Information Processing Letters 34 (1990) 155–160
North-Holland

9 April 1990

A STRAIGHTFORWARD ITERATIVE ALGORITHM FOR THE PLANAR VORONOI DIAGRAM

John C. TIPPER

Department of Geology, Australian National University, G.P.O. Box 4, Canberra, A.C.T. 2601, Australia

Communicated by P.C. Poole
Received 23 November 1988
Revised November 1989

Several published algorithms construct the planar N-point Voronoi diagram by adding the points in turn to a diagram already constructed for a subset of those points. The algorithm described here works in this way, but stipulates that every added point be outside the convex hull of the already constructed partial diagram. This stipulation, which is implemented by presorting the points by increasing x-coordinate, ensures that the core of the algorithm runs in almost linear time. The algorithm is inherently iterative, not recursive, and provides a rapid, efficient way to construct the Voronoi diagram for large data sets.

Keywords: Applications: graphics, spatial data structures, triangulation, Voronoi diagram

1. Introduction

Consider the set $P = \{P_1, P_2, \ldots, P_N\}$ of N points in the plane. The Voronoi polygon V_i, corresponding to point P_i, bounds that region of the plane within which all points are closer to P_i than to any other element of P. The Voronoi diagram for P is the set $V = \{V_1, V_2, \ldots, V_N\}$.

Dividing the plane in this way is a common first step in much of spatial analysis, for each Voronoi polygon is the most natural statement of the "zone of influence" of its associated data point, a critical concept irrespective of the particular data involved. Lee and Schachter [7] and Boots and Murdoch [1] refer to many applications of the Voronoi diagram, to data drawn from fields as disparate as astronomy, computer science, crystallography and the earth sciences. Indeed it is precisely this variety of application that has led to the Voronoi diagram being known by such a variety of names: "Dirichlet domain", "proximal polygon", "S cell", "Thiessen polygon", "tile", "Wigner–Seitz cell" are all synonyms for "Voronoi polygon".

Creating the Voronoi diagram, either from a given set of data points or in conjunction with the generation of that set by some spatial point process [4,1], is a problem that has troubled many workers. Computational strategies have become more sophisticated since the initial brute-force solutions, an inevitable result of increased understanding of the mathematical properties of the Voronoi diagram: Preparata and Shamos [12] summarise the most important of these. What are needed, above all, are robust algorithms that can create the Voronoi diagram for large point sets ($N > 1000$), rapidly and efficiently. This paper describes one such algorithm.

It should be noted that the Voronoi diagram is not restricted just to points in the plane, but that its computation in k-dimensional space is very much more complex. This present algorithm applies only to the planar case: general k-dimensional algorithms are given by, for instance, Brown [3], Bowyer [2] and Watson [16].

2. Construction of the Voronoi diagram: Some preliminaries

In solving many problems in computational geometry an approach that uses the Voronoi diagram is often found to have substantial advantages, for many such problems may be transformed directly into one another via the structure of the Voronoi diagram [12]. Examples include triangulation, determination of the convex hull and the Euclidean minimum spanning tree, and various nearest neighbour problems. Sorting, too, can be cast in the light of the Voronoi diagram. It is hardly surprising, then, that solutions of these problems can themselves assist in constructing that same Voronoi diagram. Two constructs in particular, the Delaunay triangulation and the convex hull, are of critical importance.

The Delaunay triangulation of the set P is formed by connecting all point pairs, P_i and P_j, for which the corresponding Voronoi polygons, V_i and V_j, are adjacent. This triangulation, often taken as the preferred triangulation of a point set [11,6,14,7,10], is the dual of the Voronoi diagram.

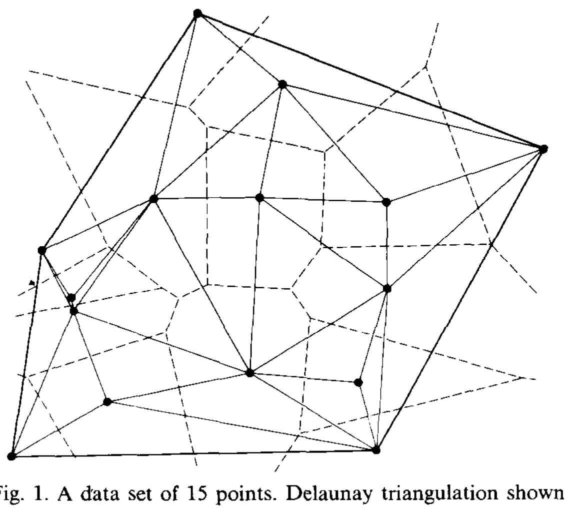
Fig. 1. A data set of 15 points. Delaunay triangulation shown by solid lines. Voronoi diagram shown by broken lines. Thicker solid lines define the convex hull.

The convex hull of the set P bounds the smallest convex domain containing P: its segments are also segments of the Delaunay triangulation (Fig. 1).

3. An iterative algorithm

Assume initially that the set P is divided into two subsets, A and B, such that the convex hull of A encloses no elements of B. Assume furthermore that for A (of size not less than three) the Delaunay triangulation and convex hull have both been computed.

The algorithm proceeds by transferring elements one by one from B to A, at each stage maintaining the condition that the convex hull of the new A encloses none of the elements remaining in B. Between each transfer, the convex hull and Delaunay triangulation of A are both updated, the latter by the swapping technique of Lawson [6]. The process terminates when B is empty, and the Delaunay triangulation of A is then that of P. A quite straightforward transformation then produces the Voronoi diagram of P.

This type of fundamentally iterative approach has been used successfully many times before [6,5,7], the Voronoi diagram for a subset of the data points being successively modified by the addition of the remaining points in turn until the final diagram is obtained. This present algorithm, however, achieves an important simplification in the solution by insisting that every added point always be outside the convex hull of the already processed subset. This condition, most readily fulfilled by adding the points in order of increasing x-coordinate, implies that sorting (one of the problems transformable to the Voronoi diagram problem) must be a key preliminary part of the algorithm. With the set P sorted, the initial division is into $A = \{P_1, P_2, P_3\}$, and $B = \{P_4, P_5, \ldots, P_N\}$. The Delaunay triangulation and convex hull of A are identical, the set of segments $\{P_1P_2, P_2P_3, P_3P_1\}$.

Point P_4 is added to A by forming segments that connect it to all points on the convex hull of A that are visible from it (Fig. 2), points P_i being defined as visible from P_4 if it is the closest point to P_4 that is both on the convex hull of A and on

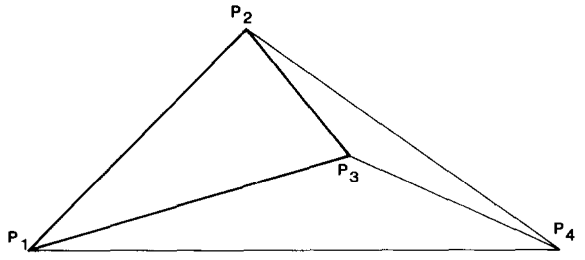

Fig. 2. Iterative construction of the Delaunay triangulation. Point P_4 is connected to all points on the convex hull of subset $A=\{P_1, P_2, P_3\}$, because all are visible from it. Segments P_1P_3 and P_2P_3 must be tested for retention in the Delaunay triangulation of the new subset $A=\{P_1, P_2, P_3, P_4\}$.

the line P_4P_i. Each of these newly formed segments is added to the Delaunay triangulation of the new $A=\{P_1, P_2, P_3, P_4\}$: two of them must also belong to the new convex hull. The convex hull is then updated, firstly by adding the two new segments from P_4 (P_2P_4 and P_1P_4 in the configuration shown in Fig. 2), and secondly by deleting any previous hull segments which are now internal (P_2P_3 and P_1P_3 in Fig. 2). Each of these deleted segments is then tested to see if it is to be retained in the Delaunay triangulation, using the max-min angle criterion of Lawson [6]. If it is not, another must replace it (for example P_kP_n replacing P_lP_m

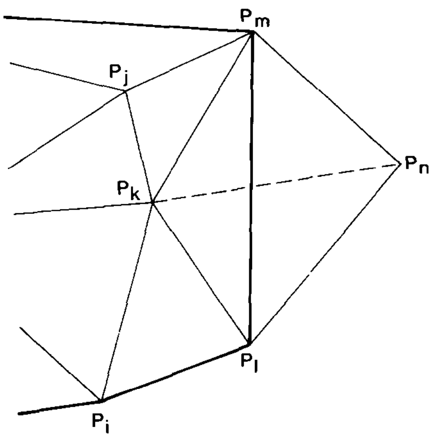

Fig. 3. Addition of point P_n to subset $A=\{P_a,\ldots, P_m\}$: thicker solid line defines its convex hull. Segment P_lP_m is not part of the Delaunay triangulation of the new subset $A=\{P_a,\ldots, P_m, P_n\}$, because the minimum angle in triangles $P_kP_lP_m$ and $P_lP_mP_n$ is less than the minimum angle in triangles $P_kP_mP_n$ and $P_kP_lP_n$. Segment P_lP_m is replaced by segment P_kP_n, and segments P_kP_l and P_kP_m are then tested.

in Fig. 3), and this in turn will call other segments of the existing Delaunay triangulation into question (P_kP_l and P_kP_m in Fig. 3). This procedure is continued until no more segments must be replaced, at which stage the Delaunay triangulation of the new A has been obtained [6]. With point P_4 now part of A, point P_5 is next added, and so on. The core of the algorithm is thus a single loop of $N-3$ iterations.

In implementing this algorithm a data structure is needed that enables both the updating of the convex hull and the operation of the swapping technique to be as efficient as possible. The convex hull itself is appropriately stored as a linked list. This is simple to update and, as this is done, the segments being deleted from the convex hull are obtained directly. It is these segments that are the starting candidates for the swapping technique. The data structure for each segment is (1) a sequential list containing the indices of the data points at each end of the segment, and those which are its left-hand and right-hand neighbours (P_l, P_m, P_k, P_n for the segment being replaced in Fig. 3), and (2) linked lists which record for each data point the indices of the segments which radiate from it. With this cross-indexed structure, the replacement of any one segment automatically points to those neighbouring ones which must then be tested next. Likewise the correct configuration of a replacing segment is automatically obtained from that of the segment that it has replaced. Searching within the data structure is minimised.

4. Performance of the algorithm

Constructing the Voronoi diagram (or the Delaunay triangulation) is a problem for which the asymptotically optimal running time is commonly claimed to be $\Theta(N \log N)$, [12]. Two algorithms which achieve this are described by Lee and Schachter [7] and Preparata and Shamos [12], each using a divide-and-conquer approach. Other algorithms are commonly $O(N^2)$, although the iterative algorithm also described by Lee and Schachter [7] is empirically found to be approximately $O(N^{3/2})$. Maus [9] has suggested that Radix

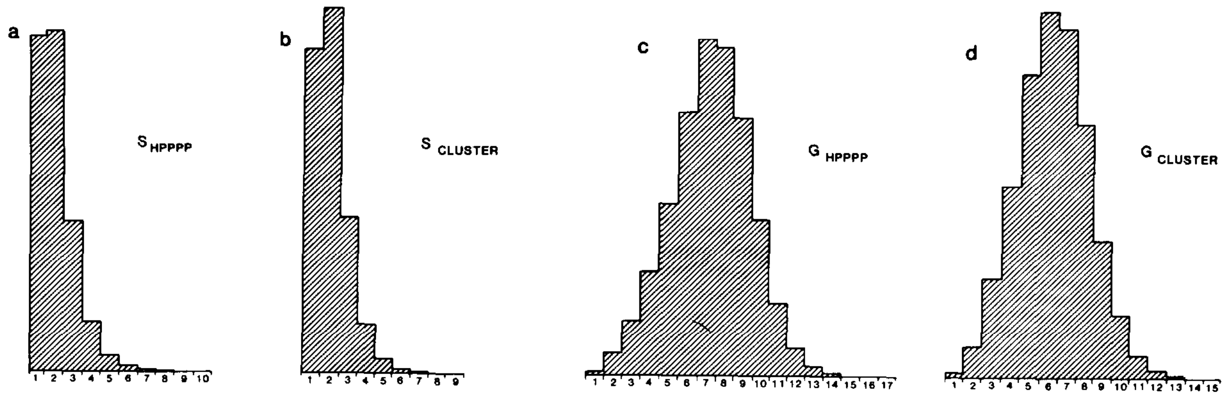

Fig. 4. (a), (b) Distribution of S, the initial number of segments called into question by the addition of a new point. (c), (d) distribution of G, the number of generations of segment replacement. (a), (c) HPPPP is a homogeneous planar Poisson point process. The distributions are derived from five runs, each of 10000 points. (b), (d) CLUSTER is a cluster process. The distributions are derived from five runs, each of 5000 points in approximately 33 clusters.

sorting can be used as the basis for an O(N) Delaunay triangulation algorithm, as Radix sorting is potentially capable of sorting N points, each represented as K bits, in O(NK) time. Although this analysis is quite appropriate for low precision work (low K), for the general case of real, randomly positioned data points the $\Theta(N \log N)$ result is correct.

In analysing the asymptotic, worst-case performance of this present algorithm, it is important to recognise that, unlike most other Voronoi algorithms, it can be formally divided into several distinct parts, each complete in itself. It is thus appropriate to analyse the performance of each part separately. These are: (1) the initial sorting by increasing x-coordinate, (2) the core loop, (3) the transformation from Delaunay triangulation to Voronoi diagram. The running time for the first part is dependent on the sorting algorithm selected, $\Theta(N \log N)$ being asymptotically optimal. For the core loop the running time is O(N^2) in the worst case, a feature that is typical of several iterative Voronoi algorithms [7]. The running time for the third part is O(N). In the worst case, then, the algorithm runs in O(N^2) and would not be preferred to the $\Theta(N \log N)$ algorithms, especially for large data sets.

For real applications, however, the situation is markedly different, as the running time of the core loop may, for many data sets, be close to O(N). The running time of the algorithm as a whole will then be controlled by the running time of its sorting part: the algorithm as a whole will run in $\Theta(N \log N)$. For applications in which the data points are supplied already ordered by increasing x-coordinate, the algorithm is even more attractive. Under these circumstances, with sorting unnecessary, the running time to construct the Voronoi diagram is that of the core loop, approximately O(N). This is the case for data obtained by scanning digitisers, and may also apply in some spatial simulation studies [8].

The running time of the core loop is dependent on two factors: (1) the number of segments of the existing Delaunay triangulation initially called into question as each new point is added (this is one less than the number of points on the convex hull that are visible to that point), and (2) the number of generations of segment replacement necessary at each step to ensure that the max-min criterion is satisfied throughout. A theoretical treatment of either factor seems unpromising except for the simplest of point configurations, and the contention that the core loop may be close to O(N) is based largely on empirical evidence. Statistics for the initial number of segments called into question, S, and for the number of generations of segment replacement, G, are presented in Fig. 4, for realisations of two common types of point process.

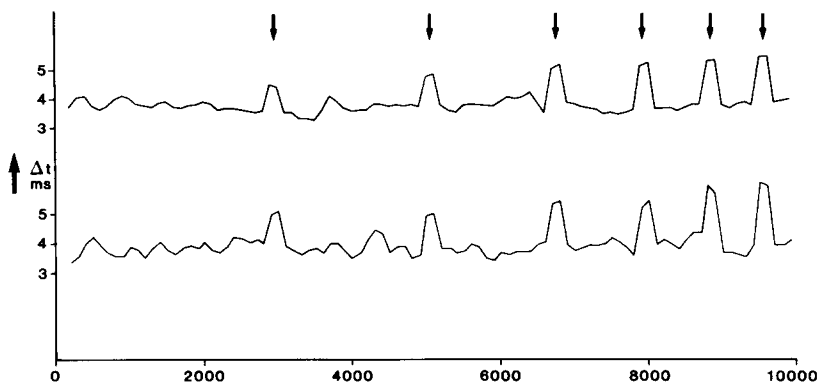

Fig. 5. Core loop timings for two runs of HPPPP, each with 10000 points. Δt is the cpu-time increment for each iteration of the core loop. Arrowed peaks result from a garbage collector periodically compacting the program's storage.

For the homogeneous Poisson point process on a rectangle, the simplest and perhaps the commonest model in spatial analysis, it is possible to give a weak upper bound for the expected number of points on the convex hull visible to any new point that is added. This is one half of the expected number of points on the convex hull for an N-point set:

$$\tfrac{4}{3}(\gamma + \log_e N) + O(1),$$

for large N, where γ is Euler's constant [13]. This bound would seem to point to a running time of $O(N \log N)$.

This figure must, however, systematically overestimate the number of visible points because of the effect that the addition of any new point has on the number of segments of the convex hull. Only if just two points are visible to the new point (Fig. 3) does the number of segments in the convex hull increase as a point is added. If three or more points are visible, the addition of a new point must *always* reduce the number of segments in the convex hull; and hence the maximum number of starting candidates for swapping as the subsequent point is added. This effect is very clearly seen in the statistics for S (Fig. 4), as for both the processes used the expected value of S is a full order of magnitude less than the calculated upper bound. Tests indicate that S also seems independent of N.

The statistics for G (Fig. 4) do not support the contention of Lee and Schachter [7] that the swapping technique rarely involves more than two generations of segment replacement: up to ten generations at any step is quite a common phenomenon. There is, however, no indication that G is at all dependent on N, and indeed little reason to suppose that it should be. With S and G both apparently independent of N, and S generally small, the core loop must be expected to be close to $O(N)$. This is very strongly supported by timing results for a FORTRAN implementation of the algorithm (Fig. 5).

5. Discussion

The practical advantages that this algorithm possesses arise from its treating creation of the Voronoi diagram as a problem that may be posed in three distinct parts: sorting, construction of the Delaunay triangulation, and transformation of that to the Voronoi diagram. This enables computational strategies to be selected that are appropriate to each part independently, allows the sorting part to be bypassed if necessary, and allows the transformation to the Voronoi diagram to be omitted if only the Delaunay triangulation is required.

Although divide-and-conquer algorithms have been shown to be optimal for the problem as a whole, it is only in its sorting part that there is inherent recursion: the actual construction of the Delaunay triangulation from the x-ordered point set can equally well be formulated iteratively. A standard (recursive) approach can thus be used to solve the sorting problem and the iterative scheme outlined here then used to create the Delaunay triangulation. What results is an algorithm that is easily understood, straightforward to program in any language, economical of space and, especially for preordered data sets, highly efficient in running time.

The FORTRAN implementation of the algorithm is described elsewhere [15]. It has been developed on a VAX 8700, operating under VMS, and all timings given here refer to that environment. The program has storage requirements of $O(N)$, and contains GKS code to enable both the Delaunay triangulation and Voronoi diagram to be plotted. Important features of the implementation are (1) that line intersections are never calculated, and (2) that calculations involving angles occur only during the swapping process. As a result, rounding errors present few problems, even in single precision, although double-precision arithmetic is used for added safety.

Acknowledgment

I thank Mike Robson and Brian Molinari for their advice and encouragement. Margaret Mackisack, Roger Miles, Mike Robson and an anonymous reviewer provided helpful comments on the manuscript.

References

[1] B.N. Boots and D.J. Murdoch, The spatial arrangement of random Voronoi polygons, *Comput. Geosci.* **9** (1983) 351–365.

[2] A. Bowyer, Computing Dirichlet tessellations, *Comput. J.* **24** (1981) 162–166.

[3] K.Q. Brown, Voronoi diagrams from convex hulls, *Inform. Process. Lett.* **9** (1979) 223–228.

[4] I.K. Crain, The Monte-Carlo generation of random polygons, *Comput. Geosci.* **4** (1978) 131–141.

[5] P.J. Green and R. Sibson, Computing Dirichlet tessellations in the plane, *Comput. J.* **21** (1978) 168–173.

[6] C.L. Lawson, Software for C^1 surface interpolation, in: J.R. Rice, ed., *Mathematical Software III* (Academic Press, New York, 1977) 161–194.

[7] D.T. Lee and B.J. Schachter, Two algorithms for constructing a Delaunay triangulation, *Internat. J. Comput. Inform. Sci.* **9** (1980) 219–242.

[8] P.A.W. Lewis and G.S. Shedler, Simulation of nonhomogeneous Poisson processes by thinning, *Naval Res. Logist. Quart.* **26** (1979) 403–413.

[9] A. Maus, Delaunay triangulation and the convex hull of *n* points in expected linear time, *BIT* **24** (1984) 151–163.

[10] M.J. McCullagh and C.G. Ross, Delaunay triangulation of a random data set for isarithmic mapping, *Cartograph. J.* **17** (1980) 93–99.

[11] R.E. Miles, On the homogeneous planar Poisson point process, *Math. Biosci.* **6** (1970) 85–127.

[12] F.P. Preparata and M.I. Shamos, *Computational Geometry* (Springer, New York, 1985).

[13] A. Rényi and R. Sulanke, Über die konvexe Hülle von *n* zufällig gewählten Punkten I., *Z. Wahrscheinlichkeitstheorie und verwandte Gebiete* **2** (1963) 75–84.

[14] R. Sibson, Locally equiangular triangulations, *Comput. J.* **21** (1978) 243–245.

[15] J.C. Tipper, A FORTRAN program to construct the planar Voronoi diagram, in preparation.

[16] D.F. Watson, Computing the *n*-dimensional Delaunay tessellation with application to Voronoi polytopes, *Comput. J.* **24** (1981) 167–172.