Computers & Geosciences Vol. 19, No. 10, pp. 1463–1474, 1993
Printed in Great Britain.

0098-3004/93 $6.00 + 0.00

FAST TOPOLOGICAL CONSTRUCTION OF DELAUNAY TRIANGULATIONS AND VORONOI DIAGRAMS

VICTOR J. D. TSAI
Department of Civil and Environmental Engineering, University of Wisconsin-Madison, Madison, WI 53706, U.S.A.

(Received 18 February 1993; revised 14 June 1993)

Abstract—This paper describes a *Convex Hull Insertion* algorithm for constructing the Delaunay triangulation and the Voronoi diagram of randomly distributed points in the Euclidean plane. The implemented program on IBM-compatible personal computers takes benefits from the partitioning of data points, topological data structures of spatial primitives, and features in C++ programming language such as dynamic memory allocation and class objects. The program can handle arbitrary collections of points, and delivers several output options to link with GIS and CAD systems. Empirical results of various sets of up to 50,000 points show that the proposed algorithm speeds up the construction of both tessellations of irregular points in expected linear time.

Key Words: Delaunay triangulation, Triangulated irregular network, Voronoi diagram, Thiessen polygons, Convex hull, Geographical information systems, Computational geometry, Topology, C++.

INTRODUCTION

The spatial analysis of thematic variables is of significant importance for the understanding of the characteristics of these variables. Spatial adjacency problems involving irregularly spaced points on a plane usually require intensive calculations and comparisons of distances between data points. For example, the problem of triangulating arbitrary collections of points and the problem of determining the site that is closest to a query point occur in many applications in engineering, isarithmic mapping, spatial process modeling, geographic information systems (GIS), terrain modeling, finite-element analysis, and computational geometry. This seems to involve exhaustively checking the distance between points in the set, but better solutions are possible by constructing the Delaunay triangulation (Delaunay, 1934) and the Voronoi diagram (Voronoi, 1908) of the given points for these applications.

The theory, computations, and applications of Delaunay triangulations and Voronoi diagrams have been described intensively in the literature (Lawson, 1972, 1977; Shamos and Hoey, 1975; Sibson, 1978; Green and Sibson, 1978; Brassel and Reif, 1979; Lee and Schachter, 1980; McCullagh and Ross, 1980; Bowyer, 1981; Watson, 1981; Mirante and Weingarten, 1982; Sloan, 1987; Macedonio and Pareschi, 1991; Kao, Mount, and Saalfeld, 1991; Gold, 1989, 1992; Puppo and others, 1992). Most recently, Aurenhammer (1991) provided a detailed survey and bibliography on the Voronoi diagram and related structures, emphasizing the unified exposition of their mathematical and algorithmic properties. Because the Delaunay triangulation and the Voronoi diagram are geometric duals, it is desirable to discuss briefly the theory and properties of both data structures. For standard and detailed sources, see Aurenhammer (1991) and Preparata and Shamos (1985).

The Voroni diagram, or the Dirichlet tessellation (Dirichlet, 1850) or Thiessen polygons (Thiessen, 1911), is one of the fundamental data structures in computational geometry. It is a union of contiguous polygonal regions whose boundaries are made up of the perpendicular bisectors of the lines joining neighboring points. The Voronoi diagram of N distinct points on the plane divides the plane according to the *nearest-neighbor* rule: each point is associated with the region of the plane closest to it (Aurenhammer, 1991). The region, or the Voronoi polygon, associated with each point is unique, determined by the spatial distribution of the points, and defines the region of influence of that point (Hayes and Koch, 1984; Gold, 1989). On the other hand, the Delaunay triangulation is constructed by connecting the points whose associated Voronoi polygons share a common edge. A Delaunay triangle thus is formed from three adjacent points whose associated Voronoi polygons meet at a common vertex, which is the center of the circumscribed circle of the Delaunay triangle. Figure 1 depicts the duals of the Delaunay triangulation and the Voronoi diagram for a set of twelve randomly distributed points in the Euclidean plane.

This paper describes a fast algorithm, the *Convex Hull Insertion* algorithm, for the construction of Delaunay triangulations and Voronoi diagrams of arbitrary collections of points on the Euclidean plane. The proposed algorithm not only computes both geometric structures, but also builds in a relational data model the topologies of the spatial primitives in

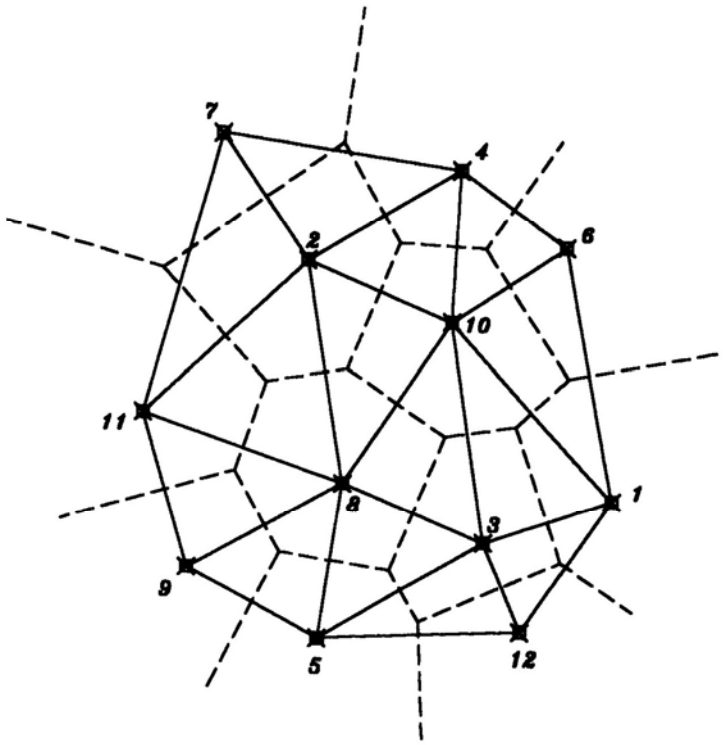


Figure 1. Delaunay triangulations (—) and Voronoi diagrams (– – –) of 12 points in plane.

both structures. The proposed algorithm is Delaunay-based and has been implemented on IBM compatible personal computers (PCs) using Borland C++ 3.1. The remainder of this paper is devoted to the description of the criteria of Delaunay triangulation, technical procedures of the *Convex Hull Insertion* algorithm, and the implementation including topological data structures, data input and output, and empirical examples. Results of various examples up to 50,000 points indicate that the proposed *Convex Hull Insertion* algorithm constructs both Delaunay triangulations and Voronoi diagrams of randomly distributed points in expected linear time, that is the program runs in approximately $O(N)$ for N randomly distributed points.

CRITERIA OF DELAUNAY TRIANGULATION

Lawson (1972) suggested the *max–min angle criterion* to construct triangulations with the *local equiangularity* property: in every convex quadrilateral formed by two adjacent triangles, the replacement of their common edge does not increase the minimum of the six interior angles concerned. Based on the *max–min angle criterion*, Lawson (1977) also gave a *local optimization procedure* (*LOP*) to swap diagonals of a convex quadrilateral for a locally optimal triangulation. Figure 2 illustrates the completion of the *LOP* swapping for a new point inserted into an existing triangulation.

However, the Delaunay triangulation of a set of points is an aggregate of adjoining but nonoverlapping triangles such that the circumcircle of each triangle contains no other point in its interior. This property hereinafter is termed the *Delaunay criterion* in the construction of Delaunay triangulations for a set of distinct planar points. Both Lawson (1977) and Sibson (1978) have observed that the Delaunay triangulation automatically satisfies the *max–min angle criterion*, and, uniquely, is locally equiangular. Intimately, the *max–min angle criterion* implies the *Delaunay criterion* when a triangulation is constructed by applying the *LOP* to all triangles of the triangulation, until no diagonal swapping occurs.. Consequently, the *Delaunay criterion* is used in the following *Convex Hull Insertion* algorithm to construct the Delaunay triangulation and then the Voronoi diagram of a point set.

THE CONVEX HULL INSERTION ALGORITHM

Tsai and Vonderohe (1991) presented a generalized algorithm, the *Convex Hull Insertion* algorithm, for the construction of Delaunay triangulations in the n-dimensional Euclidean space. Here the *Convex Hull Insertion* algorithm is improved further and implemented to construct the Voronoi diagram of a set of planar points as well as the Delaunay triangulation. The improvement expedites the convex hull computation and the incremental insertion of points

by partioning the point set into cells and maintaining spatial primitives in topological data structures.

For a set of $N \geqslant 3$ distinct points in the Euclidean plane, the *Convex Hull Insertion* algorithm involves the following phases:

(1) Partition the point set into N/k cells, that is $\sqrt{N/k}$ equidistant rows and columns, where k is the average number of points per cell and may be selected arbitrarily (default $k = 4$).
(2) Determine the *convex hull* of the whole point set.
(3) Construct the Delaunay triangulation of the *convex hull* by applying *Delaunay criterion.*
(4) Iteratively insert other points, which are not on the *convex hull*, and refine the existing triangulations.
(5) Construct the Voronoi diagram from the Delaunay triangulation of the set.

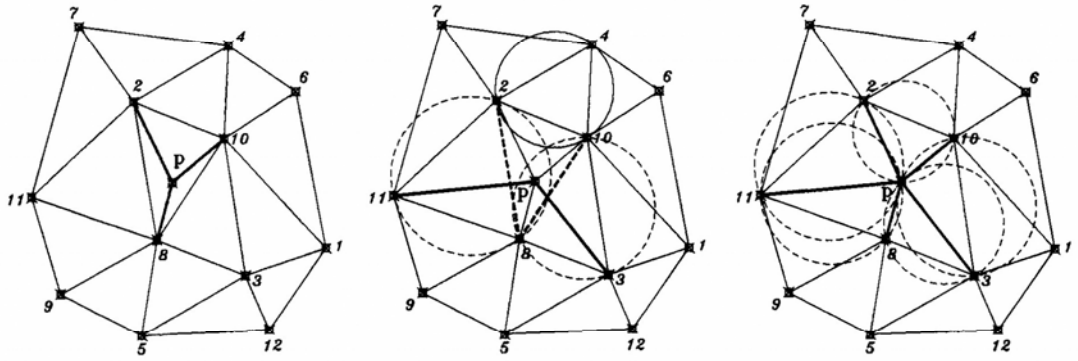
Initial data partitioning

Partitioning the set of points into cells has been used increasingly in geometric algorithms such as Voronoi tessellation and Delaunay triangulation (Shamos and Hoey, 1975; Lee and Schachter, 1980; McCullagh and Ross, 1980; Maus, 1984; Dwyer, 1987; Macedonio and Pareschi, 1991). Generally, the geographical partitioning process is a two-dimensional (2-D) sort that enables fast access to points lying in the proximity of others, thus improving the expected run time of the algorithm. Sorting the points, however, is opulent and superfluous when data points should exist in the same order as what they are to be accessed. Thus it is preferred in run-time programs to leave the points unsorted while partitioning them into cells. A faster alternative is to create a 1-D array or list to store the index of the first point in each cell, then store along with each point the index of next point lying in the same cell (Larkin, 1991). Figure 3 shows the partitioning structures of the 12-point set shown in Figure 1. Partitioning points in this manner accelerates the computation of the *convex hull*, and runs in linear time, that is, $O(N)$, for a uniform distribution of points, but in $O(N^2)$ for the worst possible case (Larkin, 1991). Similar partitioning structures also are used to store the centroids of triangles for quick search of triangles, whose circumcircles enclose the inserted point, in refining the existing triangulations.

Convex hull computation

The *convex hull* of a set of planar points is the natural extreme perimeter of the point set. Obviously, edges of the *convex hull* are part of the Delaunay triangulation of the set. Once the point set was partitioned into cells, its *convex hull* can be computed in approximately $O(N)$ for a uniform set and $O(N^2)$ for the worst possible case (Maus, 1984; Larkin, 1991). Based on the algorithm improved by Larkin (1991), the following procedures commute the *convex hull* of the partitioned set as depicted in Figure 4:

(a) Determine the points with the minimum $x - y$, $x + y$, and maximum $x - y$, and $x + y$ values, respectively. These points are all on the *convex hull* perimeter and lie as near as to the four extreme corners of the set, for example, points 7, 9, 12, and 6 in Figure 4A.
(b) Store these points in a linked circular list in counterclockwise order and remove any redundancy.
(c) For each point **I** and its subsequent point **J** in the list, call the recursive subalgorithm CONVEX(**I**, **J**) to locate all the points on the hull to the right of the line segment $\vec{IJ}$.



(A) Insert new point P and connect to vertices of the enclosing triangle

(B) Apply max-min angle criterion and swap diagonals

(C) Updated triangulations with circumcircles on new triangles

Figure 2. Completion of Lawson's *LOP* for inserting point P into triangulation in Figure 1.

(A) Partitioned cells

(B) CELL structure	
Cell	First Pt
A	5
B	1
C	2
D	4

(C) POINT structure			
Pt	X	Y	Next Pt
1	118.95	72.00	3
2	51.83	123.46	7
3	90.97	62.48	12
4	85.12	143.27	6
5	54.86	41.22	8
6	108.43	126.86	10
7	32.87	150.32	-
8	60.01	75.04	9
9	26.24	56.16	11
10	83.49	110.38	-
11	16.25	90.02	-
12	99.26	43.21	-

Figure 3. Partitioning structures of 12 points in Figure 1.

The recursive subalgorithm CONVEX(**I**, **J**) does:

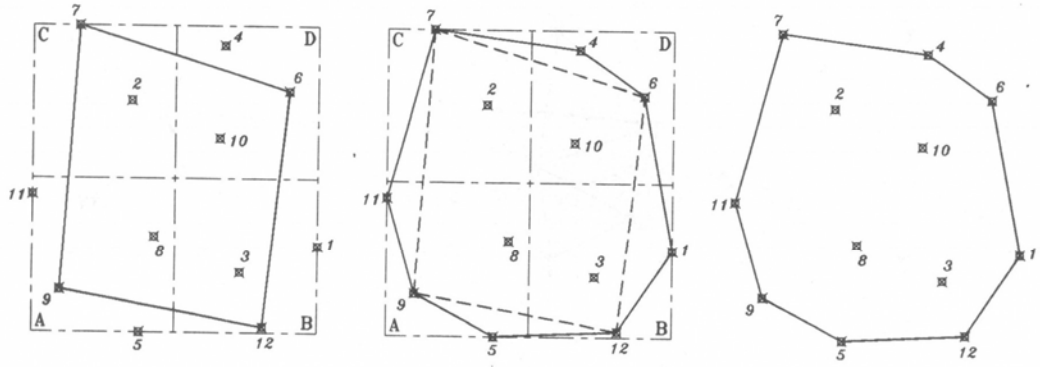
(d) Examine all points lying in the cells that are intersected by or to the right of the line segment $\vec{IJ}$, and locate the point **K** with the largest offset from $\vec{IJ}$, where points to the right of $\vec{IJ}$ are assigned positive offsets and those to the left negative ones.

(e) Test the sign of the largest offset:

(i) if it is positive, insert **K** into the list between points **I** and **J**, and call CONVEX(**I**, **K**) and CONVEX (**K**, **J**).

(ii) if it is zero and **K** lies between **I** and **J**, insert point **K** into the list between points **I** and **J**, and call CONVEX(**I**, **K**) and CONVEX(**K**, **J**).

(iii) else terminate this call to CONVEX.



(A) Initial boundaries 7, 9, 12, and 6

(B) Find hull vertices 11, 5, 1, and 4

(C) The *convex hull*

Figure 4. Completion of *convex hull* computation.

Convex hull triangulation

Knowing the M vertices of the *convex hull* in counterclockwise order, the following procedures compute the Delaunay triangulation of the *convex hull* as shown in Figure 5:

(a) Store the first edge of the *convex hull* in a list of *base edges* which is initialized empty.
(b) For the next *base edge*, check the vertices lying to its left to locate the third point by applying *Delaunay criterion*.
(c) Form the Delaunay triangle and append all internal edges to the list of *base edges* where new edges of the triangle are assigned as from the *from_node* of the *base edge* to the third point, and then to the *to_node* of the *base edge*.
(d) Update the adjacent topologies of edges and triangles in the topological data structures, which are described later in the implementation section.
(e) Repeat steps (b)–(d) to create all Delaunay triangles until all *base edges* are served.

Insertion of other points

All other points within the *convex hull* can be inserted iteratively into the initial Delaunay triangulation of the *convex hull*. By determining the *influence triangulation* of the added point, existing triangulations then are locally refined. The following procedures update the existing triangulations in expected linear time as illustrated in Figure 6:

(a) Determine all of the triangles whose circumcircles enclose the new point by applying *Delaunay criterion*, forming the *influence triangulation* of the new point. Determining such triangles is speeded through the use of a partitioning cell structure for the centroids of triangles and the topological data structures, which keep the adjacent relationships between edges and triangles. That is, locate the first candidate triangle through the partitioning cell structure, and then locate the rest by investigating its neighboring triangles outwardly.
(b) Delete all internal edges shared by two adjacent triangles in the *influence triangulation* of the new point. This is done concurrently with

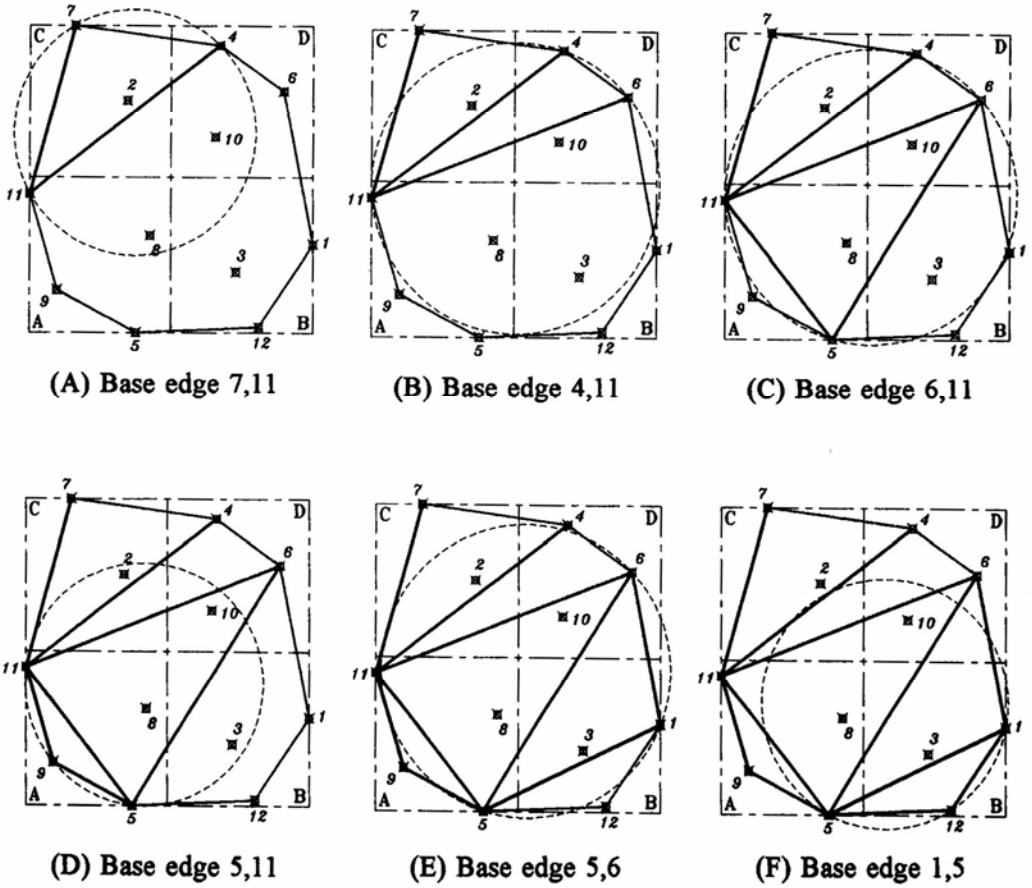


Figure 5. Completion of *convex hull* triangulation.

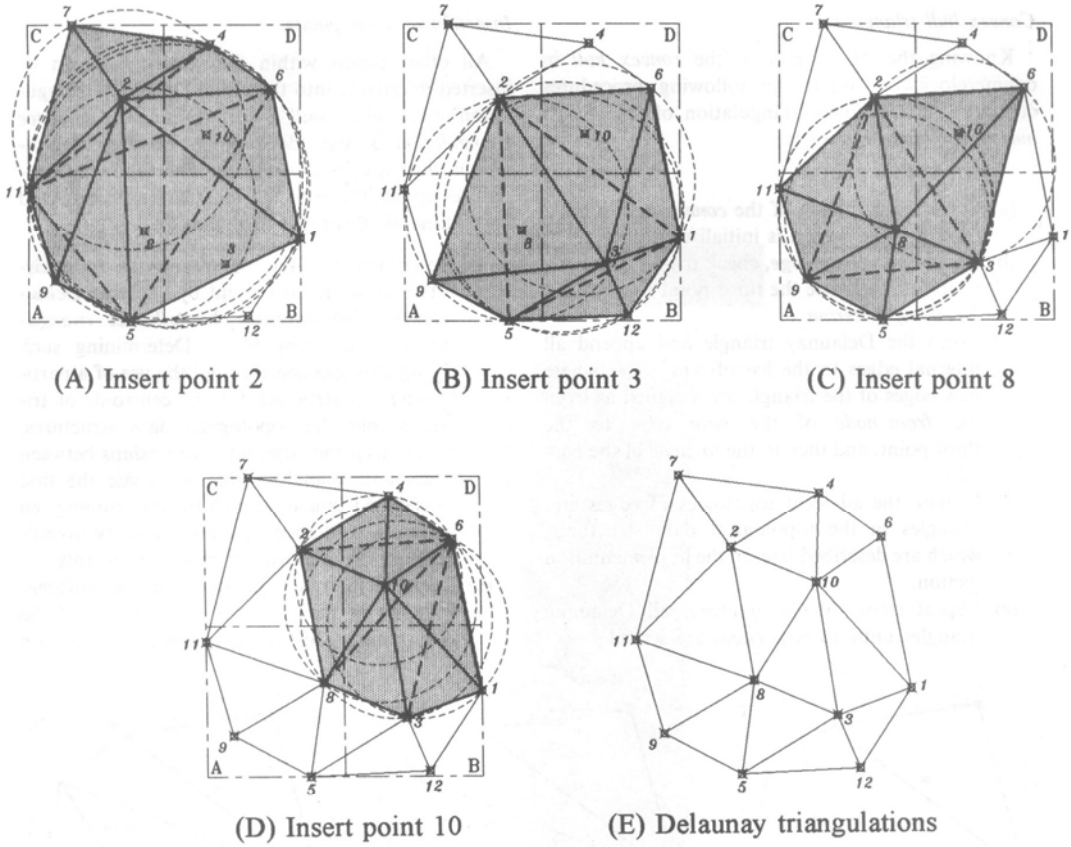


Figure 6. Insertion of other points into existing triangulations (shaded region shows *influence triangulation* of new point).

step

(a) by pushing the edges of the enclosing triangles into a dynamic linked list, and dropping (or popping out of the list) the duplicate edges.
(c) Construct new triangles by connecting the new point to the vertices of each boundary edge of its *influence triangulation*. Note that *Lawson's LOP* is implicitly and automatically involved in this way.
(d) Update the adjacent topologies of edges and triangles that are within or adjacent to the refined *influence triangulation* of the new point.
(e) Repeat steps (a)–(d) until all other points within the *convex hull* are inserted.

Construction of the Voronoi diagram

Because the Voronoi diagram and the Delaunay triangulation are geometric duals, the Voronoi diagram of N sites in the plane can be obtained in $O(N)$ time after the Delaunay triangulation was created. Note that the adjacent topologies of edges and triangles in the Delaunay triangulation are kept in the topological data structures. It is trivial to form the edges of the Voronoi diagram by connecting the circumcenters (which are computed and stored in the topological data structures for the Delaunay triangulation) of the left and right triangles of each Delaunay edge. For those Delaunay edges on the convex hull, a perpendicular line that bisects the boundary edge is computed outwardly from the circumcenter of the only involving triangle. Meanwhile, topologies of spatial primitives in the Voronoi diagram can be easily maintained in similar topological data structures as those for the Delaunay triangulation.

IMPLEMENTATION

The *Convex Hull Insertion* algorithm has been implemented for IBM-compatible PCs using Borland C++ 3.1. Features known for the C++ programming language, such as dynamic memory allocation and class objects, are utilized fully in the implementation to keep the capability and flexibility of the program in handling arbitrary collections of points in the plane. Of the implemented code, the aforementioned partitioning structures and the topological data structures for spatial primitives most significantly dominate and speed up the construction of Delaunay triangulations and Voronoi diagrams accordingly. These topological data structures are

described in detail in the next subsection, followed by the data input and output of the implementation and empirical results of various sets of up to 50,000 points.

Topological data structures

As shown in Figure 7, a C-like description of the data structures implemented in the program encodes the Delaunay triangulation as a set of topological relations of points, edges, and triangles. Coordinates of points are kept only in tuples of POINT, which also maintains the index of the next point in the same partition cell. Tuples of EDGE store the indices of the *from_node* and *to_node* points, and the indices of the *left* and *right* triangles of each edge. For each triangle in the Delaunay triangulation, a tuple of TRIANGLE maintains the indices of the bounded edges and corresponding adjacent triangles, X and Y coordinates of the circumcenter, square of circumradius, and the index of the next triangle whose centroid falls into the same partition cell. Such topological data structures as well as the aforementioned partitioning cell structures for points and triangles improve the performance of the implemented program.

The number of points that can be handled in the implemented program is limited by the amount of memory available on the PC. As shown in Figure 7, the sizes of each tuple of the POINT, EDGE, and TRIANGLE data structures are 12, 16, and 40 bytes, respectively. For N distinct points in the plane, of which M lie on the convex hull, the Delaunay triangulation has $3(N-1)-M$ edges and $2(N-1)-M$ triangles; the Voronoi diagram has $3(N-1)-M$ edges and $2(N-1)$ computed points, including the $2(N-1)-M$ circumcenters of triangles and M auxiliary points that lie outside the convex hull. Since the POINT, EDGE, and TRIANGLE data structures are shared by both Delaunay triangulations and Voronoi diagrams, the amount of memory required for the program can be calculated exactly once the convex hull of the set was determined. For example, up to about 1.34 MB of memory is required for handling each 10,000 points in the set. In the implemented program, tuples of POINT, EDGE, and TRIANGLE are all maintained in the extended memory blocks (above 1-MB) regardless of their sizes and dimensions, leaving the conventional memory (below 640-KB) for partitioning cell structures, runtime dynamic buffering, and general computations.

Data input and output

The program assumes that the spatial coordinates of points are Cartesian or Euclidean. It accepts input data of points from text file in free format, that is, each record (line) of the file lists the X and Y coordinates of a point. On the other hand, there are several output options in the program to present the resulting Delaunay triangulations and Voronoi diagrams:

(1) Build a set of relational files in binary forms for the topologies of data points, edges, and triangles in the Delaunay triangulation;

```
struct POINT                /* Mass data points    */
{
     float  xy[2];          /* X & Y coordinates   */
     long   next;           /* Next POINT in cell  */
};

struct EDGE                 /* Edges in triangulation */
{
     long ft[2];            /* From_node and To_node  */
     long lr[2];            /* Left & Right triangles */
};

struct TRIANGLE             /* Delaunay triangles     */
{
     long  ed[3];           /* Bounded EDGEs' indices */
     long  at[3];           /* Adj TRIANGLEs' indices */
     float xc[2];           /* X & Y of circumcenter  */
     float r2;              /* Square of circumradius */
     long  next;            /* Next triangle in cell  */
};
```

Figure 7. Topological data structures for construction of Delaunay triangulations and Voronoi diagrams.

Edge	From_Pt[1]	To_Pt[1]	Lft_TIN	Rht_TIN	Edge	From_Pt[1]	To_Pt[1]	Lft_TIN	Rht_TIN
1	7	11	1	0	14	3	5	5	4
2	7	4	0	2	15	3	12	6	5
3	2	7	1	2	16	8	2	8	11
4	4	6	0	12	17	8	9	9	7
5	2	11	8	1	18	8	11	7	8
6	2	4	2	3	19	10	6	12	14
7	10	2	11	3	20	8	3	10	4
8	6	1	0	14	21	8	5	4	9
9	3	1	13	6	22	10	4	3	12
10	5	9	0	9	23	10	8	10	11
11	9	11	0	7	24	10	3	13	10
12	1	12	0	6	25	10	1	14	13
13	12	5	0	5					

TIN	Edge1	Edge2	Edge3	Adj_TIN1	Adj_TIN2	Adj_TIN3
1	5	3	1	8	2	0
2	6	3	2	3	1	0
3	22	7	6	12	11	2
4	21	20	14	9	10	5
5	14	15	13	4	6	0
6	15	9	12	5	13	0
7	18	17	11	8	9	0
8	18	16	5	7	11	1
9	17	21	10	7	4	0
10	24	23	20	13	11	4
11	7	23	16	3	10	8
12	19	22	4	14	3	0
13	25	24	9	14	10	6
14	25	19	8	13	12	0

Note: 1 Data points in figure 1 with coordinates listed in figure 3

Figure 8. Edge and triangle topologies of Delaunay triangulation in Figure 1.

(2) Build a set of relational files in binary forms for the topologies of data points, computed points, and edges in the Voronoi diagram;
(3) Create AutoCAD DXF (data exchange format) and script files for graphical representation of the Delaunay triangulation;
(4) Create AutoCAD DXF and script files for graphical representation of the Voronoi diagram;
(5) Display lists of data points, edges, triangles, and associated topologies in the resulting Delaunay triangulation on the screen, or to a printer or a text file; and
(6) Display lists of data points, computed points (including circumcenters of Delaunay triangles and auxiliary points), edges, and associated topologies in the resulting Voronoi diagram on the screen, or to a printer or a text file.

These output options can be set individually or simultaneously in the MS-DOS executable program (CHIDTVD.EXE) as optional command-line arguments, which also include input file name, number of points per partitioning cell, and recording of execution times used in each phase of the proposed algorithm. Listing of spatial primitives and associated topologies in text forms allows manual checking and

verification of the resulting Delaunay triangulations and Voronoi diagrams. Representation of Delaunay triangulations and Voronoi diagrams in relational tables provides interfaces to GIS packages for further spatial analysis upon the topologies and associated nonspatial attributes of spatial primitives. Furthermore, results in AutoCAD DXF and script files not only allow visual verification of the resulting Delaunay triangulations and Voronoi diagrams, but also provide another linkage to computer-aided drafting (CAD) systems where powerful third-party utilities are available for graphical rendering, contouring, automated mapping as well as linking to GIS applications.

Examples

To verify the completeness and authenticity of the proposed *Convex Hull Insertion* algorithm, Figures 8 and 9 list resulting topologies of spatial primitives in the Delaunay triangulations and Voronoi diagrams for the 12-point set in Figure 1. Meanwhile, Figure 10 depicts resulting Delaunay triangulations and Voronoi diagrams of 500 randomly spaced points by importing corresponding DXF or script files in AutoCAD software, in which outer edges of the Voronoi diagram were clipped.

For various data sets of up to 50,000 points, Table 1 shows the execution times in seconds used by the

Point[1]	X	Y	Point[1]	X	Y
1	20.319	121.339	12	90.749	126.506
2	59.270	148.834	13	97.866	88.091
3	72.004	127.435	14	109.208	98.571
4	70.378	56.159	15	-14.368	130.900
5	76.934	45.024	16	63.081	177.077
6	107.975	58.378	17	106.955	149.525
7	36.432	77.571	18	141.484	104.760
8	43.113	97.086	19	31.088	30.563
9	46.358	59.817	20	-1.499	66.380
10	82.349	85.668	21	123.544	47.730
11	60.693	100.056	22	78.063	19.838

Edge	From_Pt[1]	To_Pt[1]	Lft_DPt[2]	Rht_DPt[2]	Edge	From_Pt[1]	To_Pt[1]	Lft_DPt[2]	Rht_DPt[2]
1	1	15	11	7	14	5	4	5	3
2	2	16	7	4	15	6	5	12	3
3	1	2	7	2	16	8	11	2	8
4	12	17	4	6	17	9	7	9	8
5	8	1	11	2	18	7	8	11	8
6	2	3	4	2	19	12	14	6	10
7	11	3	2	10	20	10	4	3	8
8	14	18	6	1	21	4	9	5	8
9	13	6	1	3	22	3	12	4	10
10	9	19	5	9	23	10	11	8	10
11	7	20	9	11	24	13	10	3	10
12	6	21	1	12	25	14	13	1	10
13	5	22	12	5					

Notes: 1 Computed points, including the circumcenters (1-14) of Delaunay triangles in figure 8 and the outer vertices (15-22) of the Voronoi edges which fall outside the triangulation

2 Data points as shown in figure 1 with coordinates listed in figure 3(C)

Figure 9. Point and edge topologies of Voronoi diagram in Figure 1.

Convex Hull Insertion program, running on a Hewlett Packard Vectra 486/66U PC. The execution time required for the program depends on the distribution and size of the point set, and the operating environment (including the type of platform, disk operating system, memory and memory manager) under which the program runs. The higher level and the faster the CPU (central processing unit), the less execution time used for the program. For example, it takes about 90 sec for constructing Delaunay triangulations and Voronoi diagrams of 10,000 points on an 80486 DX2/66 MHz PC, but it takes 615 sec on an 80386 SX/20 MHz PC. Although it may take $O(N^2)$ time in the worst case for N distinct planar points, the timing statistics in Table 1 show that the *Convex Hull Insertion* program constructs the Delaunay triangulation, the Voronoi diagram, and associated topologies in approximately linear time, that is, the overall run-time complexities of the algorithm are $O(N^{1.032})$ for Delaunay triangulations and $O(N^{1.008})$ for Voronoi diagrams accordingly.

CONCLUSIONS

This paper has presented a nearly linear-time algorithm, the *Convex Hull Insertion* algorithm, for the construction of Delaunay triangulations and Voronoi diagrams of arbitrary collections of points in the Euclidean plane. The algorithm takes benefits from partitioning data points into cells and from maintaining spatial primitives in topological data structures. The algorithm has been implemented on IBM compatible PCs using Borland C++ 3.1; the program can handle large sets of points as long as sufficient memory is available on the PC. Empirical results from various sets of up to 50,000 points show that the program runs in approximately $O(N)$ for N randomly spaced points. Moreover, the program delivers several output options; thus provides linkages to GIS and CAD systems for advanced spatial analysis and mapping applications.

An archived file, CHIDTVD.ZIP for this work, is available by anonymous ftp from shelf.ersc.wisc.edu (128.104.83.146) in the *pub* directory subject to nonprofit and noncommercial purposes. The archived file contains several example data files, a brief document (CHIDTVD.DOC), and the MS-DOS executable program (CHIDTVD.EXE) of the *Convex Hull Insertion* algorithm by the author. A brief help for the usage of the CHIDTVD.EXE program can be displayed by running the program with either "?" or "/h" option.

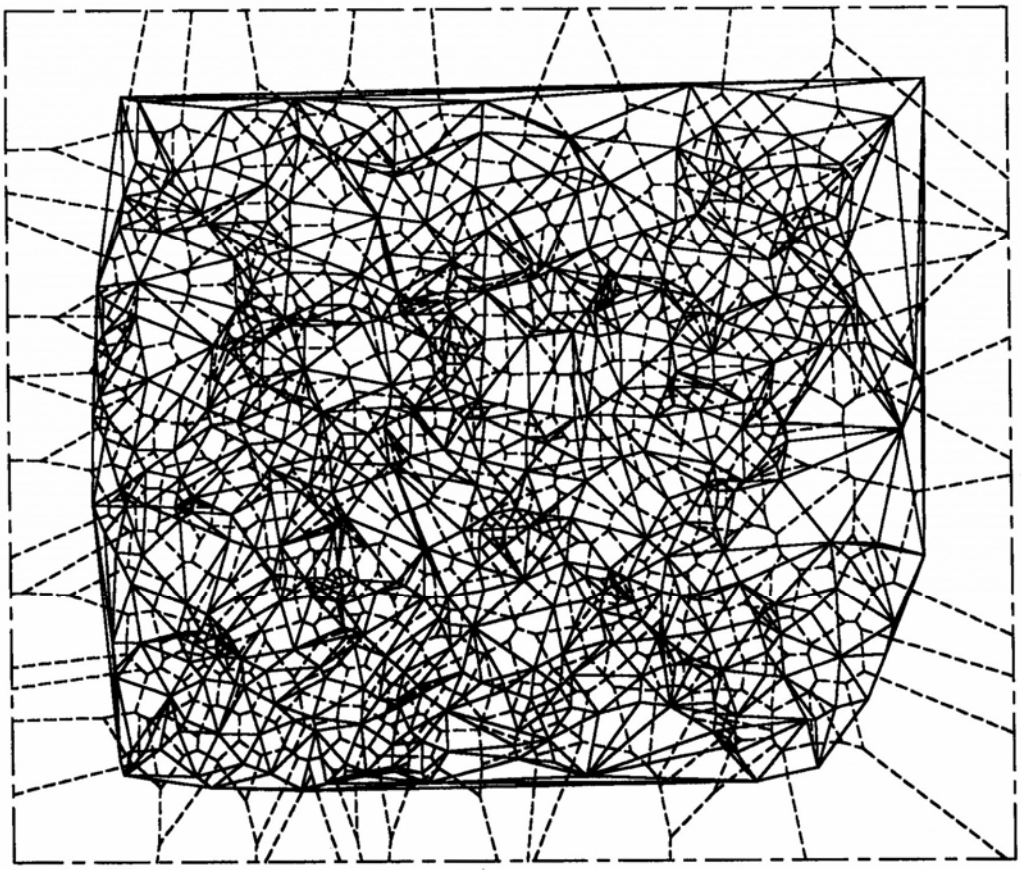


Figure 10. Delaunay triangulations (—) and Voronoi diagrams (– – –) of 500 points by proposed *Convex Hull Insertion* algorithm.

Table 1. Execution times[1] of *Convex Hull Insertion* algorithm for constructing Delaunay triangulations and Voronoi diagrams of various sets of randomly spaced points on plane

Number of Points	Delaunay Triangulations					Voronoi Diagrams
(N)	Initial Data Partitioning (1)	Convex Hull Computation (2)	Convex Hull Triangulation (3)	Other Points Insertion (4)	Total (5) = (1)+(2)+(3)+(4)	(6)
1,000	0.085	0.037 (16)[2]	0.014	8.203	8.339	0.121
2,000	0.171	0.062 (20)	0.017	16.497	16.747	0.247
3,000	0.266	0.083 (17)	0.014	25.485	25.848	0.376
4,000	0.369	0.110 (16)	0.014	34.562	35.055	0.505
5,000	0.448	0.129 (19)	0.016	42.880	43.473	0.620
6,000	0.581	0.149 (15)	0.014	52.043	52.787	0.758
7,000	0.679	0.174 (19)	0.015	60.349	61.217	0.892
8,000	0.774	0.190 (22)	0.019	68.965	69.948	1.018
9,000	0.869	0.224 (25)	0.022	77.486	78.601	1.149
10,000	1.059	0.251 (26)	0.022	88.429	89.761	1.274
20,000	2.742	0.492 (20)	0.016	195.180	198.430	2.557
30,000	3.473	0.781 (26)	0.027	278.162	282.443	3.736
40,000	3.590	0.978 (31)	0.029	348.396	352.993	4.964
50,000	4.519	1.195 (32)	0.033	438.788	444.535	6.205
a[3]	1.052	0.876	1.015	1.028	**1.032**	**1.008**

[1]CPU time in seconds, excluding data input/output, for a Hewlett Packard Vectra 486/66U PC (Intel 80486 DX2/66 with 8 Mbytes RAM, under MS-DOS 5.0 and QEMM 386 6.0).
[2]Number of points located on the convex hull of the point set.
[3]Overall run-time complexity in $O(N^a)$, 'a' values obtained by least-squares fitting.

REFERENCES

Aurenhammer, F., 1991, Voronoi diagrams—a survey of a fundamental geometric data structure: ACM Computing Surveys, v. 23, no. 3, p. 345–405.

Bowyer, A., 1981, Computing Dirichlet tessellations: Computer Jour., v. 24, no. 3, p. 162–166.

Brassel, K. E., and Reif, D., 1979, Procedure to generate Thiessen polygons: Geographical Analysis, v. 11, p. 289–303.

Delaunay, B. N., 1934, Sur la sphère vide: Bull. Acad. Science USSR VII: Class. Sci. Math., p. 793–800.

Dirichlet, G. L., 1850, Über die Reduction der positiven quadratischen Formen mit drei unbestimmten gazen Zahlen: Jour. Reine Angew. Math., v. 40, p. 209–227.

Dwyer, R. A., 1987, A fast divide-and-conquer algorithm for constructing Delaunay triangulations: Algorithmica, v. 2, p. 137–151.

Gold, C. M., 1989, Spatial adjacency—a general approach: Proc. Auto-Carto 9, Baltimore, Maryland, p. 298–312.

Gold, C. M., 1992, An object-oriented dyamic spatial model, and its application in the development of a user-friendly digitizing system: Proc. 5th Intern. Symposium on Spatial Data Handling, v. 2, Charleston, South Carolina, p. 495–504.

Green, P. J., and Sibson, R., 1978, Computing Dirichlet tessellations in the plane: Computer Jour. v. 21, no. 2, p. 168–173.

Hayes, W. B., and Koch, G. S., 1984, Computing and analyzing area-of-influence polygons by computer: Computers & Geosciences, v. 10, no. 4, p. 411–430.

Kao, T., Mount, D. M., and Saalfeld, A., 1991, Dynamic maintenance of Delaunay triangulations: Proc. Auto-Carto 10, Baltimore, Maryland, p. 219–233.

Larkin, B. J., 1991, An ANSI C program to determine in expected linear time the vertices of the convex hull of a set of planar points: Computers & Geosciences, v. 17, no. 3, p. 431–443.

Lawson, C. L., 1972, Generation of a triangular grid with application to contour plotting: California Institute of Technology, Jet Pollution Laboratory, Technical Memorandum No. 299, unpaginated.

Lawson, C. L., 1977, Software for C^1 surface interpolation, *in* Rice, J., ed., Mathematical Software III: Academic Press, New York, p. 161–194.

Lee, D. T., and Schachter, B. J., 1980, Two algorithms for constructing a Delaunay triangulation: Intern. Jour. Computer and Information Sciences, v. 9, no. 3, p. 219–242.

Macedonio, G., and Pareschi, M. T., 1991, An algorithm for the triangulation of arbitrary distributed poins: applications to volume estimate and terrain fitting: Computers & Geosciences, v. 17, no. 7, p. 859–874.

Maus, A., 1984, Delaunay triangulation and the convex hull of n points in expected linear time: BIT, v. 24, p. 151–163.

McCullagh, M. J., and Ross, C. G., 1980, Delaunay triangulation of a random data set for isarithmic mapping: The Cartographic Jour., v. 17, no. 2, p. 93–99.

Mirante, A., and Weingarten, N., 1982, The radial sweep algorithm for constructing triangulated irregular networks: IEEE Computer Graphics and Applications, v. 2, no. 3, p. 11–21.

Preparata, F. P., and Shamos, M. I., 1985, Computational geometry: an introduction: Springer-Verlag, New York, 398 p.

Puppo, E., Davis, L., DeMenthon, D., and Teng, Y. A., 1992, Parallel terrain triangulation: Proc. 5th Intern. Symposium on Spatial Data Handling, v. 2, Charleston, South Carolina, p. 632–641.

Shamos, M. I., and Hoey, D., 1975, Closest-point problems: Proc. 16th Annual Symposium on the Foundations of Computer Science, IEEE, p. 151–162.

Sibson, R., 1978, Locally equiangular triangulations: Computer Jour., v. 21, no. 3, p. 243–245.

Sloan, S. W., 1987, A fast algorithm for constructing Delaunay triangulations in the plane: Advanced Engineering Software, v. 9, no. 1, p. 34–55.

Thiessen, A. H., 1911, Precipitation average for large area: Monthly Weather Review, v. 39, p. 1082–1084.

Tsai, V. J. D., and Vonderohe, A. P., 1991, A generalized algorithm for the construction of Delaunay triangulations in Euclidean n-space: Proc. GIS/LIS '91, v. 2, Atlanta, Georgia, p. 562–571.

Voronoi, M. G., 1908, Nouvelles applications des parameters continus à la thèorie des formes quadratiques: Jour. Reine Angew. Math., v. 134, p. 198–287.

Watson, D. F., 1981, Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes: Computer Jour., v. 24, no. 2, p. 167–172.