PERGAMON

Computers & Geosciences 29 (2003) 523–530

COMPUTERS & GEOSCIENCES

www.elsevier.com/locate/cageo

# Delete and insert operations in Voronoi/Delaunay methods and applications ☆

Mir Abolfazl Mostafavi[a,*], Christopher Gold[b], Maciej Dakowicz[b]

[a] *Geomatics Department, Laval University, Quebec City, Quebec G1 K 7P4, Canada*
[b] *Department of Geo-Informatics, Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong*

Received 27 March 2001; received in revised form 1 November 2002; accepted 22 November 2002

**Abstract**

This paper presents simple point insertion and deletion operations in Voronoi diagrams and Delaunay triangulations which may be useful for a wide variety of applications, either where interactivity is important, or where local modification of the topology is preferable to global rebuilding. While incremental point insertion has been known for many years, point deletion is relatively unknown. The robustness and efficiency of a new algorithm are described. A variety of potential applications are summarized, and the included computer program may be used as the basis for many new projects.


*Keywords:* Insertion; Deletion; Algorithms; Voronoi; Delaunay triangulation

## 1. Introduction

Delaunay triangulations have been well known in the geosciences for many years, as are triangulated terrain models. Somewhat less known are the applications of Voronoi diagrams, although there are many scientists who consider them to be "a fundamental geometric data structure" (Aurenhammer, 1987)—for example the work of Sambridge et al., (1995). The Voronoi and Delaunay structures are the duals of each other (see Fig. 1), and so the construction of one automatically creates the structure of the other—although the advantages of each are not necessarily apparent when working in the other mode.

While the "batch" construction of the Delaunay triangulation of a set of data points, for example, is well known, the ability to insert and delete individual points at will is often desired, but not usually available. In computer science terminology, we would be using "dynamic" data structures and algorithms, as modifications to the structure could be made without reconstructing the whole thing each time. It should be noted that this use of "dynamic" does not mean that the points are considered to be moving—that is another property, usually known as "kinetic" (Guibas et al., 1991).

The need to modify the set of data points is most obvious in the case of an on-screen interactive demonstration, and that is the illustration given in this paper. The accompanying computer program is written in Delphi (essentially visual object-oriented Pascal) as we have found this to satisfy our research needs better than the other choices—it compiles very fast, executes as fast as C/C++, has an excellent visual interface and development environment, and is considerably easier to master than C++—unless one is a professional programmer! Our second choice would have been Java, but at present this is still too slow. In both these cases the object-oriented structuring is ideal for dynamic data structures. The popularity of Delphi varies widely, being less in the US and more in South America and Europe.

☆ Code on server at http://www.iamg.org/CGEditor/index.htm

*Corresponding author.
*E-mail addresses:* mir-abolfazl.mostafavi@scg.ulaval.ca (M.A. Mostafavi), christophergold@voronoi.com (C. Gold).

0098-3004/03/$ - see front matter 
doi:10.1016/S0098-3004(03)00017-7

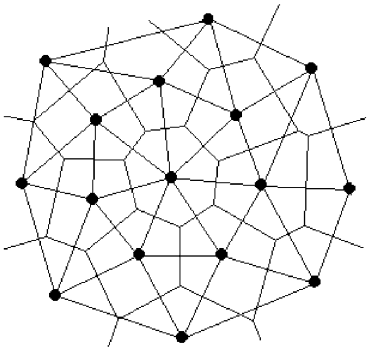

Fig. 1. Delaunay triangulation and dual Voronoi diagram.

Nevertheless, it is largely self-documenting, and we hope that the methods presented here may either be used directly, or be readily translated.

Whereas a variety of algorithms exist to create a Delaunay triangulation in "batch" mode (for an excellent evaluation see Shewchuk, 1996) the incremental algorithm is both one of the earliest and one of the most robust, although not the fastest. It also forms the basis of a dynamic algorithm, consisting of insertion and deletion of individual points. Insertion consists of finding the triangle enclosing the new point (often by walking through the existing triangulation), splitting this triangle into three, and recursively checking if each edge (or pair of triangles) conforms to the Delaunay criterion. If not, the diagonal of the triangle pair is swapped, and checking continues. Deletion is largely the reverse of insertion, but has a variety of specific difficulties.

For very large data sets the least efficient step is to find the enclosing triangle. Gold et al. (1977) first described the simple "walk" algorithm, which is expected to use sqrt(n) time for each point, where n is the number of points. Improved methods can reduce this—the most practical being to store a small proportion of the points, search them to find the closest to the new point, and use this to make an initial estimate of the nearest triangle. For this paper we have limited ourselves to the simple walk.

Lastly, the code presented here does not attempt to partition a very large triangulation into smaller portions for storage on disk if there is insufficient memory. This is a messy problem, and beyond the scope of this paper.

## 2. Background and implementation

The Delaunay triangulation was introduced by Voronoi (1908) for sites that form a lattice and was extended by Delaunay (1934) for irregularly placed sites by means of empty circle methods. There is an excellent introduction to the Delaunay triangulation and its properties in Aurenhammer (1991) and Okabe et al. (2000).

There are several useful properties of the Delaunay triangulation, which make it distinct from other triangulation methods. If we consider a set of points in two-dimensional Euclidean space, the Delaunay triangulation of the set of points is the triangulation of the point set with the property that no point falls in the interior of the circumcircle of any triangle in the triangulation. If we connect the centres of these circles between pairs of adjacent triangles we get the Voronoi diagram, the dual of the Delaunay triangulation, with one Voronoi edge associated with each Delaunay edge. The Voronoi diagram consists of cells around the data points such that any location in a particular cell is closer to that cell's generating point than to any other.

A variety of algorithms for Voronoi/Delaunay construction have been described in the literature—see Aurenhammer (1991) and Okabe et al. (2000). The most efficient of these are "batch" methods, where the point set is known in advance—for example see Fortune (1992). In many applications, however, the diagram must be adjusted interactively, and the simple incremental insertion algorithm is more appropriate. The best-known paper describing the algorithm is Guibas and Stolfi (1985), although Lawson (1977), Green and Sibson (1977), Gold et al. (1977) and others used similar techniques.

Of particular interest is Guibas and Stolfi's use of the "Quad-Edge" data structure to store the network, rather than the alternative of a set of triangles with pointers to their three vertices and adjacent triangles. Shewchuk (1996) has shown that, not only does the Quad-Edge structure take approximately twice the storage of the triangle structure, but also the updating of the structure may take twice the time. We have stayed with the Quad-Edge structure for its elegance and simplicity, and because it manages the primal and dual representations simultaneously. Shewchuk also discusses the awkward problem of the robustness of algorithms due to the computer's finite-precision arithmetic, but in our experience this appears to be a problem only when working with constrained triangulations. For the simple point insertion case, the only robustness issue appears to be to avoid inserting two points at the same (or near-identical) location. A simple tolerance check (a visible disk in our program) is usually used after finding the enclosing triangle.

In the insertion algorithm, an initial "frame" triangle is created, and points are inserted individually by:

(1) Performing a "walk" from some previous triangle to the triangle containing the new point $P$;
(2) Splitting that triangle into three new triangles, each having $P$ as a vertex; and
(3) Ensuring that each edge is Delaunay by recursively testing each edge (triangle pair) and switching the diagonal if necessary.

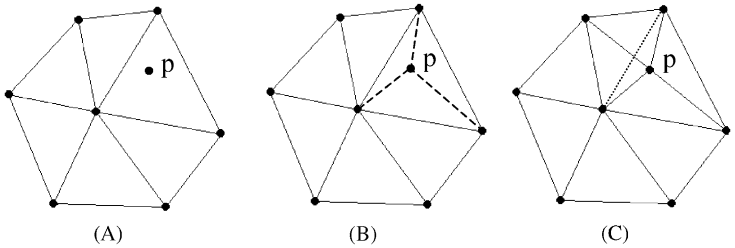


Fig. 2. Steps to insert new point in Delaunay triangulation after "Walk" operation: (A) initial triangulation; (B) splitting enclosing triangle; (C) "Swap" operation.

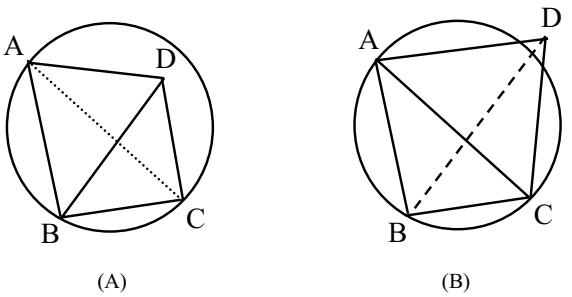


Fig. 3. Empty circumcircle test.

Fig. 2 illustrates point location, point insertion, and edge switching. Fig. 3A shows the case of vertex $D$ being inside the circumcircle of triangle $(A, B, C)$, and thus being a neighbour of $B$, and Fig. 3B shows vertex $D$ outside—with the result that the diagonal $(B, D)$ has been switched and $D$ is no longer a neighbour of $B$.

The walk operation uses the determinant $D$ (Eq. (1)) to test if the point $P$ is to the left of directed edge $AB$ of some initial triangle. If so, the next clockwise edge from $B$ is used, if not the next clockwise edge from $A$—and the test is repeated until three successive edges have $P$ on the left. This is, theoretically, the least efficient part of the algorithm, and more efficient search algorithms may be used for extremely large data sets.

$$D(A,B,P) = \begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_P & y_P & 1 \end{vmatrix}. \quad (1)$$

## 3. The storage data structure

Our algorithms for insertion and deletion of points in the Delaunay triangulation and the Voronoi diagram use the Quad-Edge data structure of Guibas and Stolfi (1985) as it is simple to implement and allows us an edge-to-edge navigation through the mesh by means of its algebraic operations. As there is no distinction between the primal and dual representations we can represent Delaunay and Voronoi edges simultaneously with the same data structure. The main disadvantage of the method is that storage costs are relatively high, although this is becoming less of a practical issue than it used to be. Shewchuk (1996) discussed this in some detail.

The Quad-Edge data structure captures all the topological information of the subdivision of a surface. Each complete Quad-Edge is composed of four branches, which in our object-oriented implementation are connected together in anticlockwise order by means of the $R$ (or Rot) field. Each branch has two additional pointers: the $V$ (or Vertex) field points to geometric or attribute objects (a vertex or face) and the $N$ (or Next) field points to the next anticlockwise edge around any vertex or face. The low-level operations that use the $R$, $N$ and $V$ fields (Fig. 4A) are:

(a) "Sym", which moves the pointer from the current edge $Q$ to the edge that faces in the opposite direction. Thus $Q\,\mathrm{Sym} = QRR(Q\,Org = QV$, the vertex connected to $Q$, and $Q\,\mathrm{Dest} = QRRV$, the vertex opposite $Q$).
(b) "Onext" moves the pointer from the current edge $Q$ to the next edge anticlockwise around $Q\,\mathrm{Org}$. Thus $QO\,\mathrm{next} = QN$.
(c) "Oprev" moves the pointer from the current edge $Q$ to the next clockwise edge around $Q\,\mathrm{Org}$. This can be achieved by saying that $Q\,\mathrm{Oprev} = QRNR$.
(d) "Lnext" moves the pointer from the current edge Q to the next clockwise edge around Q.Dest. Thus $Q\,\mathrm{Lnext} = Q\,\mathrm{Sym}\,\mathrm{Oprev} = QRRRNR$.
(e) "Lprev" moves the pointer from the current edge Q to the next anticlockwise edge around $Q\,\mathrm{Dest}$. Thus $Q\,\mathrm{Lprev} = Q\,\mathrm{Sym}\,\mathrm{Onext} = QRRN$.

To create and modify the graph only two basic functions are used: "Make-Edge" and "Splice". Make-Edge (Fig. 4B) is used to create a new edge ready to be connected into the current graph. Splice (Fig. 4C) is used to connect edges together or disconnect them from each other and takes two edges $(a, b)$ as arguments. In the first case, going from right to left in Fig. 4C, Splice$(a, b)$ connects two separate loops around the common node,

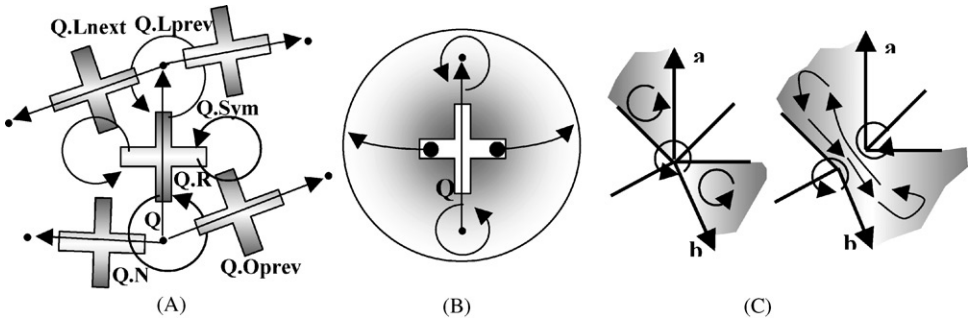


Fig. 4. Quad-Edge data structure: (A) Navigation operations, (B) Make-Edge and (C) Splice.

joining the two nodes together, and at the same time splitting the loop around the common face. In the second case, going from left to right in Fig. 4C, Splice$(a, b)$ splits the common node and at the same time merges the faces at the left side of the edges. Calling Splice twice with the same arguments results in the initial configuration. For more details see Guibas and Stolfi (1985).

## 4. Our algorithm for point deletion from Delaunay triangulation

We mentioned that one of the properties of a dynamic data structure is that it allows deletion of objects from the structure with only local updating. The problem of removing points from the Delaunay triangulation is mentioned briefly in the literature as the inverse of the incremental insertion algorithm, but only Heller (1990) had previously described it. In his method the set of neighbouring points of the point $P$ to be deleted are examined, as triples, (or potential triangles) in anti-clockwise order. The potential triangle with the smallest circumcircle is removed by swapping the edge (the inverse of the insertion algorithm described previously) to reduce the set of neighbours of $P$ by one, and the process is repeated until only three triangles are left. Again as the inverse of the insertion algorithm, $P$ is removed and the three triangles merged into one.

In our own work we considered the empty circumcircle property of a Delaunay triangulation, and deduced that any triangle removed by the above method (often called an "ear") must be empty of other vertices except $P$ (which is being removed). (An ear is a triple of adjacent neighbouring vertices to $P$ which is convex outwards from $P$.) Thus any ear that has no points in its circumcircle may be removed immediately. Our work was described briefly (not the full algorithm) in a preliminary report in 1998 and was published in Gold (2000). Concurrently, Devillers (1999) showed that Heller's test was wrong, and devised an ear elimination algorithm based on the power of a point with respect to a circle that was able to order the vertices in the sequence required for their elimination. As will be described below, our algorithm is simpler, but becomes less efficient as the number of neighbours increases. However, for most data sets the two approaches are equally efficient. Despite the fact that point deletion often has the potential to break down in degenerate cases due to the limitations of floating-point precision, we have tested our method with a variety of nasty cases (such as 1000 points in a circle, with point $P$ inserted and deleted in the centre!) without problems.

The algorithm goes as follows:

(1) Locate the point to be deleted using the "Walk" operation mentioned previously.
(2) Each triple of vertices connected to $P$ is considered as a "potential triangle" $T$, with vertices $v1, v2$ and $v3$. T is rejected immediately if $D(v1, v2, v3)$ (as defined above) is negative (meaning that T forms a re-entrant, not an ear, of the polygon formed by the neighbours of $P$). It is also rejected if $D(v1, v3, P)$ is negative (meaning that $T$ encloses $P$).
(3) If the ear satisfies these conditions, the remaining neighbouring points of $P$ are tested to see if they fall inside the circumcircle of $T$, using the test that $H(v1, v2, v3, P) > 0$. (Eq. (2)) If none of the remaining neighbours fall inside then the ear will be Delaunay, and may be removed by switching the diagonal.

$$H(A,B,C,D) = \begin{vmatrix} x_A & y_A & x_A^2+y_A^2 & 1 \\ x_B & y_B & x_B^2+y_B^2 & 1 \\ x_C & y_C & x_C^2+y_C^2 & 1 \\ x_D & y_D & x_D^2+y_D^2 & 1 \end{vmatrix}. \quad (2)$$

(4) Steps (2) and (3) are repeated until only three neighbours remain, in which case the three triangles are merged into one and $P$ is deleted.

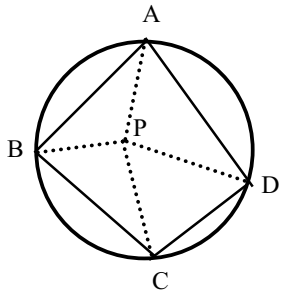


Fig. 5. Degenerate case where points $A$, $B$, $C$ and $D$ are on same circle.

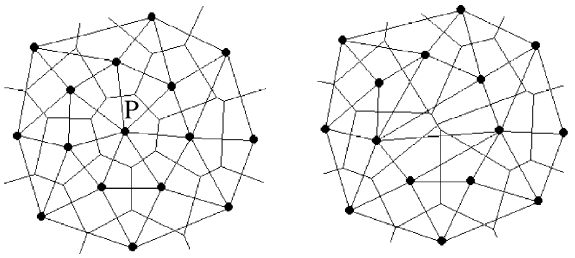


Fig. 6. Delaunay triangulation and Voronoi diagram before (A) and after (B) deleting point $P$.

This method is effective in all cases where no more than three neighbours of $P$ are cocircular. However, due to the limitations of floating-point arithmetic, points on this circle may have very small positive values of $H$, leaving us with more than three neighbours at the termination of the algorithm (Fig. 5). Fortune (1992) showed that the flipping algorithm for point insertion could be implemented using floating-point arithmetic to produce an "approximate Delaunay triangulation". This means that the triangulation has almost-empty circumcircles: any site lying inside a circumcircle is only "slightly" inside. The meaning of "slightly" depends upon the relative error of floating-point arithmetic, the number of sites, and the distance of the sites from the origin, but not on any geometric properties of the configuration of the sites.

Following Fortune (1992) we imagine that the circumcircle is shrunk by a very small amount, which can be achieved by testing if $H >$ epsilon, where epsilon is a very small positive value (currently $10^{-18}$ in function "Incircle"). While using a tolerance in operations of this type is undesirable, its very small value has no effect on the operational robustness of the algorithm. Fig. 6 shows the result of deleting point $P$ from a small triangulation.

## 5. Algorithm comparison

Devillers (1999) showed that ears might be removed from the set of neighbours of $P$ in order of the power of $P$ (Aurenhammer, 1987) with respect to the circumcircle of the ear. The power of $P$ with respect to the ear with vertices $v1, v2$ and $v3$ is simply $H(v1, v2, v3, P)/D(v1, v2, v3)$. Since it is necessary to select the ears for removal in order of their power, a priority queue data structure is required. This data structure saves ears in the order they are supplied, and returns them in the order "smallest first". He showed that the ear that has the smallest power is guaranteed to be Delaunay, and may thus be removed. The algorithm may then be stated as:

(1) Put the valid ears around the point in the priority queue in ascending order of their power.
(2) While the number of ears in the priority queue is greater than three, take the first ear from the list and swap the diagonal to form a triangle.
(3) Remove the ears previous to and next to the treated ear from the priority queue.
(4) Compute the power for the two new ears.
(5) Update the priority queue for these ears.
(6) Repeat steps 2 to 5 until three ears remain in the list.
(7) Splice the point from the three remaining points.
(8) Free the point.

For more details see Devillers (1999). It can be seen that the method is somewhat more complex to implement, but the use of the priority queue improves its efficiency from $O(k^2)$ to $O(k \log k)$, where $k$ is the number of neighbours of $P$. Once we were aware of Devillers' work we made comparisons of the number of incircle tests in our method with the number of power tests in Devillers'. The results are shown in Fig. 7. As expected, our approach was significantly less efficient for large numbers of neighbours, but for up to seven or eight neighbours there was little difference. The average

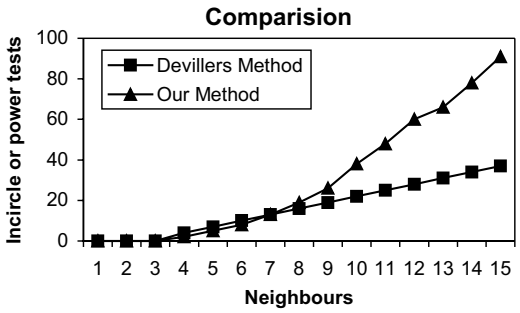


Fig. 7. Comparison between our method and Devillers' method for point deletion.

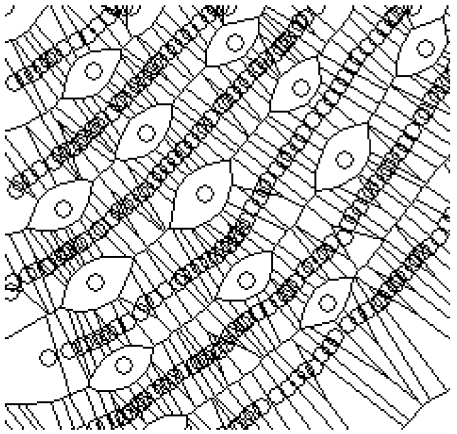

Fig. 8. Voronoi diagram for set of linearly distributed points.

number of neighbours is always six, excluding boundary conditions, but where data is in strings, such as lines or circles, the number of neighbours of an intermediate point between two lines, or at the centre of a circle, may be much larger. We conclude that for most applications our method suffices and is simpler, and we hope that the accompanying code is found useful for many applications. However, in applications where data is highly anisotropic in spatial distribution, such as along ships' tracks, (Fig. 8), it is advisable to implement Devillers' method.

## 6. Applications

In practice, dynamic Voronoi/Delaunay techniques are of interest in a variety of applications. Simple insertion techniques are appropriate for basic geological mapping problems, where each data point has the value of the observed rock type, for example. Removal of the boundaries of adjacent Voronoi cells with the same rock type produces a preliminary geological map (Okabe et al., 2000). Surface runoff and groundwater flow modelling, using the Integrated Finite Difference Method (IFDM), may be based on flow between the Voronoi cells (Lardin, 1999), as the approach may be appropriate for both object and field data. Gold et al. (1996) used the Voronoi/Delaunay relationships to develop a rapid digitizing method for polygonal data. A simple test based on the work of Amenta et al. (1998) and Gold (1999), and improved in Gold and Snoeyink (2001), allows the extraction of the skeleton, or medial axis transform (Blum, 1967) from Voronoi/Delaunay diagrams constructed from contour data, as well as the extraction of connected curves from unordered input points. This led to improved terrain modelling (Thibault and Gold, 2000), where the skeleton points along ridges and valleys may have elevations estimated, based on circumcircle ratios. The crust and skeleton, as well as circumcircles, are provided as display options in the accompanying program.

Point deletion, as described here, extends the range of available operations. Most obviously, the data set may be modified as well as created by adding and deleting individual points. This is often extremely useful to maintain the connectivity of features, for example, when they were originally insufficiently sampled. ''Area-stealing'' or Sibson'' or ''Natural-neighbour'' interpolation (Sibson, 1981; Gold, 1989 and others) has been shown to be an effective method for precise interpolation, especially for highly anisotropically distributed data points, such as those in Fig. 8. In principle it involves the insertion of a sampling point ($P$ in Fig. 6A) into the data set; detection of its Voronoi neighbours and the calculation of their areas; deletion of point $P$ (Fig. 6B) and the recalculation of the areas. This gives the areas stolen by $P$ from each of the neighbours—which form the weights used to estimate $P$'s elevation, using a simple weighted average. (See Gold, 1989 for details.) This method gives a continuous smooth surface that precisely matches the data elevations. With an efficient point deletion function this is now easy to perform.

The work of Thibault and Gold (2000) describes a method for shape generalization (as in curve smoothing) based on the crust and skeleton, and then retracting the minor branches. This relies on our point deletion algorithm, and re-insertion, to simulate the movement of points towards the smoother curve. Gold and Condal (1995) describe the use of the moving-point Voronoi diagram to simulate the movement of a boat, for collision detection. This can also be achieved by point deletion and insertion—and depth soundings, based on the area-stealing method, may be performed at the same time. Finally, we are currently completing work on simulating global tides (Mostafavi and Gold, 2002) by the movement of individual Voronoi cells representing fixed-mass packets of water, using the Free-Lagrange approach of Fritts et al. (1985).

The software provided with this article is intended to illustrate some interactive applications, in the hope that the reader may be motivated to extend this to fit his/her own needs, using the functions provided. The basic system gives the ability to insert and delete points, while maintaining the Delaunay triangulation—and, at the same time, displaying the dual Voronoi diagram. This tool, in our experience, is a sufficiently interesting ''toy'' to allow the development of new ideas and applications. The operation is simple—a left-button mouse click adds a point at that location, and a right-click deletes the closest point (i.e. the point whose Voronoi cell contains the cursor). One final display option is given: following

Gold (1999), the Delaunay and Voronoi edge-pairs may be classified as "crust" or "skeleton", allowing the generation of connected curves of points, and the display of the medial axes between them—useful for generalization and terrain representation.

## 7. Conclusions

Simple point Voronoi diagrams and Delaunay triangulations have many properties that are highly desirable for spatial analysis and modelling applications. Their relatively low usage may be attributed partly to unfamiliarity, partly because incremental modification is rarely available, and partly by the unavailability of simple, robust code. We hope that the code and algorithms presented here, together with the examples, will encourage greater exploration of the properties and applications of this approach to spatial analysis and data structures.

## Acknowledgements

This research was made possible by an operating grant from Natural Sciences and Engineering Research Council of Canada, and from the Ministry of Culture and Higher Education of Iran. The authors gratefully acknowledge the assistance of Mr. Hugo Ledoux in program development and testing.

## References

Amenta, N., Bern, M., Eppstein, D., 1998. The crust and the beta-skeleton: combinatorial curve reconstruction. Graphical Models and Image Processing 60, 125–135.

Aurenhammer, F., 1987. Power diagrams: properties algorithms and applications. SIAM Journal of Computing 16, 78–96.

Aurenhammer, F., 1991. Voronoi diagrams—a survey of a fundamental geometric data structures. ACM Computing Surveys 23, 345–405.

Blum, H., 1967. A transformation for extracting new descriptors of shape. In: Whaten Dunn, W. (Ed.), Models for the Perception of Speech and Visual Form. MIT Press, MA, Cambridge, pp. 153–171.

Delaunay, B., 1934. Sur la sphère vide. Bulletin of the Academy of Sciences of the U.S.S.R. Classe des Sciences Mathématiques et Naturelle, Series 7 (6), 793–800.

Devillers, O., 1999. On deletion in Delaunay triangulations. 15th Annual ACM Symposium on Computational Geometry, pp. 181–188.

Fortune, S.J., 1992. Numerical stability of algorithms for 2D Delaunay triangulations. Proceedings of the Eighth Annual ACM Symposium on Computational Geometry, pp. 83–92.

Fritts, M.J., Crowley, W.P., Trease, H.E., 1985. The Free–Lagrange Method: Lecture Notes in Physics, Vol. 238, Springer, Berlin, 313pp.

Gold, C.M., 1989. Surface interpolation, spatial adjacency and GIS. In: Raper, J. (Ed.), Three Dimensional Applications in Geographic Information Systems. Taylor and Francis, London, pp. 21–35.

Gold, C.M., 1999. Crust and anti-crust: a one-step boundary and skeleton extraction algorithm. Proceedings, ACM Conference on Computational Geometry, Miami, pp. 189–196.

Gold, C.M., 2000. An algorithmic approach to a marine GIS. In: Wright, D., Bartlett, D. (Eds.), Marine and Coastal Geographical Information Systems. Taylor & Francis, London and Philadelphia, pp. 37–52.

Gold, C.M., Condal, A.R., 1995. A spatial data structure integrating gis and simulation in a marine environment. Marine Geodesy 18, 213–228.

Gold, C.M, Charters, T.D., Ramsden, J., 1977. Automated contour mapping using triangular element data structures and an interpolant over each triangular domain. In: George, J. (Ed.). Proceedings: Siggraph '77. Computer Graphics 11(2), 170–175.

Gold, C.M., Nantel, J., Yang, W., 1996. Outside–in: an alternative approach to forest map digitizing. International Journal of Geographical Information Systems 10, 291–310.

Gold, C.M., Snoeyink, J., 2001. A one–step crust and skeleton extraction algorithm. Algorithmica 30, 144–163.

Green, P., Sibson, R., 1977. Computing Dirichlet tessellations in the plane. Computing Journal 21, 168–173.

Guibas, L., Stolfi, J., 1985. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. Transactions on Graphiques 4, 74–123.

Guibas, L., Mitchell, J.S.B., Roos, T., 1991. Voronoi diagrams of moving points in the plane. In: Proceedings, 17th International Workshop on Graph Theoretic Concepts in Computer Science: Lecture Notes in Computer Science, Vol. 570, Springer, Berlin, pp. 113–125.

Heller, M., 1990. Triangulation algorithms for adaptive terrain modelling. In: Proceedings, Fourth International Symposium on Spatial Data Handling, pp. 163–174.

Lardin, P., 1999. Application de la structure des données Voronoi à la simulation de l'écoulement des eaux souterraines par différences finies intégrées. Mémoire de maîtrise en sciences géomatiques, Faculté de foresterie et de géomatique, Université Laval, Québec, 159p.

Lawson, C.L., 1977. Software for C 1 surface interpolation. In: Rice, J.R. (Ed.), Mathematical Software III. Academic Press, New York, NY, pp. 161–194.

Mostafavi, M.A., Gold, C.M., 2002. A global kinetic spatial data structure for marine simulation. International Journal of Geographical Information Science (IJGIS), accepted.

Okabe, A., Boots, B., Sugihara, K., 2000. Spatial Tessellations: Concepts and Applications of Voronoi diagrams, 2nd Edition. Wiley, Chichester. 671p.

Sambridge, M., Braun, J., McQueen, H., 1995. Geophysical parameterization and interpolation of irregular data using natural neighbours. Geophysical Journal International 122, 837–857.

Shewchuk, J.R., 1996. Triangle: engineering a 2D quality mesh generator and Delaunay triangulator. First Workshop on Applied Computational Geometry, Association for Computing Machinery, Philadelphia, Pennsylvania, pp. 124–133.

Sibson, R., 1981. A brief description of natural neighbour interpolation. In: Barnett, V. (Ed.), Interpreting Multivariate Data. Wiley, New York, pp. 21–36.

Thibault, D., Gold, C.M., 2000. Terrain reconstruction from contours by skeleton retraction. GeoInformatica 4, 349–373.

Voronoi, G., 1908. Nouvelles applications des paramètres continus à la théorie des formes quadratiques, deuxième memoire, recherche sur les parallelloèdres primitifs. Journal für die Reine und Angewandte Mathematik 134, 198–287.