAN-CS-79-705, Computer Science Tech. Report, Stanford Univ., PhD dissertation, 1979.

[7] S. Fortune, "A sweepline algorithm for Voronoi diagrams," Preprint, October 1985.

[8] M.T. Goodrich, "An Optimal Parallel Algorithm For the All Nearest-Neighbor Problem for a Convex Polygon," Purdue University Computer Science Tech. Report CSD-TR-533, August 1985.

[9] D.G. Kirkpatrick, "Efficient Computation of Continuous Skeletons," *20th IEEE Symp. on Foundations of Computer Science, 1979*, pp. 18–27.

[10] C.P Kruskal, L. Rudolph, and M. Snir, "The Power of Parallel Prefix," *1985 Proc. Int. Conf. on Parallel Processing*, St. Charles, IL., pp. 180–185.

[11] D.T. Lee and R.L. Drysdale, III, "Generalization of Voronoi diagrams in the plane," *SIAM J. Comp.*, Vol. 10, 1981, pp. 73–87.

[12] D.T. Lee and F.P. Preparata, "Computational Geometry—A Survey," *IEEE Trans. on Computers*, Vol. C-33, No. 12, December 1984, pp. 1072–1101.

[13] T. Leighton, "Tight Bounds on the Complexity of Parallel Sorting," *IEEE Trans. on Computers*, Vol. C-34, No. 4, April 1985, pp. 344–354.

[14] K. Mehlhorn, *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, Springer-Verlag (New York: 1984).

[15] C. Ó'Dúnlaing and C.K. Yap, "A retraction method for planning the motion of a disc," *Jour. Algorithms*, Vol. 6, 1985, pp. 104–111.

[16] M.H. Overmars and J. Van Leeuwen, "Maintenance of Configurations in the Plane," *Jour. of Comp. and Sys. Sci.*, Vol. 23, 1981, pp. 166–204.

[17] F.P. Preparata, "New Parallel-Sorting Schemes," *IEEE Trans. on Computers*, Vol. C-27, No. 7, July 1978, pp. 669–673.

[18] M.I. Shamos and D. Hoey, "Closest-Point Problems," *Proc. 16th IEEE Symp. Found. of Computer Science*, Oct. 1975, pp. 151–162.

[19] Y. Shiloach and U. Vishkin, "Finding the Maximum, Merging, and Sorting in a Parallel Computation Model," *Journal of Algorithms*, Vol. 2, 1981, pp. 88–102.

[20] C.K. Yap, An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments, NYU Report No. 161, October 1984.

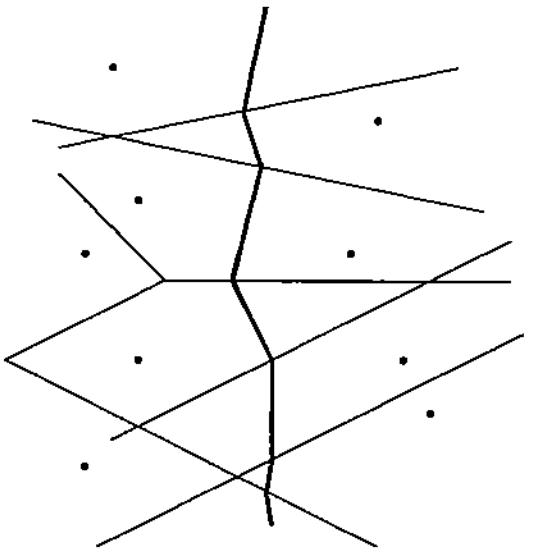

Figure 1: The contour between two Voronoi diagrams.

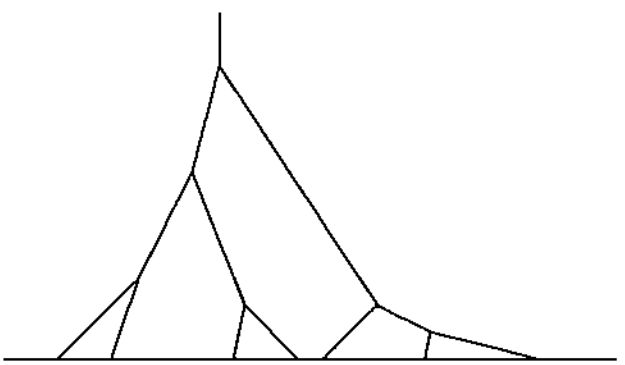

Figure 2: The binary tree structure of an R_e region.

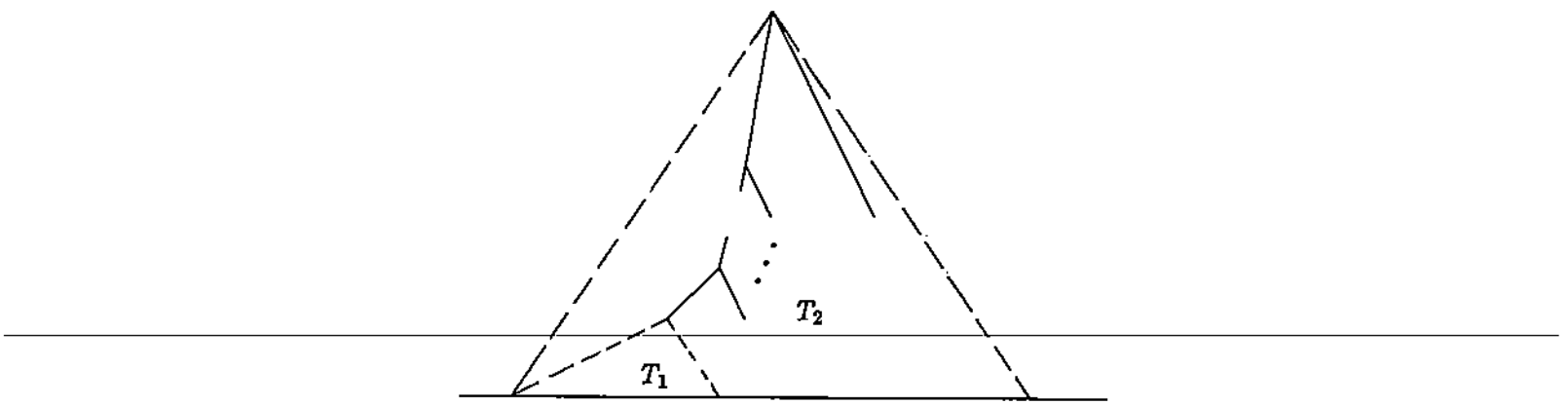

Figure 3: The partitioning of the binary tree T into T_1 and T_2. The dashed lines indicate the enclosing triangles.