The Visual Computer

Triangulated spatial models and neighbourhood search: an experimental comparison with quadtrees

Christopher B. Jones,
J. Mark Ware,
Christopher D. Eynon

School of Computing, University of Glamorgan,
Pontypridd, Wales CF37 1DL,
United Kingdom
e-mail: cbjones@glamorgan.ac.uk

We describe a Delaunay triangulation-based algorithm to search for the nearest line or polygonal boundary to an arbitrary point. We use geographical data to compare the algorithm experimentally with a linear quadtree search procedure. Both algorithms use priority queues. The rich proximity relations of the triangulation result in run-time performance that is clearly competitive with the quadtree, being quite similar with respect to the numbers of distance-based calculations and faster with regard to execution time. It is envisaged that the main benefits of triangulation for neighbourhood search are in applications in which the data are permanently triangulated or in which a triangulation assists in an application-specific analysis or transformation.

Key words: Nearest neighbour search – Delaunay triangulation – GIS – Priority queue – Quadtree

Correspondence to: C.B. Jones

1 Introduction

One of the major types of query required of spatial databases is that of proximity search. The intention is to find objects nearby or within a particular distance of a given location. In geographical information systems (GIS), proximity search is used, for example, in applications concerned with urban and regional planning where the distances from certain phenomena impose constraints on planned developments. Thus, the user may need to know of all resources of a specified type within a particular distance of a proposed new facility, or may wish to know about phenomena within a particular distance that may affect, or be affected by, a proposed development. In robotics, the movement of a machine part or of a vehicle may require continuous updating of information on nearby objects that may act as potential obstructions.

In most geographical databases, spatial queries, such as point location within a region, range searches and proximity searches, are implemented by means of a combination of a spatial index, such as a quadtree or an R-tree (or one of its variants), with geometric computation that may require a local, exhaustive search on a subset of the database obtained via the spatial index. Retrieval of candidate objects for a spatial search has been described as a filter step, while the execution of computational geometry procedures to find the correct answer has been described as a refinement step (Orenstein 1989). Kriegel et al. (1991) show that considerable benefits in spatial query can be obtained at the refinement step by employing structural decomposition of the spatial objects. This reflects the results of research in computational geometry in which the use of decomposition into trapezia or Voronoi regions, for example, has been shown to produce very efficient locational search procedures (Preparata and Shamos 1988).

When decomposition is applied in its basic form, despite the performance improvements of decomposition into the smallest spatial units such as triangles or trapezia, it imposes a considerable storage overhead. For this reason, more complex compromises between basic decomposition and non-decomposition have been proposed (Schiwietz and Kriegel 1993). However, in the particular case of triangulations, it is notable that some applications and procedures in GIS already build triangulated spatial models to exploit their benefits in modelling environmental and cartographic phenomena. The prime example in a GIS context is in terrain modelling, in which triangulated irregular networks (TINs) are widely

The Visual Computer (1999) 15:235–248

used (in preference to or in combination with regular grids) for the advantages that they bring with regard to interpolation between precise representations of original sample data (Peucker et al. 1978). This may include point, line and polygonal features that can act as constraints on the triangulation.

Interest in the use of triangulated models has increased recently in recognition of their merits in implementing the spatial transformations of map generalisation, whereby the cartographic forms of spatial phenomena are simplified to meet the requirements of information abstraction and the limitations of reduced map scales (DeLucia and Black 1987; Jones et al. 1995). Neighbourhood search is an essential aspect of automating map generalisation operators. Outside GIS, triangulations also have a long history of use in finite element analysis, computer-aided design and robotics. See Baehmann et al. (1987) and Storer and Rief (1994), for examples.

We now summarise several characteristics of triangulated spatial models that motivate further investigation of their application in processing spatial database queries.

Complete representation of map space. Conventional object-based spatial data representations focus on individual map features that may or may not be connected with neighbouring objects. In a triangulated model the entire region, including intervening areas between map objects, is explicitly represented.

Precise tessellated representation. The original precision of the source data can be maintained.

Maintenance of topological relations between spatial components. Explicit storage of connectivity between spatial data elements of a triangulation facilitates both the maintenance of consistency under geometric transformation and the efficient computation of answers to topological queries.

Rich proximity relations. Nearest-neighbouring elements are usually connected by triangulation, with the result that proximal search procedures can be implemented relatively simply and with a potential for great efficiency.

Facilitates surface interpolation. By constructing the most equiangular triangulation, which may be subject to linear constraints, high-quality interpolation is ensured when the triangulation represents a terrain model or other spatial field.

In this paper, a triangulation-based procedure for finding the object boundary nearest to an arbitrary point is described. It modifies a similar procedure described by Jones and Ware (1997), by introducing a priority queue. The performance of the procedure is evaluated with five geographical datasets and a comparison with an implementation of the same query using a PMR quadtree. This has previously been shown to provide good performance in proximity searches (Hoel and Samet 1991, 1992; Hjaltason and Samet 1995).

In the remainder of the paper, Sect. 2 reviews the characteristics of constrained Delaunay triangulations and makes a comparison with representations based on generalised Voronoi diagrams and their duals. This is followed in Sect. 3 by a description of a triangulated data structure called the simplicial data structure (SDS), which is based on constrained Delaunay triangulation. Section 4 documents a nearest-neighbour algorithm implemented with the SDS. Section 5 summarises how the same query is implemented in a quadtree, and gives experimental results. A comparison of the results of the two implementations is presented in Sect. 6, while Sect. 7 provides conclusions.

2 Delaunay triangulation

A Delaunay triangulation of a set of points is characterised by the fact that the circumcircle of each triangle contains no other points of the set. It is also the most equiangular triangulation in the sense of maximising the minimum angle (Sibson 1978). It is of particular interest in computational geometry in that it is the dual of the Voronoi diagram and, as such, it provides a solution to the problem of determining the nearest neighbour of each member of a set of points (Shamos and Hoey 1975). Voronoi regions of a set of points represent the locations that are closer to each point of the set than to any other point in the set. The Delaunay triangulation connects, through triangulation edges, each point with each other point with which it shares a boundary of their respective Voronoi regions.

Delaunay triangulations of sets of points are commonly used to represent digital elevation models in GISs and to model engineering structures for finite element analysis. In these applications it is common practice to modify the triangulation by introducing linear constraints to which the triangulation edges

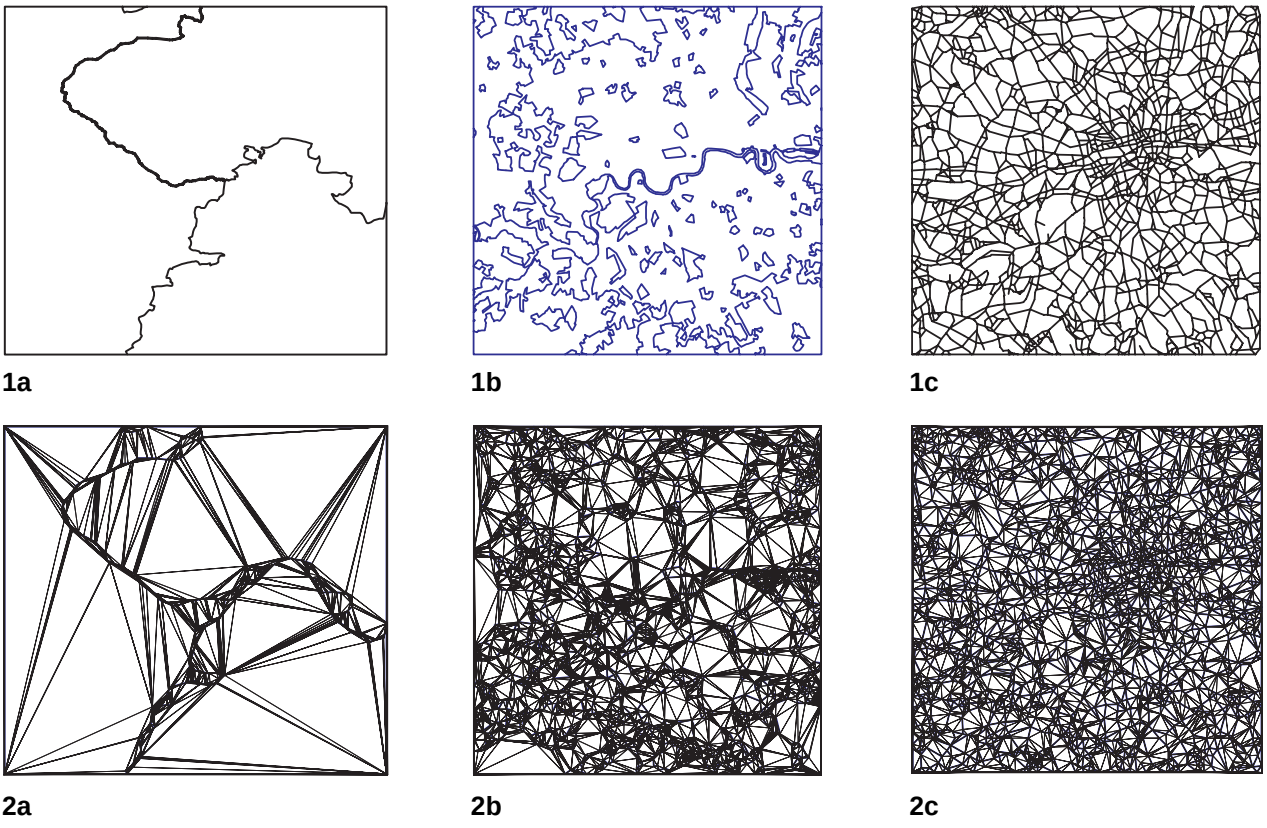


Fig. 1a–c. Test datasets **a** *ab*, **b** *ub* and **c** *rd3*

Fig. 2a–c. SDS representation of datasets **a** *ab*, **b** *ub* and **c** *rd3*

must conform. In terrain modelling these constraints may correspond to natural features such as ridges, valleys and breaks of slope, and to structures such as roads and buildings located on the terrain. The introduction of constraints to create a constrained Delaunay triangulation (CDT) results in the loss of the empty circumcircle criterion, but the triangulation is still the most equiangular triangulation, given the constraints (Lee and Lin 1986; Chew 1989). A further consequence of introducing constraints is that there is no longer a simple duality relationship with the Voronoi diagram. For a set of points and straight edges, the corresponding Voronoi diagram is sometimes referred to as a generalised Voronoi diagram, or line Voronoi diagram, and its dual consists of a triangulated graph in which nearest-neighbouring elements are directly connected (Lee and Drysdale 1981). Each node of the graph corresponds to either a point or a line element in the original data, but, unlike a CDT, it is not an explicit representation of the Cartesian coordinate-based location of the graphic elements.

In a CDT, the nearest-neighbouring point and line elements are in fact usually connected by triangulation edges. Experiments conducted by the authors using the datasets described later in this paper found nearest-neighbour connectivity in more than 99.5% of all cases tested. Figure 2b illustrates a CDT of the set of polygonal objects in Fig. 1b.

3 Proximity relations in a simplicial data structure (SDS)

A SDS is defined to consist of a constrained Delaunay triangulation T of a set of points and straight edge sites S representing the geometry of a set of objects O. Objects may be one of three types: point,

line and area. A point object $o_p \in O$ consists of a single point site $s_p \in S$. A line object $o_l \in O$ consists of one or more edges $s_e \in S$, while an area object $o_a \in O$ is a simple polygon represented by at least three edges $s_e \in S$.

It is possible to describe several proximity relations in terms of the triangulation T of a set of point and straight edge sites S. Two objects are contiguous if they share one or more elements $s \in S$. Contiguity may be either point contiguity if the objects share a vertex, or edge contiguity if the objects share an edge. Both cases are referred to here as proximal-0. Non-contiguous proximality is described as proximal-n, where $n > 0$ is the minimum number of triangulation edges that must be traversed (from vertex to vertex) to move from one object to the other.

In a Delaunay triangulation of a set of points, Voronoi neighbours are proximal-1 related. In a CDT, Voronoi neighbours, i.e. those point and line elements that would share a boundary of their respective Voronoi regions, are not necessarily proximal-1, though as already indicated, direct connectivity and hence the proximal-1 relationship is found to hold in a very large proportion of cases.

3.1 Nearest-neighbour search in a SDS

To find the nearest-neighbouring object of an object in the SDS, it is possible to implement a search procedure that enumerates the component edges of triangles in the vicinity of the source object in distance order, examining components of the SDS in increasing order of their proximal-n relation to the source object. The search terminates when it is proved that there is no nearer object than a current nearest-neighbour candidate. Clearly, if it had been known that the nearest non-contiguous neighbour was proximal-1, the search would have had a locally constrained performance that depended upon the valency of the triangulation nodes on the boundary of the source object. Without guaranteed direct connectivity, i.e. proximal-1 relations, the search will often need to proceed beyond the directly connected neighbouring elements.

In this paper we focus on searching for the boundary of the nearest neighbour of an arbitrary point p that is not a member of the original sites S. In this case the containing triangle of the point is found, and a search is conducted around that triangle starting with the proximal-0, i.e. contiguous, neighbours of the triangle.

In a boundary-nearest-a-point query, the extent to which the search needs to proceed beyond the proximal-0 elements depends upon the degree of equiangularity of the triangulation. This is illustrated in Fig. 3. Having found a candidate nearest neighbour in the proximal-0 elements of the seed triangle in an equiangular triangulation, the actual nearest neighbour has a great likelihood of also being proximal-0. In a nonequiangular triangulation, the proximal-0 initial candidate nearest neighbour may be relatively far from the seed location, and a search could require examining triangles at several degrees of proximity beyond the connected set to prove that the nearest neighbour has been found. It may be noted, however, that the search need not proceed beyond any given triangle edge if that edge is at a distance equal to or greater than the distance to a current nearest-neighbour candidate in the search procedure.

3.2 Components of the SDS

Here we describe an implementation of the SDS in which all objects are either lines or areas. Inclusion of point objects is a simple extension, and it is not described here. The data structure is based on a constrained Delaunay triangulation T of vertices and constraining edges constituting the sites S, as previously stated. The triangulation is defined in terms of a set of edges E and vertices V. An object is classified as either a line object o_l, which is associated with pointers to the set of component edges $E_{o_l} \in E$, or an areal object o_a, which is associated with pointers to the set of component triangles $T_{o_a} \in T$. Each triangle $t \in T$ is associated with a triangle identifier and a set of three pointers to the component edges $\{e_1, e_2, e_3\}$. If the triangle belongs to an areal object, the identity of that object is stored (otherwise this value is null). Each edge is classified as either a real edge if it coincides with an edge $s_e \in S$, or a virtual edge. If an edge is real and it belongs to a linear object, the identity of that object is stored; otherwise this value is null. An edge is associated with pointers to its two component vertices $\{v_s, v_e\}$ and to each of its adjacent triangles on the left and right sides $\{t_l, t_r\}$.

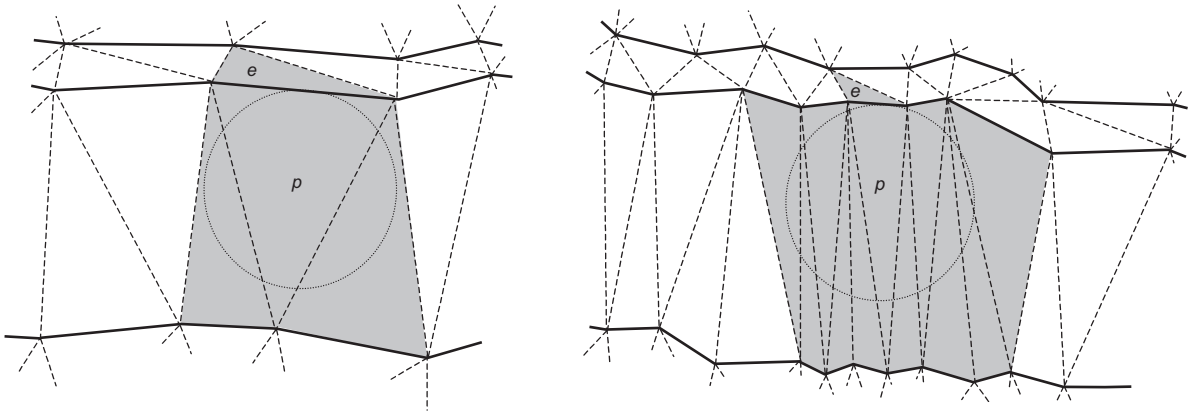


Fig. 3a, b. Finding the nearest object boundary to a point *p*. In both cases, a candidate element *e* is found. Case **a** requires fewer triangles to be processed than **b** due to greater triangle equiangularity in **a**

4 The SDS algorithm for the neighbour nearest to a point

The problem of finding the object boundary nearest to an arbitrarily placed point can be reduced to that of finding the real edge nearest to that point. If this edge belongs to a linear object, then the identity of the object is found by examining the edge's object pointer. Alternatively, if the edge belongs to an areal object(s), then the object(s) is found by examining the object identifiers associated with its adjacent triangles.

The algorithm for finding the real edge e_n nearest to an arbitrarily placed point p is listed in pseudocode in Fig. 4. The algorithm assumes that the triangle t_c containing p has been found previously by means of a spatial index (in this implementation the index is in the form of a regular grid). To find e_n, the search proceeds outward from t_c, enumerating the SDS edges in increasing order of distance from p. A record of the real edge currently found to be nearest is kept throughout the search, together with its associated minimum separating distance d_n from p. During the search, edges and triangles are marked as having been visited when they are processed.

Initially, each edge e of t_c is examined in turn (step 1). If the minimum separating distance d between p and e (calculated with the function *PointEdgeDistance*) is less than the current value of d_n (initially set to an appropriate *large_value*), then a test is carried out (with the Boolean function *IsReal*) to check whether e is a real edge. If it is, then e_n is set to the value of e and d_n is updated to d. Alternatively, if e is a virtual edge (and lies within a distance d_n of p), then it is added to a queue *search_queue* of edges waiting further processing. The *search_queue* is initially set to empty by *InitialiseQueue*, and edges are added with the function *AddToQueue*. Each *search_queue* item records, in addition to e, the identity of the triangle used in locating e and the minimum separating distance d. *AddToQueue* ensures that edges are maintained in *search_queue* in order of increasing distance from p.

The next part of the search algorithm (step 2) consists of a loop in which items (each consisting of an edge e, a triangle t and a distance d) are removed from the head of *search_queue* with the function *TakeFromQueue*. The loop is repeated until either *search_queue* is emptied or no edge in *search_queue* lies nearer to p than the current nearest edge e_n. After an edge e is removed from the queue, processing the edge begins with a test to see if its distance d to p is less than the current d_n. If it is not, then this indicates that the true nearest real edge has already been found and the search can stop. Alternatively, if d is less than d_n, then further processing of e is required. This involves, firstly, finding the triangle t_1 adjacent to t and sharing with it the common edge e with the function *TriangleEdgeConnect*.

```
BeginPriorityQueueSDSSearch
  InitialiseQueue(search_queue)
  d_n ← large_value
  STEP 1:
  For each edge e ∈ t_c
    d ← PointEdgeDistance(p, e)
    If (d < d_n)
      If (IsReal(e))
        e_n ← e
        d_n ← d
      Else
        AddToQueue(search_queue, e, t_c, d)
      Endif
    Endif
    Mark e as visited
  Endfor
  Mark t_c as visited
  continue ← TRUE
  STEP 2:
  Do while (NotEmpty(search_queue)) and (continue)
    TakeFromQueue(search_queue, e, t, d)
    If (d < d_n)
      t_1 ← TriangleEdgeConnect(t, e)
      If (NotAlreadyVisited(t_1))
        For each edge e_1 ∈ t_1
          If (NotAlreadyVisited(e_1))
            d_1 ← PointEdgeDistance(p, e_1)
            If (d_1 < d_n)
              If (IsReal(e_1))
                e_n ← e_1
                d_n ← d_1
              Else
                AddToQueue(search_queue, e_1, t_1, d_1)
              Endif
            Endif
          Endif
          Mark e_1 as visited
        Endfor
        Mark t_1 as visited
      Endif
    Else
      continue ← FALSE
    Endif
  Enddo
EndPriorityQueueSDSSearch
```

Fig. 4. Pseudo-code for Algorithm 1, a SDS search procedure using a priority queue

Then, provided t_1 has not been processed previously, each edge of t_1 is examined in turn. For a particular edge e_1, this involves an initial test to check if the edge has been examined previously. If it has not, then the distance d_1 between p and e_1 is calculated. If d_1 is less than d_n and e_1 is a real edge, then e_n and d_n are set to the values of e_1 and d_1, respectively. If, however, d_1 is less than d_n and e_1 is virtual, then e_1 (together with t_1 and d_1) is added to *search_queue*.

4.1 Correctness of Algorithm 1: FindNearestRealEdgeToPoint

Proof of correctness of the algorithm is based on the assertion that the algorithm will build a polygonal region that contains the query point and the nearest-neighbouring edge, and that, when the algorithm terminates, all parts of the boundary of the region will be at least as far away from p as the nearest edge is. It is assumed that the triangulation is fully edge-connected in that it is possible to move within the triangulation from any point to any other point by crossing triangle edges and passing within the interior of triangles.

Lemma 1. *Given a query point p located inside a triangle t_s belonging to the triangulation T, the edge-based search in Algorithm 1 will build and maintain a polygonal region that contains p and consists of the union of the processed triangles T_p.*

Proof. Once processed, the triangle t_s constitutes the initial state of the polygon, and by definition t_s contains p. The next triangle t_n to be processed at all stages in the search process is the one externally connected to the edge in the queue that is nearest to the query point p. Triangle t_n must therefore be edge-connected to the processed triangle set, since edges are placed in the queue only as a consequence of processing a triangle which becomes a member of the set T_p.

Lemma 2. *Given a query point p located inside a constrained triangulation T, Algorithm 1 will find the real edge $e_n \in T$ such that the distance from p to e_n is smaller than or equal to the distance from p to all other real edges $e_k \in T$.*

Proof. Since T_p defines a polygonal region containing p, it will be possible to prove that the nearest edge has been found if there is a real edge $e_n \in T_p$ and there is no other edge (real or virtual) $e_k \in T_p$ that is nearer to p than e_n is. The first step of the algorithm consists in processing the triangle t_s that contains p. Thus, from Lemma 1, the algorithm cannot terminate without having built a polygon containing p. When t_s has been processed, if one or more of its edges is real, then there will be a candidate nearest edge e_n. If this edge is nearer to p than the other edges of t_s, then no triangles will have been placed in the queue and the algorithm will terminate, having identified the real edge on the boundary of T_p that is nearer than all other parts of the boundary. Alternatively, if T_p

grows beyond t_s, the process of growing T_p will only terminate when there is no edge in the queue that is nearer to p than e_n is. If there is no such edge in the queue, then all boundary edges of T_p must be farther from p than e_n; otherwise, they would have been queued.

4.2 Analysis

The algorithm requires a preprocessing stage to build the constrained Delaunay triangulation T. This stage requires $O(N \log N)$ time and $O(N)$ storage (Chew 1989), where N is the number of vertices. Step 1 of the algorithm calculates three distances and adds a maximum of three items to the queue; it therefore requires $O(1)$ time. Step 2 consists of an iterative process in which the function, *TakeFromQueue* and *PointEdgeDistance* each require a constant time, while the function *AddToQueue* takes $\log l$ time where l is the length of the queue. The number of iterations is equal to the number of edges that are processed.

It is easy to establish that the lower bound on the algorithm is $O(1)$, as this follows from the fact that the algorithm will terminate after step 1 if no edges are placed in the queue. If items are placed in the queue, then processing time is $O(k \log k)$ where k is the number of edges that are processed.

It is possible to envisage worst-case scenarios for the algorithm in which, due to a particular configuration of the data, all edges of T are examined before the algorithm terminates, giving a time of $O(N \log N)$. This could arise if the data points were organised in the form of two parallel rows on opposite sides of a square and the query point was located at the centre of the square. In this situation, the nearest edge would be proximal-0 to the seed triangle, but it would be necessary to search in both directions parallel to the rows to create a processed triangle set in which all boundary edges were farther than or at equal distance to the nearest edge(s).

The worst-case analysis represents what would be an extremely unlikely configuration of the data for geographical applications. The number of edges processed in a query is itself determined by the number of triangles required to construct the surrounding polygon of the query point. This polygon always covers the disc of radius equal to the distance to the nearest neighbour. Hence, the number of such triangles will be related to the degree of equiangularity (as illustrated in Fig. 2).

4.3 Experimental results for the object nearest to a point when the SDS was used

In order to examine the performance of the algorithm in practice, five datasets were used in the tests. This section reports on results from the SDS algorithm, while the next section reports on the results when a quadtree was used for the same set of searches. Three of the datasets are illustrated in Fig. 1. Dataset *ab* consists of administrative boundaries, such as counties. Dataset *ub* consists of urban area boundaries. Datasets *rd*1 and *rd*2 consists of two different sets of roads, while dataset *rd*3 consists of a generalised version of *rd*2, i.e. one in which the numbers of vertices has been reduced. The characteristics of the datasets are summarised in Table 1.

Table 1. Description of datasets

Data set	Number of objects	Number of edges	Number of vertices	Number of SDS triangles	SDS storage		Average real edge length (in metres)	Average virtual edge length (in metres)	Average triangle aspect ratio	Quadtree storage (in bytes)
					Triangulation	Grid				
ab	10	858	859	1 712	41 120	39 575	261.9	2218.4	30.04	48 602
ub	313	5012	4968	9 930	239 024	276 728	264.3	888.7	8.80	188 496
rd1	1423	6154	5648	11 289	279 088	321 036	305.8	945.7	7.05	319 160
rd2	4209	9252	8003	16 000	404 032	519 468	300.1	716.0	5.64	484 018
rd3	4209	4122	2963	5 920	162 112	195 540	672.3	1108.5	4.68	226 138

A triangles aspect ratio is calculated by dividing its longest edge length by its shortest height
SDS, Simplicial data structure

The tests consisted of generating 10 000 query points on a regular grid and finding the object nearest each point for each of the datasets. The results are tabulated in Tables 2 and 3. This was done in order to cast some light on the applicability of the theoretical evaluation of performance already discussed. In Table 2, the number of calculations refers to the number of point–edge distance calculations that were performed relative to edges of the triangulation, assuming that the seed triangle has already been found. Note that the point–edge distance calculations are required for examining both real and virtual edges of the triangulation. In Table 3, the number of calculations refers to a value that adds the number of point–edge distance calculations that were performed for the triangulation edges to double the number of point-in-triangle tests needed to find the seed triangle from the spatial index. The factor of two is based on a comparison of the number of multiplications and divisions used in the point-edge distance function and the numbers of such operations used in the point-in-triangle function.

Note that the average value of numbers of calculations in Table 2, based only on point–edge distance calculations is less than 24 in all cases, but the maximum value rises to 95 for two of the datasets.
The CPU times per query, on a SPARC 10 computer, ranged from 0.00009 s for dataset *rd*3 to 0.00025 s for dataset *ab*.

5 PMR Quadtree search for the neighbour nearest to a point

In an effort to evaluate the triangulation proximity search algorithm with respect to other search methods, the same set of tests was run with an implementation of the PMR quadtree (Nelson and Samet 1986). This was chosen as an alternative data structure for comparative purposes as it has been documented in some detail and has itself been compared systematically with other spatial data structures (Hoel and Samet 1992). In particular, Hoel and Samet (1991) report detailed results for finding the edge nearest to a point in a ‘bottom-up’

Table 2. Results of the object-nearest-to-point test with simplicial data structure

Data set	Number of calculations				Average number of real edges examined	Time taken (in seconds)	
	Average	Standard deviation	Minimum	Maximum		Average	Standard deviation
ab	23.66	13.39	3	84	6.66	0.000244	0.000311
ub	14.64	7.44	3	58	4.03	0.000133	0.000154
rd1	13.53	7.46	3	56	4.18	0.000123	0.000160
rd2	10.47	6.18	3	95	3.53	0.000098	0.000164
rd3	7.27	4.84	3	95	2.92	0.000070	0.000091

The initial containing triangle is assumed to be known. Each test involves 10 000 point queries

Table 3. Results of the object-nearest-to-point tests with simplicial data structure

Data set	Number of calculations				Time taken (in seconds)	
	Average	Standard deviation	Minimum	Maximum	Average	Standard deviation
ab	28.53	15.66	3	148	0.000250	0.00031
ub	19.40	7.94	3	75	0.000143	0.00016
rd1	18.81	7.83	3	76	0.000134	0.00015
rd2	15.34	7.95	3	193	0.000109	0.00017
rd3	12.06	7.25	3	197	0.000085	0.00015

This includes calculations and time required to locate the initial containing triangle. Each test involves 10 000 point queries. Note that the average number of real edges examined is the same as for the corresponding experiment in Table 2

procedure. The containing cell of the query point is found, then adjacent cells within the radius of the currently nearest identified edge are examined to find the edge that is nearest. An analysis of the PMR quadtree for finding the edge nearest to a point shows that the procedure gave an *average* time of $O(k)$, where k is the splitting threshold of the quadtree. Experimental results were presented with geographical data. The PMR quadtree has also been used as the basis of a description of a top-down distance-ranking procedure that uses a priority queue to find all neighbours nearest, or k-nearest, to a given object (Hjaltason and Samet 1995). An analysis of the latter procedure concludes that the worst-case time complexity was $O(N \log N)$, where N is the number of leaf blocks in the quadtree and the $\log N$ term refers to the cost of updating the priority queue. This analysis also applied to the case of finding the neighbour nearest to an object, but was described as corresponding to a pathological worst case.

In making a comparison with the SDS-based search procedure with regard to distance-based calculations in the quadtree search procedure, it is necessary to take account of the cost of determining whether a quadtree cell is within the current search radius. This is required as part of the pruning process whereby the contents of a cell are only examined if the cell itself is within the radius. Distance measurements to each of the horizontal and vertical faces of the cells require about one-quarter of the number of floating point operations required by the function to determine the distance from a point to an arbitrary edge. Thus, the four distance tests for all faces of a cell can be weighted as approximately equivalent to the cost of a single point–edge distance calculation.

5.1 Quadtree search procedure

A main memory linear PMR quadtree, in which only leaf nodes were stored, was implemented. The quadtree cells were identified by Morton numbers and the associated level in the tree. The search procedure uses a priority queue and adopts the bottom-up approach described by Arya et al. (1994) in their account of a nearest-neighbour procedure, which, with their specialised (not quadtree) spatial indexing method, gave a search time complexity of $O(\log N)$. Note that the quadtree was implemented in main memory to assist in timing comparisons with the SDS search technique, which is also main-memory based.

The procedure for a quadtree search with a priority queue builds a list of quadtree cells in the vicinity of the cell containing the query point following the method of Arya et al. (1994), which is adapted to the particular case of the quadtree decomposition of space. The approach is based on enumerating cells in order of their distance from the query point. The priority queue is built dynamically; only those neighbouring cells of a current cell that are candidates for containing the nearest object are added.

The procedure, which is summarised in pseudo-code in Fig. 5, starts by finding the cell containing the query point p and recording the distances to and the coordinates of the nearest locations to p on each face of the cell. The edges referenced by the cell are examined, and, if there are edges in the cell, the edge nearest to p is saved. For those faces of the cell closer than the current nearest edge, the neighbouring cell across the face is found and placed in the priority queue. Cells are inserted into the priority queue by determining the distance to the query point for each face. The smallest of these distances (and hence the shortest distance from p to the cell) is used to determine the sorting order in the priority queue. Provided the priority queue is not empty, the cell at the head of the queue is removed from it. If the distance to this cell is less than the distance to the current nearest edge, its referenced edges are examined. If its neighbouring cells are within the current search distance, they are added to the priority queue, as already indicated, provided that they have not previously been inserted into the queue. Note that one of the four neighbours of each cell will be the cell from which it was entered. The process terminates either when the cell at the head of the priority queue is farther away than the currently nearest edge, or when the priority queue is empty.

5.2 Observations on the analysis of the quadtree search procedure

The quadtree search procedure is used here for experimental comparison only and is based on principles described elsewhere (Nelson and Samet 1986; Hoel and Samet 1991, 1992; Arya et al. 1994; Hjaltason and Samet 1995). We provide some observations on the worst-case complexity of Algorithm 2, which gives the same result as that of Hjaltason and Samet, and we comment briefly on the lower-bound time complexity of the algorithm.

```
BeginPriorityQueueQuadtreeSearch
  search_radius ← maximum range of data
  Find cell c containing query point q
  For each face f of cell c
    fd_cf ← distance(q, f)
    np_cf ← nearest point to q on f
  Endfor
  distance_to_cell ← 0
  continue ← TRUE
  While ((distance _to_cell < search_radius) and continue)
    For each edge e referenced by cell c
      distance_to_edge ← distance(q, e)
      If (distance_to_edge < search radius)
        search_radius ← distance_to_edge
        nearest_edge ← e
      Endif
    Endfor
    For each face f of cell c
      If (fd_cf < search radius)
        Find neighbouring cell n at nearest point (np_cf) to q on f
        If (n is inside the quadtree and n has not been visited)
          For each face f of cell n
            fd_nf ← distance(q, f)
            np_nf ← nearest point to q on f
          Endfor
          distance_to_cell ← min{fd_nf}
          Add_to_priority_queue(n, {fd_n1, fd_n2, fd_n3, fd_n4}, {np_n1, np_n2, np_n3, np_n4},
                                distance_to_cell)
          Mark n as visited
        Endif
      Endif
    Endfor
    If (notempty(priority queue))
      c ← head of priority queue
    Else
      continue ← FALSE
    Endif
  Endwhile
EndPriorityQueueQuadtreeSearch
```

Fig. 5. Pseudo-code for Algorithm 2, a quadtree search procedure using a priority queue

Processing a cell to place it in the priority queue requires a constant number of distance calculations and a search to find the identity of the neighbouring cell for each face. In our implementation, this neighbouring-cell location procedure involves a search through the Morton-ordered list of leaf nodes to find an equivalent or immediately-less-than match between a leaf node and the Morton code of the location generated by neighbourhood operations. This therefore requires $O(\log N)$ time, where N is the number of leaf nodes.Assuming that there are m cells placed on the priority queue before the nearest edge is found, this leads to $m \log N$ time. Insertion into the priority queue is governed by a sorting order and therefore takes time $O(\log m)$, where m is the length of the priority queue.

A worst-case scenario for Algorithm 2 in which all data edges are located tangential to a circle and in which the query point is located at the centre of the circle can be envisaged. Since the algorithm will only terminate when either there are no items in the priority queue or there is no item in the queue that is nearer than the nearest edge, it will be necessary to process all k quadtree cells that reference data. This is due to the fact that the boundaries of those cells will be either at the nearest-neighbour search radius or inside it. The only cells of the quadtree not to be processed would be those beyond (relative to the query point) the set of cells referencing data items. Thus, the algorithm will require $O(\log N)$ operations to find neighbouring cells plus $O(k \log N)$ operations to insert cells into the queue. Since k will approximate to (but not exceed) N, the total number of leaf nodes in the data configuration envisaged, the worst-case time is $O(N \log N)$.

A lower bound on the time complexity of the algorithm is given by the case in which the quadtree cell containing the query point also contains the nearest edge, and in which all faces of the cell are farther from the query point than the nearest edge. In this situation, no cell will be placed in the priority queue, and the algorithm will terminate after processing a single cell giving a time of $O(1)$. Note that the number of edges that can be referenced by a cell is limited to that of the splitting threshold value specified for the quadtree.

5.3 Quadtree search test results

Table 4 contains the results of the quadtree search tests for the five datasets, and Fig. 6 illustrates the quadtrees for three of the datasets. The quadtree splitting threshold was 8 in all cases. Note that the number of distance calculations is based on the number of point–edge distance calculations plus the number of point–cell distance calculations. Note that the numbers of such calculations for the datasets varied on average between 11.9 and 26.3. One column of the table reports separately the numbers of real edges of the data that were examined, i.e. for which point-edge distance calculations were performed. The average CPU times per query, on a SPARC 10, ranged from 0.00045 s for dataset *rd*2 to 0.00130 s for dataset *ab*.

6 Comparison of results for SDS and quadtree search procedures

In comparing results from the SDS and quadtree search procedures, the results presented focus on the distance computations, expressed in terms of comparable calculations. In summary, the average numbers of calculations ranged from 12.1 to 28.5 for the SDS including the search for the seed triangle, and from 11.9 to 26.3 for the quadtree, taking account of all datasets. In an effort to provide some comparison with the method of presenting results used by Hoel and Samet (1991, 1992) for a similar query, we have also reported the numbers of edges ('real edges') of the source data that were examined for purposes of point-edge distance calculations. In the SDS these range from 2.9 to 6.7, while in the quadtree they range from 9.6 to 16.5. The lower and upper values correspond to the same datasets, namely *rd*3 and *ab*, respectively, for both procedures. Hoel and Samet (1991) found that the numbers of such edges examined in their datasets ranged between 29.67 and 37.22. No experimental results were reported in association with the top-down quadtree-ranking procedure of Hjaltason and Samet (1995).

The average CPU times per query in the SDS including the search for the seed triangle range from 0.00009 to 0.00025, while in the quadtree they range from 0.00045 to 0.0013.

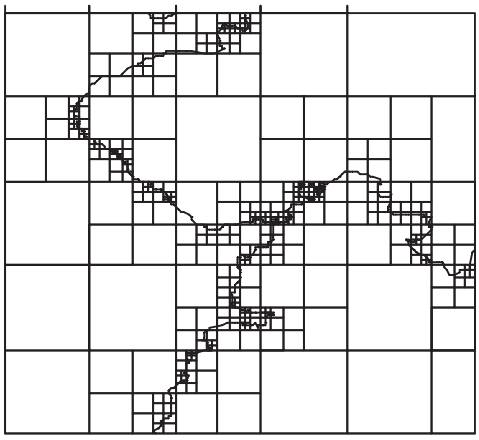
The storage of the SDS, when the spatial grid associated with the reported results is included, was about double the storage required for the quadtree with a threshold of 8.

If the threshold of the quadtree is set to 4, then the storage requirements for the two schemes are very similar. Doing so results in a very small improvement in the numbers of distance calculations for the quadtree, but leads to a notable increase in the amount of CPU time, as would be expected from the theoretical analysis, which indicates the sensitivity to the number of leaf nodes in the quadtree. The SDS

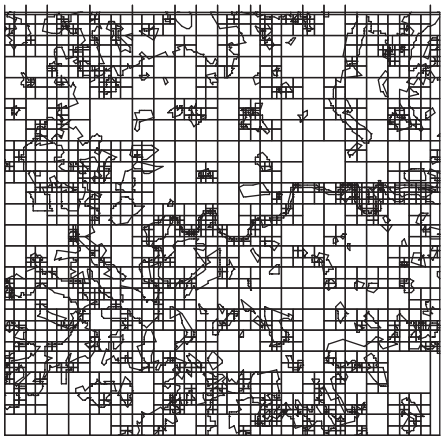
Table 4. Results of the object-nearest-to-point search using a quadtree

Data set	Number of calculations (edges + cells)				Average number of real edges examined	Time taken (in seconds)	
	Average	Standard deviation	Minimum	Maximum		Average	Standard deviation
ab	16.4 + 9.9 = 26.3	14.1 + 32.2	1 + 1	98 + 48	16.45	0.00130	0.0011
ub	12.6 + 4.4 = 17.0	7.8 + 13.9	1 + 1	59 + 24	12.56	0.00062	0.00047
rd1	12.8 + 3.9 = 16.7	8.2 + 13.3	1 + 1	67 + 29	12.78	0.00059	0.00059
rd2	11.1 + 2.9 = 14.0	7.0 + 8.6	1 + 1	54 + 16	11.13	0.00045	0.00032
rd3	9.6 + 2.3 = 11.9	5.9 + 6.4	1 + 1	45 + 12	9.64	0.00070	0.00061

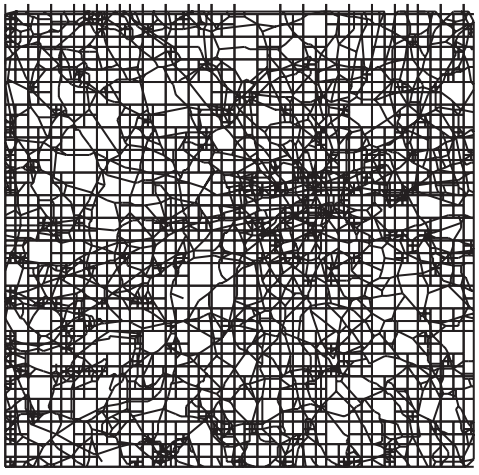
Each test involves 10 000 point queries



a



b



c

Fig. 6a–c. Quadtree representation of datasets **a** *ab*, **b** *ub* and **c** *rd3*

spatial grid cell size chosen in the results reported here was based on an average of one cell per four triangles. The use of smaller cell sizes leads to improved performance for the numbers of calculations, though there is no difference in the numbers of real edges examined, because they reflect a search process that takes place after finding the initial triangle.

7 Conclusions

This paper has shown how the rich proximity relations of constrained Delaunay triangulation (CDT) can be exploited for the purposes of answering nearest-neighbour queries. The particular example of finding the nearest linear or polygonal object to an arbitrary point has been used. A triangulation-based procedure has been presented and its performance evaluated experimentally with digital map datasets. The results have been compared with a procedure to perform the same operation implemented with a main memory linear PMR quadtree. The results obtained with the SDS compare favourably to those obtained with the quadtree. On the basis of the numbers of distance calculations, the SDS results are very similar to those of the quadtree. However, timings for the experimental data show differences: the SDS procedure is faster than the quadtree. This difference reflects the exploitation of the explicit topological relations of the triangulation in the SDS. Thus, a neighbouring triangle in the SDS is accessed simply via a single pointer, whereas access to a neighbouring cell in the quadtree requires a separate procedure. In a linear implementation of

the quadtree, this requires searching the sorted list of Morton codes, while in an explicit pointer-based tree implementation of a quadtree it requires a partial rewinding of the recursive traversal (Samet 1990). The top-down, pointer-based tree, distance-ranking procedure of Hjaltason and Samet (1995) avoids this search by recording all visited nodes in a priority queue, but the queue is necessarily longer than that built by the method of Arya et al. (1994). The benefit of the top-down scheme becomes most apparent when ranking of multiple or all objects is required, as opposed to finding the nearest object.

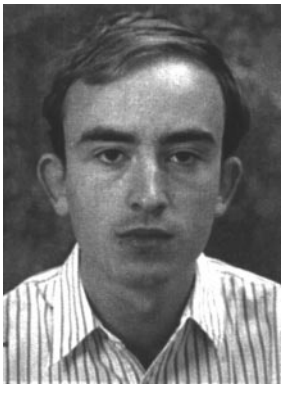
The research was motivated by the fact that CDTs are used in several applications of spatial databases, notably for terrain modelling, but also for tasks in cartographic map generalisation, robotics and computer-aided design. It is therefore assumed that the data structure may already be stored for such purposes, and it is of interest to develop efficient spatial search procedures that exploit it. Kriegel et al. (1991) also show that spatial decomposition is potentially a very efficient approach to query processing. The results presented here contribute further to that field of study. We remark that permanently stored triangulations impose more storage overhead than simpler spatial data representations. As Jones et al. (1994) point out, considerable saving on storage can be achieved by using an implicit triangulation, whereby the explicit triangulation is constructed at the time of query from a spatially localised and application-specific part of what may be a very large database.

References

1. Arya S, Mount DM, Netanyahu NS, Silverman R, Wu A (1994) An optimal algorithm for approximate nearest neighbour searching. Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms, Arlington, Virginia, ACM, pp 573–582
2. Baehmann PL, Wittchen SL, Shephard MS, Grice KR, Yerry MA (1987) Robust, geometrically based, automatic two-dimensional mesh generation. Int J Numerical Methods Eng 24:1043–1078
3. Chew PL (1989) Constrained Delaunay triangulations. Algorithmica 4:97–108
4. DeLucia A, Black T (1987) A comprehensive approach to automatic feature generalization. Proceedings of the 13th International Cartographic Conference, Morelia, Mexico, International Cartographic Association, Morelia, Mexico, pp 169–191
5. Hoel EG, Samet H (1991) Efficient processing of spatial queries in line segment databases. In: Gunther O, Schek H-J (eds) Advances in spatial databases: proceedings of the 2nd Symposium on the Design and Implementation of Large Spatial Databases (SSD '91) (Lecture Notes in Computer Science, vol 525) Springer, Zurich, pp 237–256
6. Hoel EG, Samet H (1992) A qualitative comparison study of data structures for large line segment databases. ACM SIGMOD Record 21:205–214
7. Hjaltason GR, Samet H (1995) Ranking in spatial databases. In: Egenhofer MJ, Herring JR (eds) Advances in spatial databases. (Lecture Notes in Computer Science, vol 951) Springer, Berlin Heidelberg New York, pp 237–256
8. Jones CB, Ware JM (1998) Proximity relations with triangulated spatial models. Comput J 41:71–83
9. Jones CB, Kidner DB, Ware JM (1994) The implicit triangulated irregular network and multiscale spatial databases. Comput J 37:43–57
10. Jones CB, Bundy GL, Ware JM (1995) Map generalisation with a triangulated data structure. Cartography Geogr Information Syst 22:317–331
11. Kriegel H-P, Horn H, Schiwietz M (1991) The performance of object decomposition techniques for spatial query processing. Advances in spatial databases: proceedings of the 2nd Symposium on the Design and Implementation of Large Spatial Databases (SSD '91) (Lecture Notes in Computer Science, vol 525) Springer, Berlin Heidelberg New York, pp 257–276
12. Lee DR, Drysdale RL (1981) Generalization of Voronoi diagrams in the plane. SIAM J Comput 10:73–87
13. Lee DT, Lin AK (1986) Generalized Delaunay triangulation for planar graphs. Discrete Computat Geom 1:201–217
14. Nelson RC, Samet H (1986) A consistent hierarchical representation for vector data. Computer Graphics (SIGGRAPH '86 Proceedings) Vol 20, No 4, 197–206
15. Orenstein JA (1989) Strategies for optimising the use of redundancy in spatial databases. 1st Symposium on the Design and Implementation of Large Spatial Databases, Santa Barbara, Calif., Lecture Notes in Computer Science Vol 409, Springer, Berlin Heidelberg New York, pp 115–134
16. Peucker TK, Fowler RJ, Little JJ, Mark DM (1978) The triangulated irregular network. ASP/ACSM Digital Terrain Models (DTM) Symposium, ACSM, Falls Church, Virginia, 516–540
17. Preparata FP, Shamos MI (1988) Computational geometry. Texts and monographs in computer science. Springer, New York
18. Samet H (1990) The design and analysis of spatial data structures. Addison-Wesley, Reading, Mass.
19. Schiwietz M, Kriegel H-P (1993) Query processing of spatial objects: complexity versus redundancy. In: Abel D, Ooi BC (eds) Advances in spatial databases. (Lecture Notes in Computer Science, vol 692) Springer, Berlin Heidelberg New York, pp 377–396
20. Shamos MI, Hoey D (1975) Closest-point problems. The 16th Annual IEEE Symposium on Foundations of Computer Science, University of California, Berkeley. IEEE pp 151–162
21. Sibson R (1978) Locally equiangular triangulations. Comput J 21:243–245
22. Storer JA, Rief JH (1994) Shortest paths in the plane with polygonal obstacles. J ACM 41:982–1012

CHRIS JONES is Professor of Geographical Information Systems at the University of Glamorgan. He worked previously at the University of Cambridge, BP Exploration and the British Geological Survey. He studied geology and graduated from Bristol University. He received a PhD from the University of Newcastle upon Tyne for research on periodicities in fossil growth rhythms. His current research interests include the use of geographical information in hypermedia for public information access, multiscale spatial databases, the computer reconstruction and visualisation of fossils, and automated cartographic design with regard to map generalisation and automated text placement.

MARK WARE is a Lecturer at the School of Computing at the University of Glamorgan. He studied Maths and Computing and graduated from the then Polytechnic of Wales in 1989. He received a PhD (1994) from his present university for research on multiscale data structures for spatial information systems. His current research interests include automated cartography (map generalisation), spatial database error, automated environmental change detection and digital terrain modelling.



CHRISTOPHER EYNON is a Research Assistant in Geographical Informations Systems at the University of Glamorgan. He has a BSc in Physics with Astronomy and an MSc. in Computing, both from the University of Wales, Cardiff. His current research interests include computational geometry, quadtree-based spatial indexing techniques and automated map generalisation.