Proximity Search with a Triangulated Spatial Model

CHRISTOPHER B. JONES AND J. MARK WARE

School of Computing, University of Glamorgan, Pontypridd, Mid Glamorgan, CF37 1DL, UK
Email: cbjones@glamorgan.ac.uk

The proximity relations inherent in triangulations of geometric data can be exploited in the implementation of nearest-neighbour search procedures. This is relevant to applications such as terrain analysis, cartography and robotics, in which triangulations may be used to model the spatial data. Here we describe neighbourhood search procedures within constrained Delaunay triangulations of the vertices of linear objects, for the queries of nearest object to an object and the nearest object to an arbitrary point. The procedures search locally from object edges, or from a query point, to build triangulated regions that extend from the source edge or point by a distance at least equal to that to its nearest neighbouring feature. Several geographical datasets have been used to evaluate the procedures experimentally. Average numbers of edge–edge distance calculations to find the nearest line feature edge disjoint to another line feature edge ranged between 15 and 39 for the different datasets examined, while the average numbers of point–edge distance calculations to determine the nearest edge to an arbitrary point ranged between 7 and 35.

Received July 14, 1995; revised April 8, 1998

1. INTRODUCTION

Data models in spatial information systems are usually based either on boundary representations consisting of points, line segments and polygons, or on regular, rectangular subdivisions of space in which each cell is associated with an identity or classification of the phenomena represented. An alternative approach to regular decomposition in which subdivision is independent of the phenomena is structural decomposition, in which the form of the cells is adapted to the boundaries of the spatial objects. These irregular structural decompositions, typically based on triangulations or trapezia, are known to support efficient implementation of computational geometry problems such as point location in a planar subdivision [1]. They have also been shown to provide superior performance in some types of locational queries on spatial databases [2].

The purpose of this paper is to describe proximal search procedures and associated experimental results for linear features, based on the structural decomposition technique of constrained Delaunay triangulation (CDT). Specifically we look at procedures to find the nearest-neighbouring object to a given object and to find the nearest-neighbouring object to an arbitrary point. In the latter case it is assumed that the containing triangle of the point has been found using either conventional containment search spatial indexing methods such as quadtrees, regular grids, and R-trees [3, 4], or via hierarchical structures based on the triangulation [5, 6, 7].

1.1. Motivation

The motivation for developing efficient proximal search procedures based on CDT is twofold. Delaunay triangulations and the CDT are currently used in data structures for representing some types of spatial data and it is of interest therefore to seek the exploitation of the rich spatial proximity properties of these data structures for purposes of local search, rather than maintaining associated or supplementary structures. This is particularly significant when the application involves updates to a main memory spatial representation. It is also the case that some applications depending heavily upon multiple neighbourhood searches may benefit from the use of a data structure which by its nature is based on local proximity relations.

There are several examples of the use of triangulations in spatial data handling. The benefits of triangulations for spatial interpolation have underpinned their use in representing terrain elevation data, in the form of triangulated irregular networks (TINs), whereby the triangles may be regarded as planar facets for linear interpolation, or as the basis for quintic surface interpolation [8, 9]. Triangulated digital terrain models hold considerable promise for integrating elevation data with detailed boundary representations of topographic and geological features [10, 11, 12]. Such terrain models can be used for visualizing landscapes and for performing intervisibility analysis [6, 7, 13]. Triangulations have been shown to be useful in implementing a variety of automated cartographic design procedures, particularly relating to map generalization [14, 15], and their potential for the explicit maintenance of topological relations between a set of two-dimensional geographical point locations has previously been recognized [16, 17]. Triangulations have a long history of use in finite-element modelling for mechanical engineering [18] and they have been used to represent space in robotics applications [19].

1.2. Triangulations for proximal search

The idea of exploiting the structure of a triangulation for proximal search was used in some of the earliest algorithms for constructing triangulations [8]. Proximal search within a triangulation is also employed in some of the applications of triangulations, referred to above, for example in intervisibility analysis [20]. Dickerson and Drysdale [21] and Dickerson *et al*. [22] are notable for drawing attention explicitly to the advantages of triangulations for implementing nearest-neighbour search for the specific cases of fixed radius search for points and for line segments, and for enumerating the smallest distances between pairs of points, and the k nearest-neighbours of points.

In this paper we focus on the problems of neighbouring linear features and the nearest linear feature to an arbitrary point, and present experimental results for several geographical datasets. These results illustrate average case performance and highlight the variation in performance which is related at least partially to the degree of equiangularity of the triangulations of the datasets. Despite this variation, for all datasets studied the average numbers of distance measurements per edge of a source object, and per arbitrary query point, were not found to be more than about double the number of edges of triangles that would be directly connected to either the source edge, or the seed triangle containing the query point, if a triangulation with a vertex valency of six was assumed.

In the remainder of this paper we elaborate in Section 2 on the properties of triangulations and their relationships with Voronoi diagrams. In Section 3 we define the components and relevant characteristics of a constrained triangulated data model. Procedures for determining the nearest object to a specified object and the nearest-neighbouring object to an arbitrary point are described in Section 4. Section 5 presents experimental results. The paper concludes in Section 6.

2. TRIANGULATIONS AND VORONOI DIAGRAMS

For the purpose of determining neighbourhood relations between point locations, the Delaunay triangulation and its dual, the Voronoi diagram, have been well documented [1, 23]. In particular, computation of the Voronoi diagram provides a direct solution to the all nearest-neighbour problem for a set of points [24]. The properties and applications of Delaunay triangulations and Voronoi diagrams have been documented in detail in [25]. Here we summarize briefly the main properties and note characteristics relevant to our algorithms.

Given a set of point sites S, the Voronoi diagram delimits for each site $s_i \in S$ the region of space v_i such that for all locations x inside v_i, the distance $d(x, s_i)$ is less than or equal to the distance $d(x, s_k)$ to all other sites $s_k \in S$. In a Delaunay triangulation, the triangulation edges connect those neighbouring point sites which share a common edge of their respective Voronoi regions. A Delaunay triangulation of a set of points is characterized by the fact that, for each triangle, a circle passing through the vertices of the triangle does not contain any other points in the triangulation.

The characteristics of a Delaunay triangulation can be modified somewhat if the vertices belong to linear or polygonal features, the edges of which are required to be represented in the triangulation. In the resulting CDT [26], constraining edges, corresponding to the line segments, act as barriers to the intervisibility of points. As a consequence, the circumcircle property is modified such that the circle passing through the vertices of any given triangle contains no point of the triangulation that is visible to all three vertices of the triangle. In addition, the property of most equiangular triangulation is maintained, given the constraints [27]. Examples of CDTs of geographical data are illustrated in Figures 5 to 9.

Properties of connectivity of the CDT are exploited in the algorithms presented here. We define an edge-connected set of triangles as one in which every triangle in the set shares at least one edge with another member of the set. We note that in such a set it will therefore always be possible to trace a path between any given pair of triangles that are members of the set by stepping from one triangle to another across triangle edges. Such a path is called an edge-connected path. This leads to the following lemma which underpins the algorithms presented.

LEMMA 2.1. *Given a CDT of the vertices of a set of line segments, in which the triangulation is bounded by the convex hull of the vertices, all nearest-neighbouring pairs of disjoint line segments are accessible to each other by an edge-connected path in the triangulation.*

Proof. This lemma follows from a proof that the entire triangulation is itself an edge-connected set. If it was not such a set there would be at least two subsets of triangles that were connected only by a common vertex. For this to be so, there would need to be no intervening triangle present on either side of the common vertex, in which case the vertex would have to be on the boundary of the triangulation. Since the boundary of a convex hull is not self-intersecting, either the two triangle subsets would have to be edge connected on the inside of the boundary or there would have to be at least one intervening triangle to fill the internal concavity and hence provide an edge-connected path between the two subsets. Both of these situations contradict the proposition of sole vertex connectivity between the triangle subsets.

Given therefore that the entire triangulation is an edge-connected set, we may note that since each pair of nearest-neighbouring disjoint edges must belong to triangles of the triangulation, there must be an edge-connected path between those pairs of triangles. □

2.1. The line Voronoi diagram

It may be noted, in contrast to the approach adopted here, that the Voronoi diagram of line segments provides a solution to the nearest-neighbouring line segment problem, since in a line Voronoi diagram (originally referred to as a generalized Voronoi diagram [28]), nearest-neighbouring segments will share a common boundary of

the neighbouring Voronoi regions. This latter property may be used to construct the dual of the line Voronoi diagram which, analogously with the (non-constrained) Delaunay triangulation of point sites, provides an explicit record of the nearest neighbours of each object. As we show in this paper, search procedures based on the CDT can be used dynamically to determine these nearest-neighbour relations when required, without recourse to constructing the line Voronoi diagram and its dual.

3. A SPATIAL MODEL BASED ON CONSTRAINED DELAUNAY TRIANGULATION

We consider here a spatial model $M = (O, F)$ representing linear objects O and regions of free space F between disjoint objects and internally contained within polygonal-shaped objects. The linear objects consist of one or more sequences of straight line segments that constitute a connected graph. All space within the model is represented by a CDT T of line segment vertices. All line segments of all objects O are constraints within T. Of all edges E in the triangulation, those representing line segments of objects are referred to as real edges E^{r} while all other edges are referred to as virtual edges E^{v}. Hence $E = \{E^{\mathrm{r}}, E^{\mathrm{v}}\}$. The set of real edges belonging to an object o_i is designated by E^{r_i}.

3.1. A data structure for the spatial model

The structure of the triangulation is represented through a set of explicit relations between linear objects and edges, between triangles and edges and between edges and vertices. Linear objects are defined in terms of real edges with each linear object maintaining a pointer to each of its constituent edges. Triangles are described by pointers to constituent edges, each of which are in turn described by pointers to their two vertices. Each edge is associated with additional topological information in the form of a pointer to the triangle lying immediately to its left and a pointer to the triangle lying immediately to its right (that is, the two triangles to which the edge belongs). A Boolean flag is also stored with each edge indicating whether it is a real edge or a virtual edge. Real edges of linear objects reference a pointer to the linear object to which they belong. Each vertex is associated with co-ordinate data in the form of x and y values.

This triangulated data structure conforms to conventional triangulated data structures used in applications referred to in Section 1, although it differs from the simplest of these in referencing parent objects that are defined in terms of triangulation components. For a measure of storage requirements we assume that there are n points in the input line segment data and approximately n real edges. The number of triangles is given by $2n - b - 2$, where b is the number of boundary points. This approximates to $2n$ for large datasets. The number of triangle edges is $3n - b - 3$ which again approximates to $3n$. The number of virtual edges is therefore approximately $3n - n = 2n$. Each triangle requires four data items, consisting of a triangle identifier and three edge pointers. Each virtual edge requires five data items consisting of an edge identifier, two vertex pointers and two triangle pointers (the Boolean flag is regarded as negligible). Each real edge requires six data items consisting of an edge identifier, two triangle pointers, two vertex pointers and an object identifier. Each vertex is associated with a vertex identifier and two co-ordinates. Thus for each input point there is a total of $(4 \times 2 + 5 \times 2 + 6 \times 1 + 3 \times 1)$, i.e. 27, data items.

The data structure is assumed to be associated with a spatial index on the triangles. This may be a temporary structure or, if the triangulation is stored in a database, it is a permanent database spatial index.

4. PROXIMITY SEARCH PROCEDURES

In this section we describe procedures to find the nearest object to a specified object and the nearest object to an arbitrary point. We start by defining the two search problems.

The distance $d_{oo}(o_x, o_y)$ between two objects o_x and o_y is given by the minimum of the distances $d_{ee}(k, l)$ between all pairs of edges $k \in E^{\mathrm{r}_x}$ and $l \in E^{\mathrm{r}_y}$. The distance $d_{ee}(k, l)$ between two edges k and l is given by the minimum of the four distances between a vertex of one edge and the nearest point located on the other edge. The nearest neighbour $o_y \in O$ of an object $o_x \in O$ therefore satisfies the following condition:

$$d_{oo}(o_x, o_y) = \min(d_{ee}(k, l)) \qquad k \in E^{\mathrm{r}_x}; \quad l \in (E^{\mathrm{r}} - E^{\mathrm{r}_x}).$$

The nearest linear object to a point p satisfies the condition that the distance $d_{pe}(p, e)$ between the point p and one of the edges e of the object is the shortest distance between the point and all real edges in E. Thus the distance $d_{po}(p, o_x)$ from point p to the nearest object $o_x \in O$ is given by

$$d_{po}(p, o_x) = \min(d_{pe}(p, e)) \quad e \in E^{\mathrm{r}}.$$

4.1. Algorithm 1. Finding the nearest object to an object

The procedure FindNearestObjectToObject, referred to as Algorithm 1, will find the nearest-neighbouring linear object, or neighbours if there is more than one, of a linear object o_i, referred to as the source object and consisting of the set of real edges E^{r_i}. The nearest neighbour(s) is stored in the list N_i and the distance between this object(s) and o_i is stored in d_{iN}. The technique adopted in finding N_i involves a sequence of local searches from each edge of o_i. The search from an individual source edge stops when the distance between that edge and all inspected neighbouring triangulation edges at the boundary of the local search region exceeds the current value of d_{iN}.

The procedure begins by initializing N_i to empty and setting d_{iN} to a large value. The main loop of the procedure processes each source edge $e \in E^{\mathrm{r}_i}$ of object o_i. For

```
FindNearestObjectToObject (IN: Object oi ; OUT: Object_List Ni, Distance diN)

Initialise Ni to empty
diN ← large value
For each edge e ∈ Eri
    Initialise Search_Queue and Visited_List to empty
    d ← 0.0
    Add (e, d) to Search_Queue
    Add e to Visited_List
    Do while (Search_Queue not empty)
        Take (e1, d1) from head of Search_Queue
        If (d1 ≤ diN)
            For each triangle t edge-connected to e1
                If (t ∉ Visited_List)
                    Add t to Visited_List
                    For each edge e2 ∈ t
                        If (e2 ∉ Visited_List)
                            Add e2 to Visited_List
                            If (EdgesContiguous(e,e2))
                                d2 ← 0.0
                            Else
                                d2 ← EdgeEdgeDistance(e,e2)
                            Endif
                            If (d2 ≤ diN)
                                Add (e2,d2) to Search_Queue
                                otemp ← GetObject(e2)
                                If (otemp ≠ NULL) and (otemp ≠ oi) and (d2 > 0.0)
                                    If (d2 = diN) and (otemp ∉ Ni)
                                        Add otemp to Ni
                                    Else
                                        diN ← d2
                                        Reinitialise Ni to empty
                                        Add otemp to Ni
                                    Endif
                                Endif
                            Endif
                        Endif
                    Endfor
                Endif
            Endfor
        Endif
    Enddo
Endfor

End FindNearestObjectToObject
```

ALGORITHM 1.

a particular source edge e the search begins by placing e on a previously empty queue (Search_Queue) which is to contain triangle edges awaiting processing. Edges are removed from this queue in turn and processing for an individual source edge finishes when there are no more edges on the queue.

On removing an edge e_1 from the queue, a test is performed to determine whether the edge is nearer to the source edge than the current nearest object. If it is nearer then each of its adjacent triangles is processed, provided they have not previously been processed. For an individual triangle t adjacent to e_1 this processing entails examining each constituent edge e_2. For an individual edge e_2, provided it has not previously been processed in the local search, the distance d_2 between e_2 and e is calculated. The number of edge to edge distance calculations is minimized by checking if e_2 is contiguous with, and hence at zero distance from, e using the Boolean function EdgesContiguous. If this is not the case, the function EdgeEdgeDistance is used to calculate d_2.

If d_2 is less than or equal to d_{iN} then e_2 and d_2 are added to Search_Queue, for subsequent processing. If e_2 belongs to an object other than o_i, then if the object is at the same distance as the current value of d_{iN} it is added to N_i,

otherwise N_i is re-initialized with the new object and d_{iN} is updated.

4.1.1. *Correctness of Algorithm 1*

LEMMA 4.1. *Let e_n be an edge belonging to a source linear feature L_i, embedded within a CDT of the vertices of a set of linear features L. For each such edge, Algorithm 1 constructs a surrounding triangle set consisting of the union of the edge-connected triangles processed for that edge and defining a polygonal region of space completely surrounding e_n (illustrated in Figure 1).*

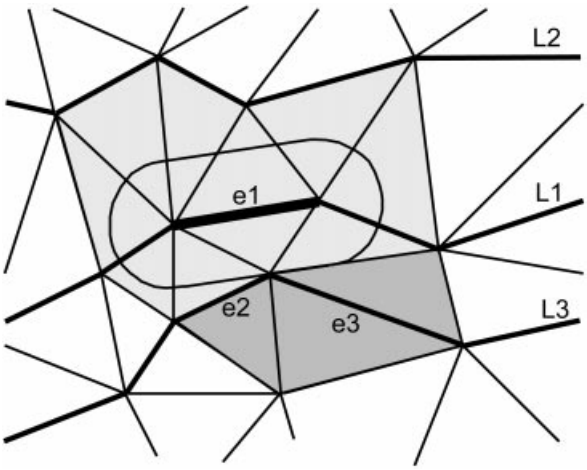


FIGURE 1. All shaded triangles represent the surrounding triangle set in a local search for the nearest neighbour of an edge e_1. The paler shaded triangles constitute the minimum surrounding triangle set. Vertices of the linear features L_1, L_2 and L_3 have been triangulated. The nearest neighbours of edge e_1 (belonging to feature L_1) are edges e_2 and e_3, which are equidistant from e_1 and belong to feature L_3. The oval boundary represents the region extending from the source edge by the distance of the nearest-neighbouring edges.

Proof. Note initially that Algorithm 1 always processes the two edge-connected triangles of a source edge (e_1 in Figure 1). At each end of the source edge is a set of triangles forming a fan around the respective vertex and connected to it by edges that are at zero distance. The fan sets are each internally edge-connected by the zero-distance edges and they are each edge-connected to the two edge-connected triangles of the source edge. All of these triangles that are vertex-connected with the source edge will be processed by Algorithm 1, since they will be found via edge connectivity across edges that are all nearer than the current distance to the nearest neighbour. They form a minimum set of surrounding triangles (the pale shaded triangles in Figure 1) for the source edge and their boundary will consist of a polygon which is disjoint from and therefore contains the source edge. Further triangles (the darker shaded triangles in Figure 1) will be added to the minimum surrounding triangle set, i.e. be processed, if they are edge-connected to the current surrounding set by edges that are nearer than or at equal distance to the candidate nearest neighbour. □

It should be remarked that the size of surrounding triangle set may vary for a particular source edge according to which of the two adjacent triangles is processed first.

LEMMA 4.2. *For an edge $e_n \in L_i$, embedded within a CDT of the vertices of a set of linear features L, Algorithm 1 finds the nearest-neighbouring disjoint linear feature edge $e_n \in (L - L_i)$ in time required to process all triangles of the surrounding triangle set of e_n.*

Proof. Note that processing of the Search_Queue will terminate when the queue is empty and hence when Algorithm 1 can find no more unprocessed edges that are closer than the candidate nearest edge(s). Since the surrounding set defines a polygonal region containing the source edge, for there to be a nearer disjoint real edge than the candidate nearest edge(s), at this stage in processing, there would have to be some part of the surrounding triangle set that was nearer than the candidate. As all edges of all triangles in the surrounding set will have been queued, there cannot be a closer edge than the final candidate edge(s). □

Proof of the correctness of Algorithm 1, in finding the nearest-neighbouring linear feature of another linear feature, follows from the repeated application of individual edge nearest-neighbour search. Processing of the first edge of the source feature finds the nearest disjoint neighbour(s) of that edge. This neighbouring edge (or edges if there is more than one at the same distance) becomes the candidate nearest neighbour against which the neighbouring edges of the second constituent edge are compared. If a neighbouring real edge of the second constituent edge is nearer than the candidate(s) it becomes the new candidate, and processing will continue for that source edge until any nearer object edge(s) are found and until no edge of its surrounding triangle set is nearer than the current candidate(s), as described above for the case of a single edge. Third and subsequent edges of the linear feature will be processed in the same way. On completion, the nearest neighbour(s) will therefore be the minimum distance nearest neighbour(s) of all constituent edges and hence the nearest neighbour(s) of the entire feature.

4.1.2. *Observations on the performance of Algorithm 1*

The algorithm requires a pre-processing step to construct the CDT of the N vertices of the linear features. This takes $O(N \log N)$ time [26]. We also construct a regular grid spatial index on the triangles which requires $O(S)$ time and $O(S + G)$ space where S is the number of triangles and G is the number of grid cells. Each constituent edge of the source linear feature requires distance calculations to all edges of all triangles of the surrounding triangle set. For each edge examined, each possible operation of distance measurement, distance comparison, enqueuing, dequeuing and access to the adjacent triangles is performed in constant time, while the tests for previous processing of edges and triangles take a maximum time proportional to the number of triangles examined (and hence the size of the surrounding set). No edge is processed more than once for each source edge.

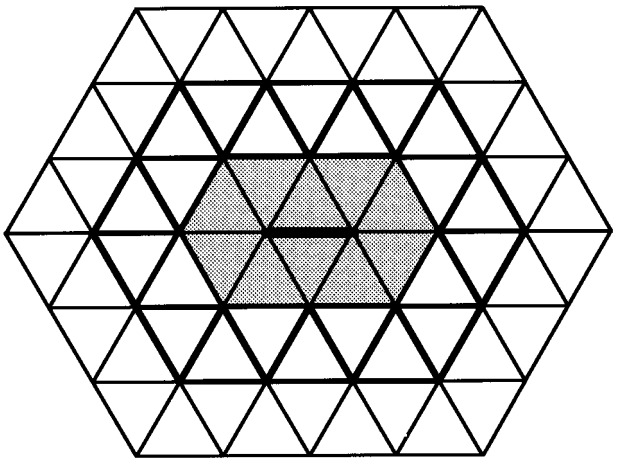


FIGURE 2. Part of a highly equiangular triangulation centred on a single source edge (highlighted in bold). The shaded triangles represent the minimum surrounding triangle set. All bold shaded edges, other than the source edge, indicate those edges that might be inspected with Algorithm 1 in a worst case scenario for a highly equiangular triangulation.

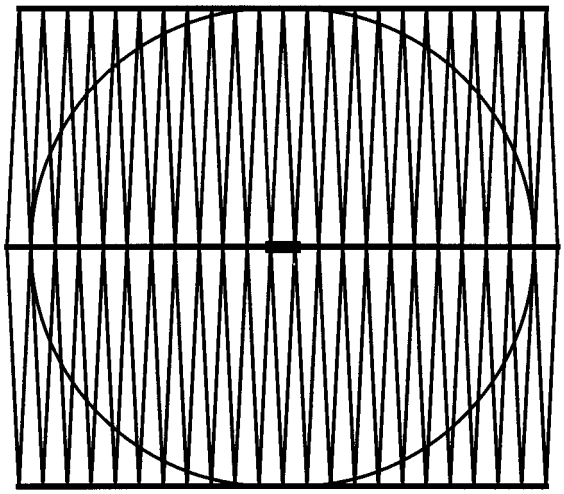


FIGURE 3. A regular non-equiangular triangulation of three parallel sets of edges belonging to three linear features, with an aspect ratio of about 10. In this 'bad case' scenario all triangles illustrated would need to be processed with Algorithm 1 in order to construct the surrounding triangle set for the central edge (highlighted in bold) of the middle linear feature. The oval represents the region extending from the source edge by the distance of the nearest neighbour(s). Note that if the triangulation extended beyond the upper and lower lines then a further four triangles (not illustrated) would be added to the surrounding triangle set.

The minimum number of edges examined will correspond to those of the minimum surrounding triangle set and hence, in the case of the average number of edges connected to a vertex being six [1], this would represent 18 edges of which only eight required distance calculations, the others being connected to the source edge (Figure 2).

The surrounding triangle set must at least cover the region of space extending from the source edge by a distance equal to that of the nearest neighbour to the edge, so that no part of the polygonal boundary of the surrounding triangle set can be nearer than the nearest neighbour. In a highly equiangular triangulation the number of triangles required to cover the search region would be expected to differ little from the minimum surrounding triangle set as illustrated in Figure 2 (in which the minimum set is shaded). For a highly equiangular triangulation, in the worst case it would be necessary in Algorithm 1 to inspect all edges of triangles edge-connected to the boundary of the minimum surrounding set, in which case 44 edge–edge distance measurements would be required.

With decreasing equiangularity the need to search beyond the minimum surrounding set would increase and hence the number of triangles inspected would tend to increase. Figure 3 illustrates a non-equiangular triangulation of three line features that are approximately parallel and in which the aspect ratio (between the smallest height and the longest edge) of the triangles is about 10. In the hypothetical example it would be necessary to perform distance calculations on 158 triangle edges in order to process the surrounding triangle set. Although aspect ratios of about 10 are quite common in datasets that we have examined, the very regular distribution of data in the figure is not. The configuration in Figure 3 was chosen to represent a relatively 'bad case' with regard to the search procedure. Note that if the real edges were long relative to the minimum triangle height, with a similar arrangement of parallel features, this would lead to a much lower number of edge distance calculations. If the example configuration of data represented the entire dataset then a worst case time complexity of $O(N)$ time therefore applies, where N is the number of triangle edges.

The emphasis of this paper is on experimental evaluation of the presented procedures and no attempt is made here to carry out a theoretical analysis of average performance. The experimental results presented subsequently do, however, support the proposition that the situation illustrated in Figure 3 represents an 'outlier' and that the average performance is much nearer to the processing required to inspect the minimum triangle set.

4.2. Finding the nearest object to a point

Here we describe procedure FindNearestEdgeToPoint (Algorithm 2) which finds the nearest real edge to a point p lying in a triangle t. The procedure FindNearestEdgeToPoint takes as input a query point p and the identity of the triangle t that contains the point, and outputs a list E_p containing the nearest edge (or edges if there is more than one at the same distance) and the corresponding distance d_p. The containing triangle is found in our implementation with the spatial index on the individual triangles. The procedure starts by initializing to empty a Search_Queue and a Visited_List (for processed triangles and edges), and

```
FindNearestEdgeToPoint (IN: Point p, Triangle t; OUT: Edge_List Ep, Distance dp)

Initialise Ep, Search_Queue, and Visited_List to empty
dp ← large value
For each edge e ∈ t
    d ← PointEdgeDistance(p, e)
    If (d ≤ dp)
        Add (e, t, d) to Search_Queue
        If (e ∈ E^r)
            If (d = dp)
                Add e to Ep
            Else
                dp ← d
                Reinitialise Ep to empty
                Add e to Ep
            Endif
        Endif
        Add e to Visited_List
    Endif
Endfor
Add t to Visited_List
Do while (Search_Queue not empty)
    Take (e, t, d) from head of Search_Queue
    If (d ≤ dp)
        t1 ← TriangleEdgeConnect(t, e)
        If (t1 ∉ Visited_List)
            For each edge e1 ∈ t1
                If (e1 ∉ Visited_List)
                    d1 ← PointEdgeDistance(p, e1)
                    If (d1 ≤ dp)
                        Add (e1, t1, d1) to Search_Queue
                        If (e1 ∈ E^r)
                            If (d1 = dp)
                                Add e1 to Ep
                            Else
                                dp ← d1
                                Reinitialise Ep to empty
                                Add e1 to Ep
                            Endif
                        Endif
                    Endif
                Endif
                Add e1 to Visited_List
            Endfor
            Add t1 to Visited_List
        Endif
    Endif
Enddo

End FindNearestEdgeToPoint
```

ALGORITHM 2.

setting the distance variable d_p to a large value (greater than the range of the data).

In the first stage of the procedure, each edge of the seed triangle is examined. For each edge, the distance to p is calculated with function PointEdgeDistance. If the distance is less than or equal to the current value of d_p then the edge's identity and the corresponding distance are queued, along with the identity of the seed triangle. Furthermore, if the edge is real, the list E_p is re-initialized with the edge's identity if the distance is less than the current value of d_p, otherwise, if it is equal, the identity of the edge is added to the list.

The main body of the procedure consists of a loop which processes edges on the Search_Queue until that queue is

empty. Dequeuing an edge is followed by a comparison of its distance from p with the current value of d_p. If the dequeued edge is further away, then no further processing is performed on it and the next edge is dequeued. Processing of a dequeued edge that is nearer than or equal in distance to the candidate nearest real edge starts by determining the identity of the triangle t_1 on the opposite side of the edge from the initial triangle to which it belonged, using function TriangleEdgeConnect. Provided that t_1 has not previously been processed, the distance to p from each previously unprocessed edge $e_1 \in t_1$ is found. If e_1 is nearer than or at equal distance to the candidate nearest real edge(s) then it is queued along with its distance and the identity of t_1, and if it is a real edge E_p and d_p are updated as in the first stage.

4.2.1. *Correctness of Algorithm 2*

Proof of the correctness of Algorithm 2 is similar to, though simpler than, that for Algorithm 1. Thus Lemma 4.3 refers to a surrounding triangle set which we use to demonstrate that the algorithm searches a region of space that extends in all directions from the point by a distance equal to or greater than the distance to the nearest-neighbouring real edge.

LEMMA 4.3. *Given a CDT of the vertices of a set of linear features L and a query point p, Algorithm 2 constructs a surrounding triangle set consisting of the union of the edge-connected processed triangles and defining a polygonal region that contains p.*

Proof. The proof of Lemma 4.3 follows from the fact that processing of the query point starts by identifying a triangle that contains p and constitutes the minimum surrounding set. Individual further triangles are processed, and hence added to the surrounding triangle set, if they are edge-connected to the boundary of the surrounding triangle set and if the connecting edge is nearer to p than is the candidate nearest neighbour (or neighbours if there is more than one at an equal distance). Processing of the Search_Queue will terminate when there are no edges on the queue that are nearer than the candidate nearest edge. At this stage all edges inside and on the boundary of the polygonal region will have been examined and hence there are no real edges belonging to the region nearer than the candidate nearest edge and no virtual edges on the boundary that are nearer. □

4.2.2. *Observations on performance of Algorithm 2*

The lower bound on the time complexity of Algorithm 2 is O(1) and corresponds to the situation in which the nearest-neighbouring edge is found in stage one as a result of processing the seed triangle. Thus one triangle and its three edges will have been processed. If the seed triangle includes a virtual edge that is nearer to the query point than is a real edge of the triangle, or if there are no real edges, then additional triangles will be processed. A worst case scenario can be envisaged in which the query point was located half way between two parallel rows of real edges that were very short compared with the separation between the rows. If the entire dataset consisted only of two such rows, the points of which were collinear, and the length of the rows was equal to their separation distance, then the surrounding triangle set could include all triangles in the triangulation, leading to a time complexity of O(N).

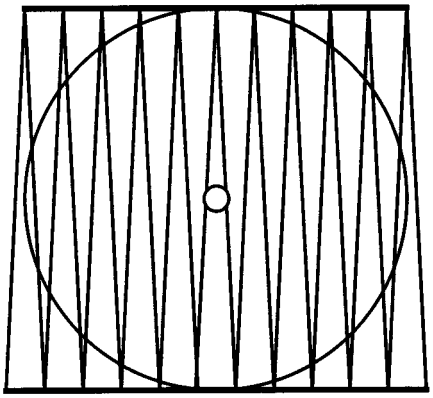
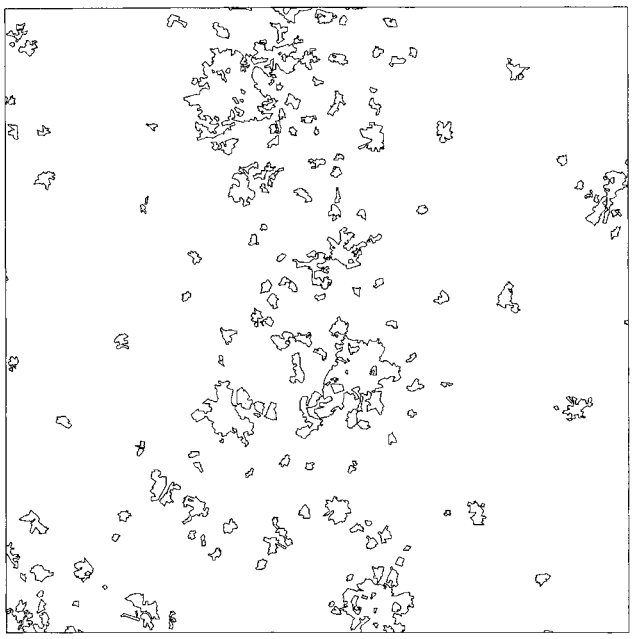


FIGURE 4. A 'bad case' for Algorithm 2 (nearest edge to a point) in which the query point lies equidistant from two parallel linear features in a triangulation with an aspect ratio of about 10. All triangles illustrated would be processed to construct the surrounding triangle set. If the triangulation extended beyond the upper and lower line features, the surrounding triangle set would include three additional triangles (not illustrated) above and below the location of the query point.

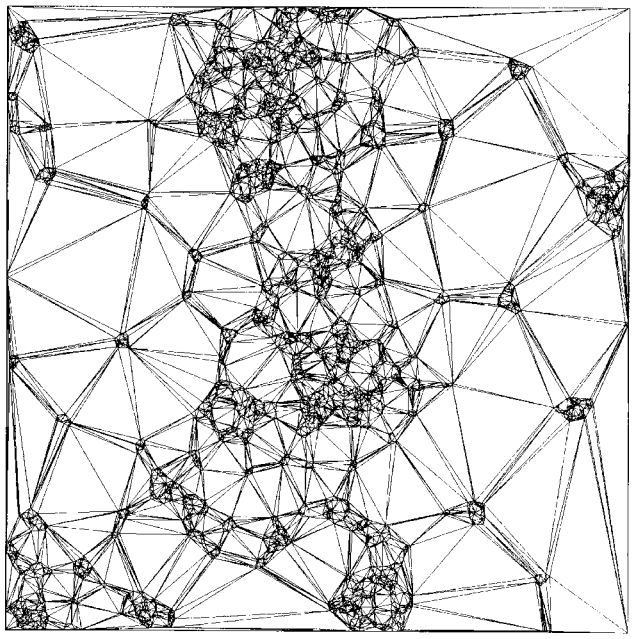
In practice, with a more uniform distribution of data, as is encountered in the geographical datasets used in our experiments, the performance can be expected to vary as a function of the degree of equiangularity. Figure 4 illustrates the surrounding triangle set in a situation in which a query point is located in the centre of a region with parallel sets of short edges and an aspect ratio of about 10. The nearest real edge is located in the seed triangle, but it is necessary to build the surrounding triangle set to cover a region of space with a radius equal to the distance to the nearest neighbour. In this case it can be seen that 21 triangles would be processed, along with 43 edges. In the experimental results presented below it can be seen that an aspect ratio of 10 is quite typical and that average performance is better than this hypothetical 'bad case' for all datasets examined.

5. EXPERIMENTAL RESULTS

Both of the algorithms presented here were tested experimentally using a variety of geographical datasets containing topographic features including administrative boundaries (very irregular), land use categories, roads and buildings. Figures 5–9 illustrate several of the datasets. The characteristics of all the datasets employed, numbered 1–13, are summarized in Table 1. All except number 12 are Bartholomews topographic digital map data for Great Britain based on source scales of 1:5000 and 1:250,000. Dataset 12 is French IGN BDTopo data, based on a source scale of 1:25,000. Datasets 4 and 5 (roads) are derived from dataset 3 by filtering vertices. Similarly, dataset 13 is derived from dataset 7 by filtering vertices.

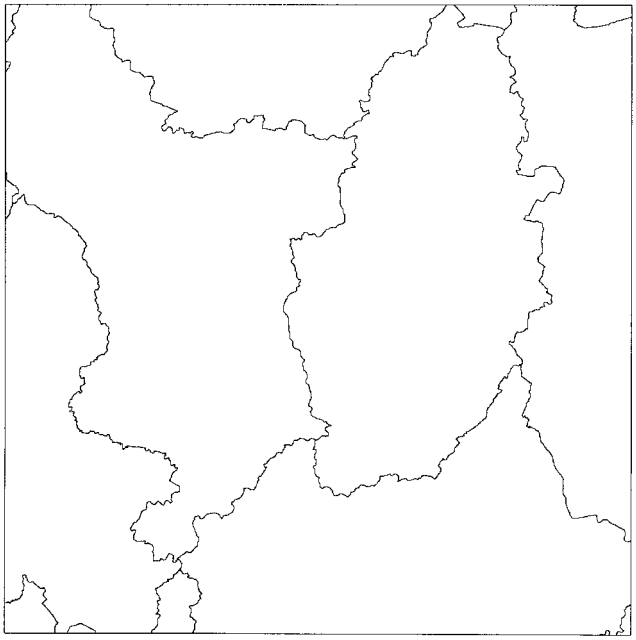


(a)

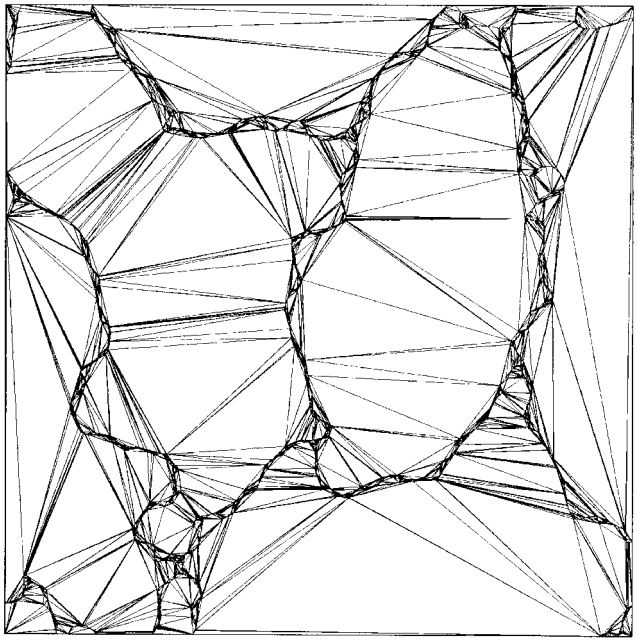


(b)

FIGURE 5. Digital map dataset 1 (a) and its CDT (b). Urban boundaries from Bartholomews Great Britain 1:250,000 digital map.



(a)

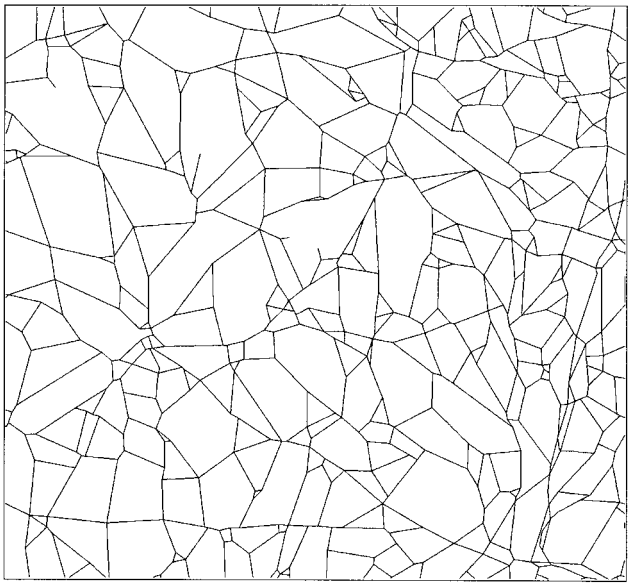


(b)

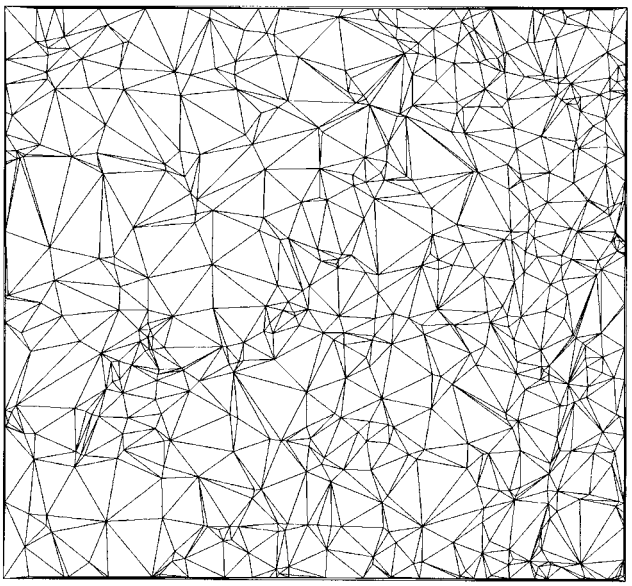
FIGURE 6. Digital map dataset 2 (a) and its CDT (b). Administrative boundaries from Bartholomews Great Britain 1:250,000 digital map.

5.1. Experimental results for nearest-neighbouring line edges

The testing of Algorithm 1 was performed using datasets consisting largely of isolated objects such as buildings and categories of land use. It was not tested on the datasets consisting of networks of linear features such as roads, since most features in these datasets were connected (and hence at zero distance from each other), rather than disjoint from the nearest neighbours as is assumed by the algorithm. The results report on finding the nearest disjoint edge to an individual edge corresponding to a single iteration of the main loop of Algorithm 1. Statistics for nearest linear object to another linear object are not reported here as the individual searches depend (linearly) on the size of the source object and hence, when averaged for multiple searches, mask the performance of the core edge–edge search procedure.

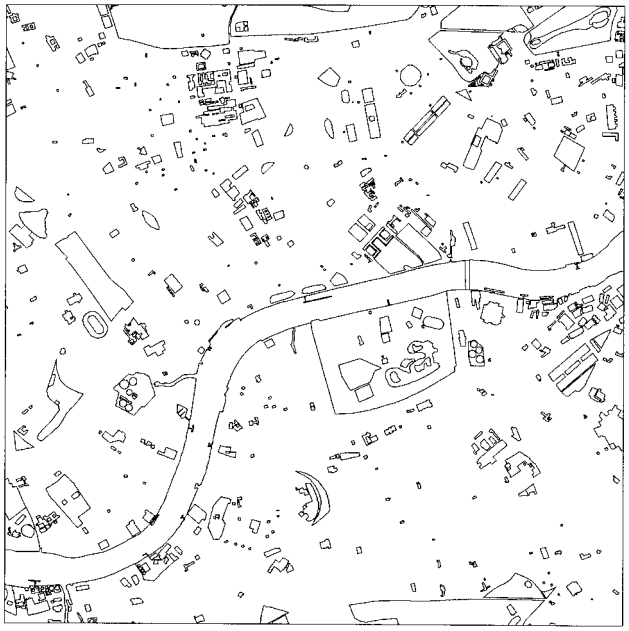


(a)

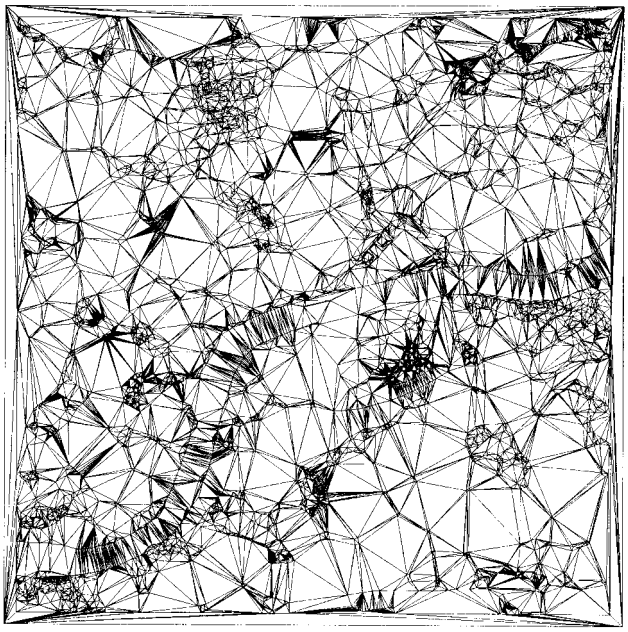


(b)

FIGURE 7. Digital map dataset 5 (a) and its CDT (b). Roads (filtered data) derived from Bartholomews Great Britain 1:250,000 digital map.



(a)



(b)

FIGURE 8. Digital map dataset 11 (a) and its CDT (b). Buildings and land use categories from Bartholomews Great Britain 1:5000 digital map.

Results are summarized in Table 2. For each dataset the number of searches was determined by the total number of real edges in the dataset, and statistics reported consist of the average numbers of edge–edge distance calculations, with their corresponding minimum, maximum and standard deviations, and the average time in seconds per source edge, again with minimum, maximum and standard deviations. Average numbers of edge–edge distance calculations (equivalent to numbers of edges 'processed' per edge) ranged between about 15 and 39. The worst results correspond to the urban area boundaries of dataset 1 consisting of small irregular shaped polygons in which the object size was small relative to the separation between objects, and with an average triangulation aspect ratio of about 7. The best results (dataset 2) were for data that were relatively uniformly distributed with a triangulation aspect ratio of about 4. Minimum numbers of calculations per edge ranged between three and five. Maximum numbers of edge–edge calculations (116 to 324) were very roughly 10 times higher than the average values. Standard deviations for edge–edge calculations were similar in magnitude to their corresponding average values.

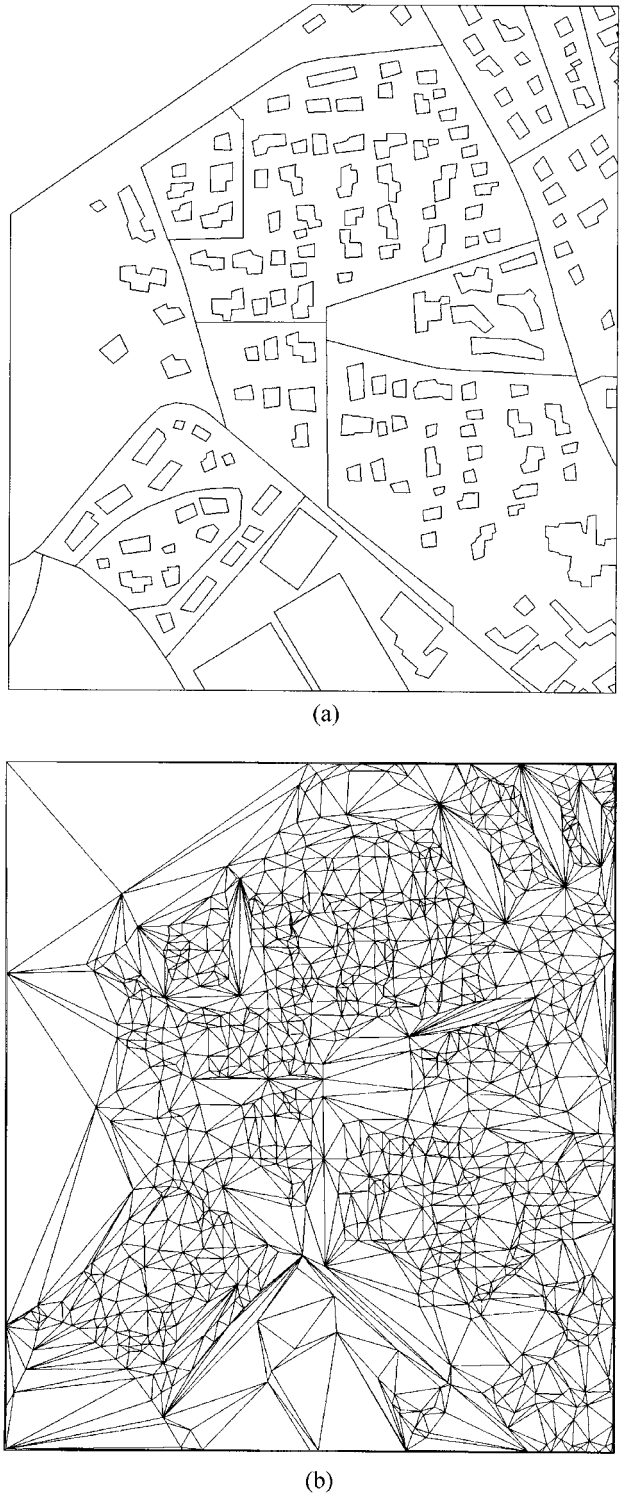


FIGURE 9. Digital map dataset 12 (a) and its CDT (b). Buildings and streets from IGN France BDTopo 1:25,000 digital map.

All timings were carried out on a SPARC 20. Average CPU times were directly proportional to numbers of edges processed and ranged for the different datasets between 0.00044 and 0.0014 s per source edge. Minimum timings were, as indicated by the minimum edge–edge calculations, similar for the different datasets, being between about 0.00008 and 0.0001 s. Maximum values were also similar ranging from about 0.021 to 0.069 s. Standard deviations for the CPU timings were mostly two to three times the average.

5.2. Experimental results for nearest edge to a point

Experiments for Algorithm 2 were carried out using all 13 datasets and are tabulated in Table 3. For each dataset, 10,000 nearest neighbour searches were performed, based on a regular grid of query points. Average numbers of point–edge distance calculations ranged between 7 and 35. The best performance datasets were the third roads dataset (5) and the buildings and street dataset (12) both of which have relatively equiangular triangulations with aspect ratios about 4. Minimum numbers of point–edge distance calculations were always three, corresponding to the theoretical analysis of a lower bound. Maximum numbers ranged from 32 to 287. Standard deviations of the numbers of distance calculations were between about half and one times the average values, for the different datasets.

Timings on a SPARC 20 were in the average case approximately linearly related to the distance calculations. Average times ranged from 0.00005 to 0.00030 s. Minimum times were consistent at about 0.000015 s, while maximum times were in the range 0.021 to 0.097 s. Standard deviations of the times ranged between about two and seven times the average values.

6. CONCLUDING REMARKS

This paper has presented search procedures to determine nearest-neighbour relationships for linear features using a CDT of their vertices. Proximal queries to determine the nearest linear object(s) to a given object, and the nearest linear object(s) to an arbitrary point, exploit the local connectivity of the triangulation to carry out a search through the triangles connected to the query object or to the triangle containing a query point.

The experimental results presented here indicate that, for the geographical datasets considered, average case nearest-neighbour queries require distance calculations only to edges in the immediate neighbourhood of the source edge or query point. A comparison of numbers of distance calculations (which dominate performance) for the object–object search procedure (Algorithm 1) may be made between the average experimental results obtained here and the expected results for an ideal hypothetical constrained triangulation in which it was known that nearest disjoint neighbouring linear objects were always connected directly by a non-line-segment (virtual) triangulation edge. This hypothetical situation may be regarded as similar to the dual of a line Voronoi diagram. For the query of finding the nearest disjoint edge to a given edge, the hypothetical number of (non-zero) edge–edge distance calculations in a triangulation with assumed valency of six edges per vertex would be a minimum of eight and a maximum of 30 (see Figure 2). The average numbers of edge–edge distance calculations in the experiments presented here

TABLE 1. Description of data sets.

Data set	Description	Scale	Objects	Edges	Vertices	Triangles	Aspect ratio
1	urban boundaries	250,000	238	5701	5704	11,402	7.16
2	urban boundaries	250,000	79	4408	4397	8788	32.37
3	roads1	250,000	883	4028	3768	7530	8.05
4	roads2	250,000	883	2376	2116	4226	3.88
5	roads3	250,000	883	882	623	1240	4.01
6	buildings	5000	944	5567	5492	10,978	10.79
7	land use	5000	518	5969	5934	11,862	85.63
8	buildings+land use	5000	1462	11,400	11,212	22,418	20.71
9	buildings	5000	640	4894	4857	9708	14.12
10	land use	5000	455	4928	4886	9766	25.73
11	buildings+land use	5000	1095	9742	9638	19,270	14.49
12	buildings+streets	25,000	337	2151	1074	2142	4.17
13	land use	5000	518	2183	2248	4490	10.57

TABLE 2. Results for nearest edge to edge search.

	Edge–Edge calculations per edge				Time taken per edge			
Data set	Average	Min	Max	s.d.	Average	Min	Max	s.d.
1	39.01	3	315	42.25	0.00139	0.000081	0.025005	0.002788
6	28.18	4	324	40.4	0.001008	0.000096	0.026221	0.003011
7	24.86	4	274	36.24	0.000865	0.000096	0.023845	0.00244
8	18.88	3	253	15.95	0.000505	0.000076	0.02204	0.000909
9	29.16	4	238	25.44	0.000813	0.000098	0.22396	0.001316
10	28.23	3	267	38.64	0.00096	0.000077	0.021684	0.002596
11	24.26	4	273	24.75	0.000684	0.000096	0.02157	0.001406
12	14.94	5	116	8.19	0.000443	0.000119	0.069829	0.002311
13	17.51	4	73	12.57	0.00044	0.000096	0.021827	0.000724

TABLE 3. Results for nearest object to arbitrary point search.

	Point–Edge calculations per query				Time taken per query			
Data set	Average	Min	Max	s.d.	Average	Min	Max	s.d.
1	25.16	3	69	10.85	0.000181	0.000016	0.022886	0.000371
2	34.87	3	106	16.44	0.00029	0.000014	0.021521	0.000472
3	14.21	3	94	9.49	0.000092	0.000015	0.023429	0.000338
4	11.41	3	92	8.19	0.000073	0.000014	0.030646	0.000391
5	7.24	3	80	6.63	0.000046	0.000014	0.021003	0.000253
6	18.39	3	116	9.05	0.000119	0.000015	0.022719	0.000294
7	30.07	3	287	28.59	0.000303	0.000015	0.02249	0.000719
8	20.25	3	287	20.42	0.000173	0.000015	0.038024	0.000614
9	20.66	3	125	10.99	0.00014	0.000015	0.020633	0.000302
10	23.64	3	118	16.93	0.000193	0.000015	0.097551	0.00104
11	18.01	3	136	11.6	0.000122	0.000015	0.021454	0.0003
12	8.84	3	32	5.02	0.000052	0.000014	0.030721	0.000376
13	17.13	3	44	7.56	0.000107	0.000014	0.021157	0.00281

range from 15 to 39 for all datasets examined. The same type of comparison cannot be made for Algorithm 2, since the number of triangles between an arbitrary point and the nearest object would still not be constrained in the hypothetical triangulation. However, it may be noted that the average numbers of point–edge distance calculations ranged between seven and 35 for the different datasets.

Average timings on a SPARC20 for the nearest-neighbouring edges queries required 1.4 ms in the 'worst' dataset and 0.4 ms in the 'best', while for the nearest edge to an arbitrary point, average timings for the different datasets ranged between 0.05 and 0.1 ms.

In summary, the results appear to confirm that triangulated data models do provide a reasonable basis for implementing proximal search queries. Performance is clearly variable between datasets. This, however, will also be the case for

search procedures implemented using conventional spatial indexing methods. The algorithms presented here are simple to implement and may be regarded as of most potential benefit for applications in which triangulated spatial models may already have been constructed.

The data structure described here incurs a storage overhead in the form of the triangulation. In the case of applications that employ large spatial databases, however, it may not be necessary to store the explicit triangulation permanently. Thus a local triangulation may be constructed for a user-specified window on which it is assumed that the user may wish to perform multiple operations requiring spatial query [12].

ACKNOWLEDGEMENTS

The digital map data used in dataset numbers 1 to 11 and number 13 employed © Bartholomew Digital Data. Reproduced with permission of HarperCollins Publishers. http://www.bartholomewmaps.com

REFERENCES

[1] Preparata, F. P. and Shamos, M. I. (1988) Computational Geometry. *Texts and Monographs in Computer Science*. Springer-Verlag, New York.

[2] Kriegel, H.-P., Horn, H. and Schiwietz, M. (1991) The performance of object decomposition techniques for spatial query processing. *Advances in Spatial Databases: Proc. 2nd Symp. on the Design and Implementation of Large Spatial Databases (SSD'91). Lecture Notes in Computer Science*, **525**, 257–276. Springer-Verlag.

[3] Guttman, A. (1984) R-trees: a dynamic index structure for spatial searching. *Proc. ACM SIGMOD Int. Conf. on the Management of Data*, pp. 47–57. ACM, Boston, MA.

[4] Sellis, T., Roussopoulos, N. and Faloutsos, C. (1987) The R^+ tree: a dynamic index for multidimensional objects. *13th Int. Conf. on Very Large Databases*, pp. 507–518.

[5] Kirkpatrick, D. (1983) Optimal search in planar subdivisions. *SIAM J. Comput.*, **12**, 28–35.

[6] De Floriani, L. (1989) A pyramidal data structure for triangle-based surface description. *IEEE Comput. Graphics Applic.*, **9**, 67–78.

[7] De Floriani, L., Gattorna, G., Marzano, P. and Puppo, E. (1994) Spatial queries on a hierarchical terrain model. *6th Int. Symp. on Spatial Data Handling SDH 94*, pp. 819–834. International Geographical Union, Edinburgh.

[8] Gold, C. M., Charters, T. D. and Ramsden, J. (1977) Automated contour mapping using triangular element data structures and an interpolant over each triangular domain. *Comput. Graphics (ACM)*, **2**, 170–175.

[9] Peucker, T. K., Fowler, R. J., Little, J. J. and Mark, D. M. (1978) The triangulated irregular network. *ASP/ACSM Digital Terrain Models (DTM) Symp.*, pp. 516–540. ACSM, Falls Church, VA.

[10] Kraak, M. J. (1993) Cartographic terrain modeling in a three-dimensional GIS environment. *Cartography Geogr. Inf. Syst.*, **20**, 13–18.

[11] Jones, C. B. (1989) Data structures for three-dimensional spatial information systems in geology. *Int. J. Geogr. Inf. Syst.*, **3**, 15–31.

[12] Jones, C. B., Kidner, D. B. and Ware, J. M. (1994) The implicit triangulated irregular network and multiscale spatial databases. *Comput. J.*, **37**, 43–57.

[13] De Floriani, L. and Puppo, E. (1988) Constrained Delaunay triangulation for multiresolution surface description. *9th Int. Conf. on Pattern Recognition*, Rome, pp. 566–569. IEEE, Washington, DC.

[14] DeLucia, A. and Black, T. (1987) A comprehensive approach to automatic feature generalization. *Proc. 13th Int. Cartographic Conf.*, pp. 169–191. International Cartographic Association, Mexico.

[15] Jones, C. B., Bundy, G. Ll. and Ware, J. M. (1995) Map generalization with a triangulated data structure. *Cartogr. Geogr. Inf. Syst.*, **22**, 317–331.

[16] Frank, A. and Kuhn, W. (1986) Cell graphs: a provable correct method for the storage of geometry. *2nd Int. Symp. on Spatial Data Handling*, Seattle, WA, pp. 411–436. International Geographical Union, Williamsville, NY.

[17] Egenhofer, M. and Jackson, J. (1989) A topological data model for spatial databases. *Symp on the Design and Implementation of Large Spatial Databases. Lecture Notes in Computer Science*, **409**, 271–286. Springer-Verlag, Berlin.

[18] Baehmann, P. L., Wittchen, S. L., Shephard, M. S., Grice, K. R. and Yerry, M. A. (1987) Robust, geometrically based, automatic two-dimensional mesh generation. *Int. J. Numer. Methods Engng.*, **24**, 1043–1078.

[19] Storer, J. A. and Rief, J. H. (1994) Shortest paths in the plane with polygonal obstacles. *J. Assoc. Comput. Mach.*, **41**, 982–1012.

[20] De Floriani L. and Magillo, P. (1994) Visibility algorithms on triangulated digital terrain models. *Int. J. Geogr .Inf. Syst.*, **8**, 13–41.

[21] Dickerson, M. T., Drysdale, R. L. S. and Sack, J.-D. (1992) Simple algorithms for enumerating interpoint distances and finding nearest neighbours. *Int. J. Comput. Geom. Appl.*, **2**, 221–239.

[22] Dickerson, M. T. and Drysdale, R. S. (1990) Fixed-radius near neighbours search algorithms for points and segments. *Inf. Proc. Lett.*, **35**, 269–273.

[23] Aurenhammer, F. (1991) Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Comput. Surveys*, **23**, 345–405.

[24] Shamos, M. I. and Hoey, D. (1975) Closest-point problems. *16th Ann. IEEE Symp. on Foundations of Computer Science*, pp. 151–162. IEEE Computer Society Press, New York.

[25] Okabe, A., Boots, B. and Sugihara, K. (1992) *Spatial Tessellations—Concepts and Applications of Voronoi Diagrams*. Wiley, Chichester.

[26] Chew, L. P. (1989) Constrained Delaunay triangulations. *Algorithmica*, **4**, 97–108.

[27] Lee, D. T. and Lin, A. K. (1986) Generalized Delaunay triangulation for planar graphs. *Discrete Comput. Geom.*, **1**, 201–217.

[28] Lee, D. R. and Drysdale, R. L. (1981) Generalization of Voronoi diagrams in the plane. *SIAM J. Computing*, **10**, 73–87.