The Application of Oct-Tree Terrain Models to Real-Time Aircraft Flight Path Planning

D. J. Allerton and M. C. Gia

(*College of Aeronautics*, *Cranfield University*)

This paper outlines a technique to represent terrain using tree structures, based on Morton ordering to avoid the use of pointers. This approach enables terrain data to be organised in a hierarchical form affording a trade-off between the speed of access to the terrain database and resolution of the terrain data extracted from the tree. A set of database access algorithms is developed that form the basis of path extraction needed for real-time mission management. Several examples are presented to illustrate the performance of the routeing algorithms developed in the paper.

KEY WORDS

1. Data (DTED). 2. Planning. 3. Air Navigation.

1. INTRODUCTION. The combination of accurate navigation sensors and digital terrain elevation data (DTED) provides the basis of terrain reference navigation (TRN). In terrain following and terrain avoidance applications, the flight path of an aircraft is optimised, either to minimise the height above terrain or to ensure an adequate margin of safety above the terrain. In these applications, the DTED in the immediate vicinity of the current flight path is accessed, in order to determine the desired flight path. However, in mission management, where it is necessary to compute a complete flight plan for a mission, it may be necessary to search large regions of a DTED in order to extract a safe or optimal flight plan.

DTEDs used in current TRN systems are normally provided in the form of digital databases (Henry and Milligan, 1992; Priestly, 1990) consisting of a regular grid of terrain elevation points, typically mapped at intervals ranging from 10 m to 100 m. Terrain data stored in this form occupies large amounts of memory. For example, a region 10000 km by 10000 km, mapped at 100 m resolution, contains 10^{10} grid points. The computation inherent in continuously accessing this number of grid points in real-time is formidable. For an airborne mission management system, where it may be necessary to re-route a mission within a few seconds, it is essential to reduce the number of operations to access the DTED or to compress the data of the DTED. Moreover, there is no topological structure in a DTED represented in this form, and the extraction of geometric properties from the DTED can entail accessing large numbers of grid points.

A quad-tree can be used to represent a two-dimensional region, typically in the form of a $2^n \times 2^n$ binary array (Gargantini, 1982; Klinger, 1976; Lauzon *et al* ., 1985; Samet, 1990). If a node represents a region with a common set of properties, it is a leaf node of the tree. For example, if all the terrain represented by a node is above

https://doi.org/10.1017/S037346330000103X Published online by Cambridge University Press

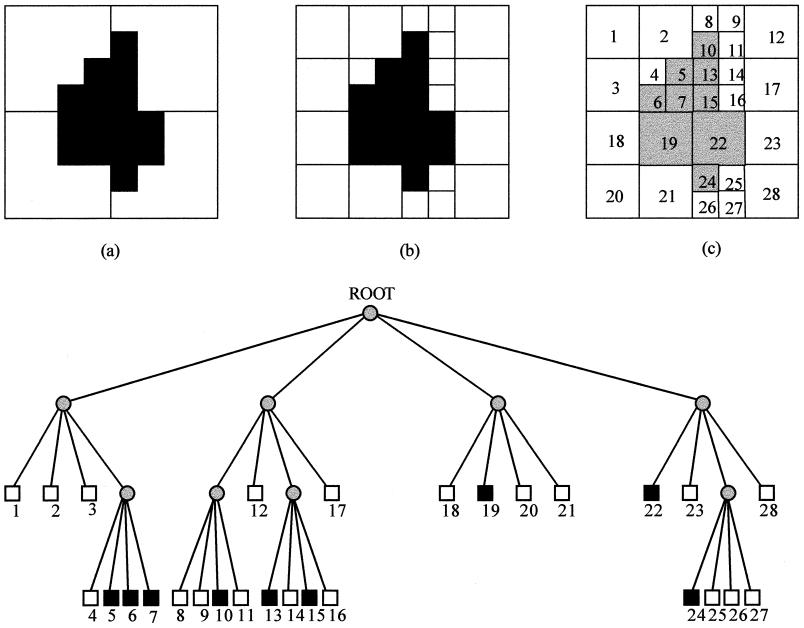

Figure 1. An example of a quad-tree.

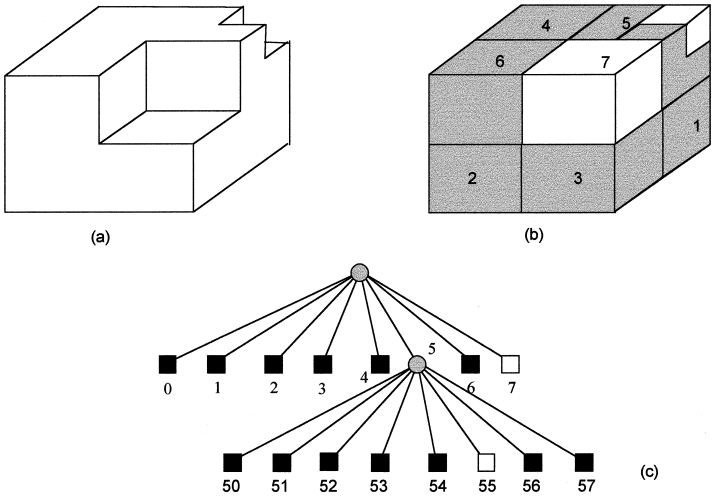

Figure 2. An example of an oct-tree.

100 m elevation or if it is all below 100 m, this node is a leaf node of the quad-tree describing the terrain with reference to 100 m. Otherwise, the node is sub-divided into four quadrants and this process of recursion continues until a quadrant satisfies the conditions for a leaf node. In practice, large regions of terrain can be represented by a single quadrant or node of the tree, as shown in Figure 1. This approach can be extended to three-dimensional objects, represented by an oct-tree, where a $2^n \times 2^n \times 2^n$ array is sub-divided into octants (Chen and Huang, 1988; Gargantini, 1982). If the elements of an octant share a common property, the oct-tree terminates, otherwise

https://doi.org/10.1017/S037346330000103X Published online by Cambridge University Press

eight further sub-octants are generated to represent the octant in more detail, as shown in Figure 2.

Oct-trees offer several advantages for applications in real-time TRN (Allerton and Gia 1996; Gia, 1994):

(a) Construction of an oct-tree from a DTED defined as a regular grid of elevation points is straightforward,
(b) The topology of the terrain is embedded in the structure of the oct-tree,
(c) Mapping between nodes of an oct-tree and a DTED co-ordinate (and vice-versa) is straightforward,
(d) An oct-tree exploits the inherent redundancy in terrain data; regions of terrain with common properties can be merged into single nodes, resulting in compression of the DTED and a reduction in the storage requirements for the DTED,
(e) An oct-tree represents terrain at a coarse resolution toward the root of the tree and at a high resolution (fine level of detail) towards the leaf nodes; this feature affords a trade-off between performance and accuracy, allowing tree access operations to be performed at an appropriate level of the tree.

The compression of terrain data in a form that simplifies the operations to access objects in a DTED is essential in real-time mission management where the extraction of a safe or efficient flight plan from a DTED may be required within a few seconds but may necessitate search strategies that access large areas of the DTED.

Generation of a flight plan involves the computation of a path between a start point and a goal point. In practice, topological constraints may be applied to the set of possible paths and, if more than one path exists, then it is necessary to determine the optimal path. Most flight-planning algorithms consist of three phases (Brook, 1983; Mitchell, 1988):

(a) Extraction of the potential obstacles from the DTED,
(b) Generation of the set of all the possible paths that avoid the obstacles,
(c) Searching the set of paths to locate a path that satisfies specific constraints.

These approaches are based on the assumption that the navigation space (containing terrain, obstacles and threats) is static and is known at the outset of the flight planning algorithm (Kamphampti *et al*., 1986; Lozano-Perez and Wesley, 1979; O'Dunlaing and Yap, 1982; Schwartz and Sharir, 1988). However, in mission management systems, although the DTED is constant, the objects, obstacles and threats can vary with operational constraints. The approach adopted in this paper is based on a method where the graph of the search space is organised according to terrain altitude (Meng, 1987) rather than methods that embed pre-computed cost indices in the graph of the search space (Chan and Foddy, 1985; Lizza and Lizza, 1985).

2. MODELLING OF THE NAVIGATION SPACE.

The flight path planning algorithm described in this paper is performed on a variant of the oct-tree known as the terrain oct-tree (Gia, 1994). The terrain oct-tree is organised as a linear list, where each element of the list represents a leaf node of the tree, and the oct-tree represents the surface of the terrain rather than the enclosed volume. Although a tree may require large amounts of storage for pointers to represent the tree structure, several methods have been devised for 'pointerless' tree structures. By using Morton

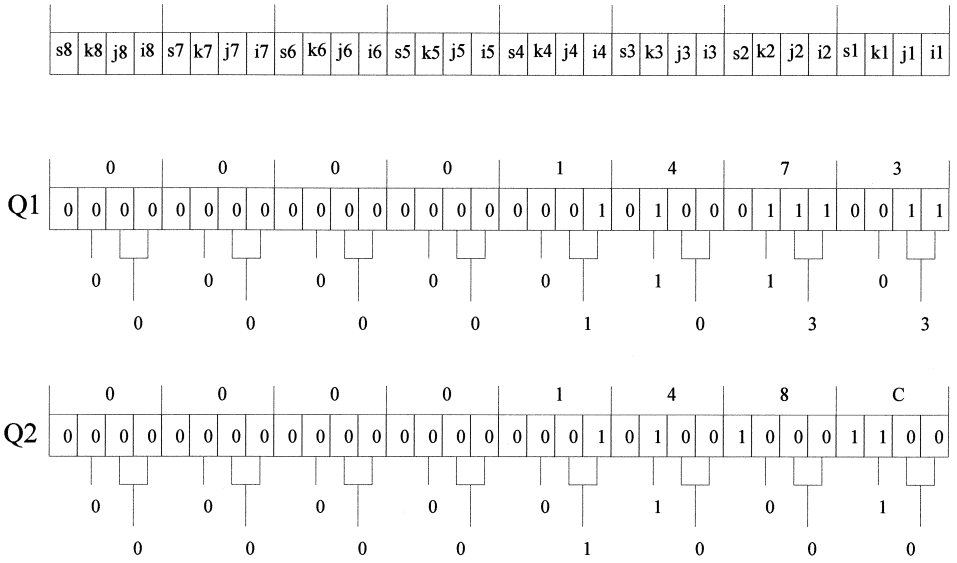

Figure 3. Data format of a terrain oct-tree node.

ordering, each node of the tree can be represented by a single integer, termed a locational code (Allerton and Gia, 1996; Gia, 1994). In Morton ordering, the four quadrants (NE, SE, SW, NW) of a quad-tree node are numbered 0, 1, 2 and 3, respectively. As each quadrant is expanded, an extra digit is added to give a unique 'locational code', which is derived from the expansion of the two bit codes at each level of the quad-tree. Figure 1(c) illustrates Morton ordering for a small region of 64 grid points.

Morton ordering offers several advantages for the manipulation of terrain objects:

(a) The reconstruction of a co-ordinate from a locational code is straightforward,
(b) Mapping between an oct-tree and a quad-tree is achieved by masking individual bits of the locational code,
(c) Topological operations such as neighbour finding, boundary detection and determination of a path segment between two points can be performed by logical operations on the locational codes of oct-tree nodes, rather than physical co-ordinates.

Figure 3 illustrates the encoding of a locational code represented as a 32-bit integer, where the oct-tree node is formed by interleaving the triplet (I,J,K) and the size s of the node.

3. THE EXTRACTION OF OBSTACLES.

For a specific set of flight conditions, it is necessary to extract the set of obstacles, which pose a threat to potential flight paths, in order to determine a safe set of waypoints that avoid these obstacles.

3.1. *Exploring the Obstacles.*

If a terrain is vertically layered, the oct-tree nodes above the flight path altitude pose a collision threat. These 'danger' nodes constitute a quad-tree, representing regions of the DTED containing the set of potential obstacles. However, the list of danger nodes does not give any explicit topographic information. In particular, detection of connectivity between nodes or the location of

https://doi.org/10.1017/S037346330000103X Published online by Cambridge University Press

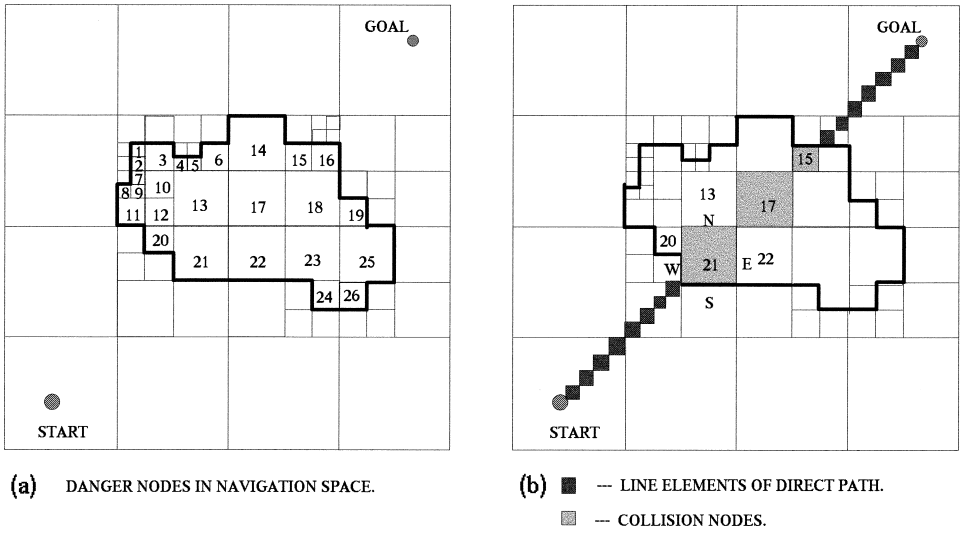

Figure 4. An example of navigation space containing danger nodes.

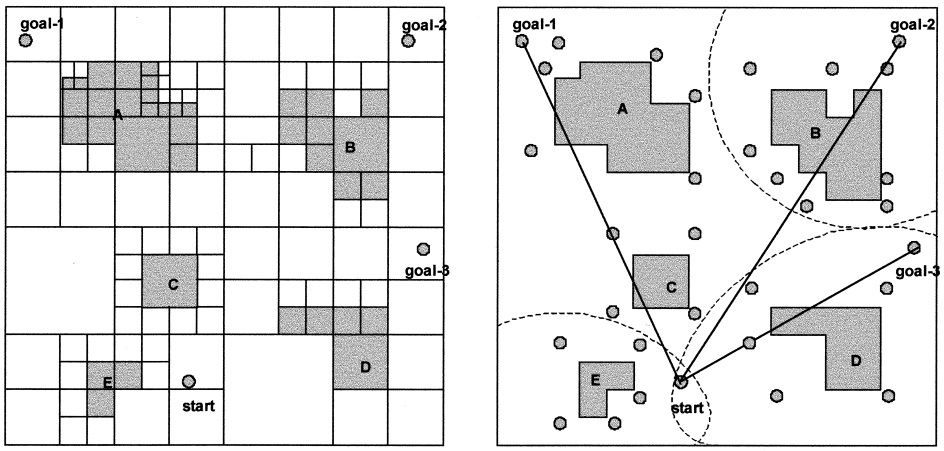

Figure 5. Obstacle regions.

boundary nodes requires additional processing. For example, in Figure 4a the danger area contains 26 nodes, organised as an ordered list, but it is not clear if nodes 1 or 26 belong to the same connected region. Moreover, although a quad-tree may represent a large region of terrain, only a few of these nodes are likely to pose a threat to a specific flight path. For example, in Figure 5, there are five connected danger regions but, for a path to goal-1, the obstacles are limited to regions A and C.

The danger nodes along a potential flight path are termed *obstacle nodes* and are organised as a set of locational codes that represent the coverage of obstacle regions in the navigation space. The extraction of obstacles involves the following steps:

STEP 1: A list of *danger nodes* is extracted from the terrain oct-tree according to an elevation *threshold* value. Each node of the oct-tree is inspected to see if its K value (embedded in the locational code) exceeds the threshold. The resultant list contains all the potential danger nodes in the navigation space.

https://doi.org/10.1017/S037346330000103X Published online by Cambridge University Press

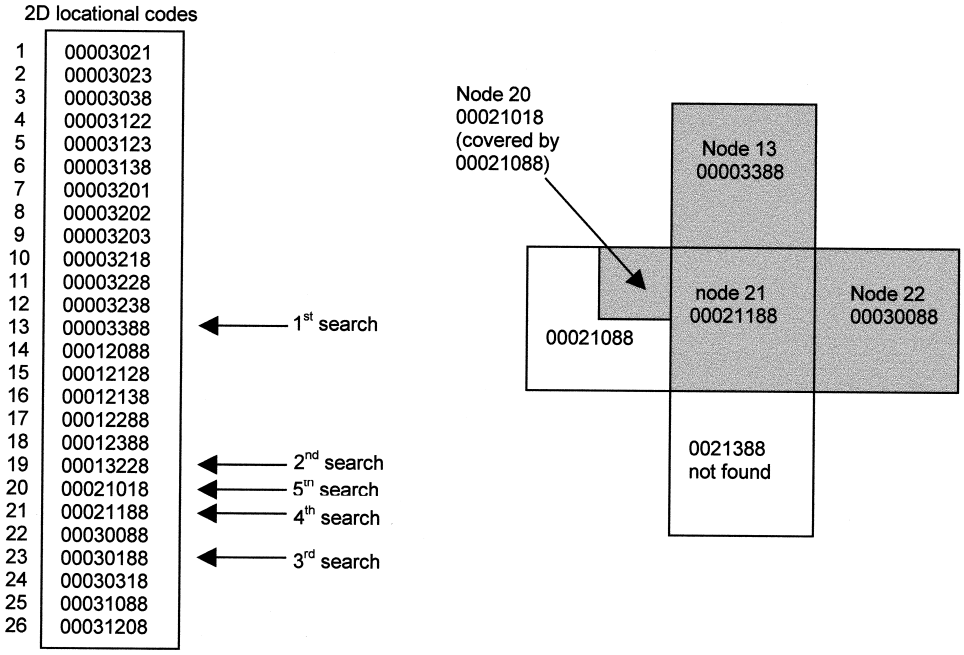

Figure 6. Binary searching to locate an obstacle node.

STEP 2: A 'generate and test' paradigm is used to obtain the *obstacle nodes list* by *collision checking*, to detect intersections between the direct path between a given start point (S) and a goal point (G) for each obstacle. The direct path is obtained by adapting Bresenham's line generation algorithm (Newman and Sproull, 1978) to determine the oct-tree nodes on the line between S and G. Each point along the line is checked against the list of danger leaf nodes. If a point on the line overlaps with any danger node, then the danger leaf node is stored in a list (termed the *SEED* list) for subsequent expansion of the obstacle region. However, an obstacle node may intersect more than one component of a point along a direct path. Figure 4b shows the intersection of a direct path and a set of obstacles nodes, where obstacle node 21 encloses 4 nodes of the flight path, whereas node 15 encloses two nodes.

STEP 3: Each obstacle leaf node in the *SEED* list is used as a 'seed' to 'grow' a connected obstacle region in order to locate the vertices of the region. The obstacle region expansion process locates the obstacle leaf nodes adjacent to the leaf node being expanded. This expansion process generates a set of potential *waypoints* corresponding to the obstacle region in navigation space, forming a flight path that avoids collisions with the obstacles.

3.2. *Obstacle Region Expansion.* Each obstacle leaf node is expanded by computing the locational codes of its neighbours in four directions. This operation is performed recursively until a *boundary node* is reached that does not have any neighbouring nodes in the list of danger nodes. The size of a $2^d \times 2^d$ node is defined as 2^d and the level of the node in the quad-tree is d. The list of danger nodes is searched for a neighbour of equal size. If this search is unsuccessful, it is repeated for a neighbour of a larger size and this process repeats until either a neighbour is found or the uppermost quadrant of the tree is encountered.

https://doi.org/10.1017/S037346330000103X Published online by Cambridge University Press

If a neighbouring node is still not found after this process, it implies that either some smaller neighbouring nodes may exist or the node is a boundary node. If the node is not present in the nodes list, it may be contained in a merged node at an upper level of the quad-tree. The node with the projection code used in the last comparison loop of a binary search procedure either covers, or is covered by, the query node (Gia, 1994). This feature is exploited to determine the existence of any quadrants of the same size as the neighbouring node. If the locational code used in the last comparison loop is covered by the neighbouring node of the node under test, a further search is necessary. Otherwise the current expanded node is a boundary node. For example, the expansion of node 21 in Figure 4 encounters a boundary node in the 'south' direction, connects to obstacle nodes in the north and east directions and requires further processing at the lower level of the quad-tree in the 'west' direction, as shown in Figure 6. In the west direction, obstacle node 20 (00021018) is a NE-quadrant of node (00021088).

The boundary type of an obstacle is encoded using the additive codes 1, 2, 4 and 8 corresponding to the northern, eastern, southern and western boundaries respectively. If none of its sides are on the boundary, a node will have a boundary code of zero. Similarly, a node with a boundary code of 13 is a vertex node. Figure 7 shows an obstacle region and potential waypoints. There are two waypoints in the

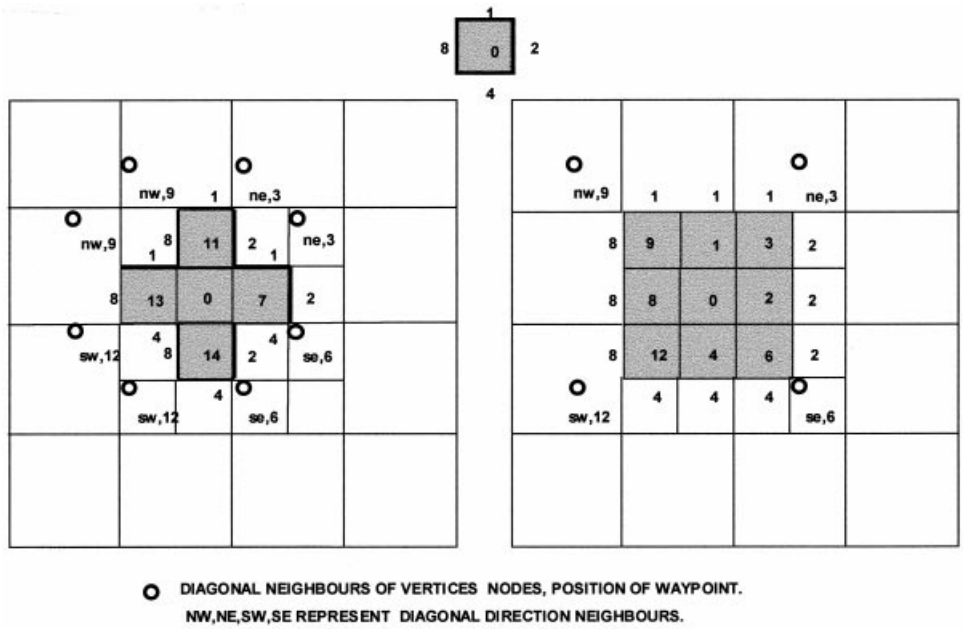

Figure 7. The combination of boundary type and waypoint position.

NW and SW diagonal direction. By convention, a waypoint is represented by the locational code of the north-west corner of its neighbour in the diagonal direction. Before a node can be appended to the list of waypoints, it is necessary to check that it is not a member of the list of danger nodes and that is does not already exist in the list of waypoints.

https://doi.org/10.1017/S037346330000103X Published online by Cambridge University Press

Two further methods are applied to reduce the computational time to extract the list of waypoints. The first approach applies when more detailed information is required. The currently expanded node is sub-divided and each sub-quadrant is expanded individually. In this case, each quadrant is treated as a boundary node and its boundary type is updated. This process is performed recursively until either a neighbouring node is found or the process reaches a boundary node. The second method exploits the hierarchical structure of a terrain oct-tree. In order to reduce the number of vertices generated, the locational code of an equal size neighbouring node is assigned as a boundary node and the expansion process is terminated. This approximation method 'truncates' nodes below the current expanded resolution level.

3.3. *Waypoint Location.* Whenever a neighbouring node is located, it implies that further expansion is needed in that current direction, otherwise the node has encountered a boundary node. After the four main directions are explored, it is straightforward to determine from the boundary type if the node is an obstacle node. This expansion process proceeds recursively from the neighbouring nodes. To avoid examining the same obstacle nodes repeatedly, each node is checked against the list of obstacle nodes. Figure 8a shows the expansion of an obstacle region. As each node is expanded, the *boundary type* of a boundary node is obtained and if it is a vertex node, a *waypoint* is created in the 'diagonal direction' adjacent to the obstacle region, as shown in Figure 8b.

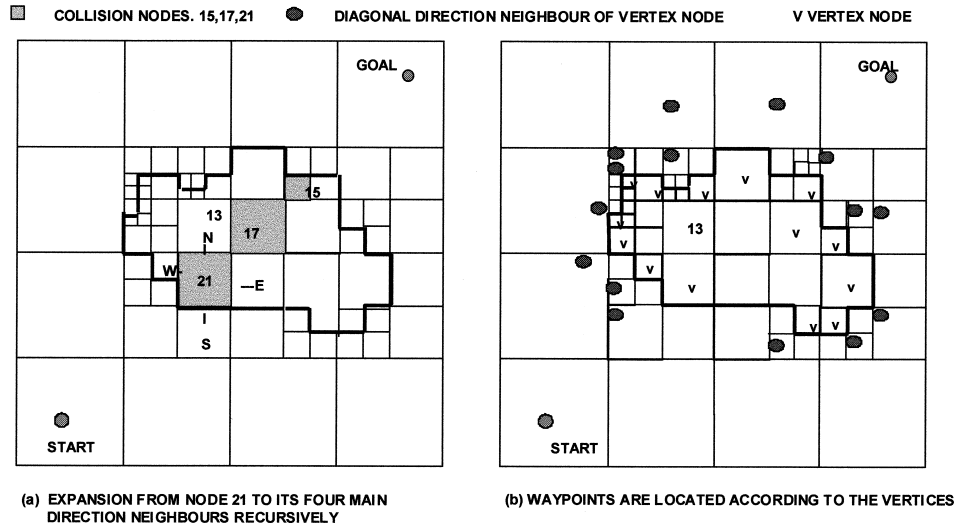

Figure 8. Obstacle region expansion and waypoint derivation.

After the expansion process has been applied to all the members of the *SEED* list, the obstacle regions are obtained along a direct flight path between the start and goal points, according to the overall direction of the flight path. The topographical information of the obstacles is extracted from the specific terrain oct-tree and transformed to a set of waypoints, in order to construct a visibility graph, which is used to determine the optimal path.

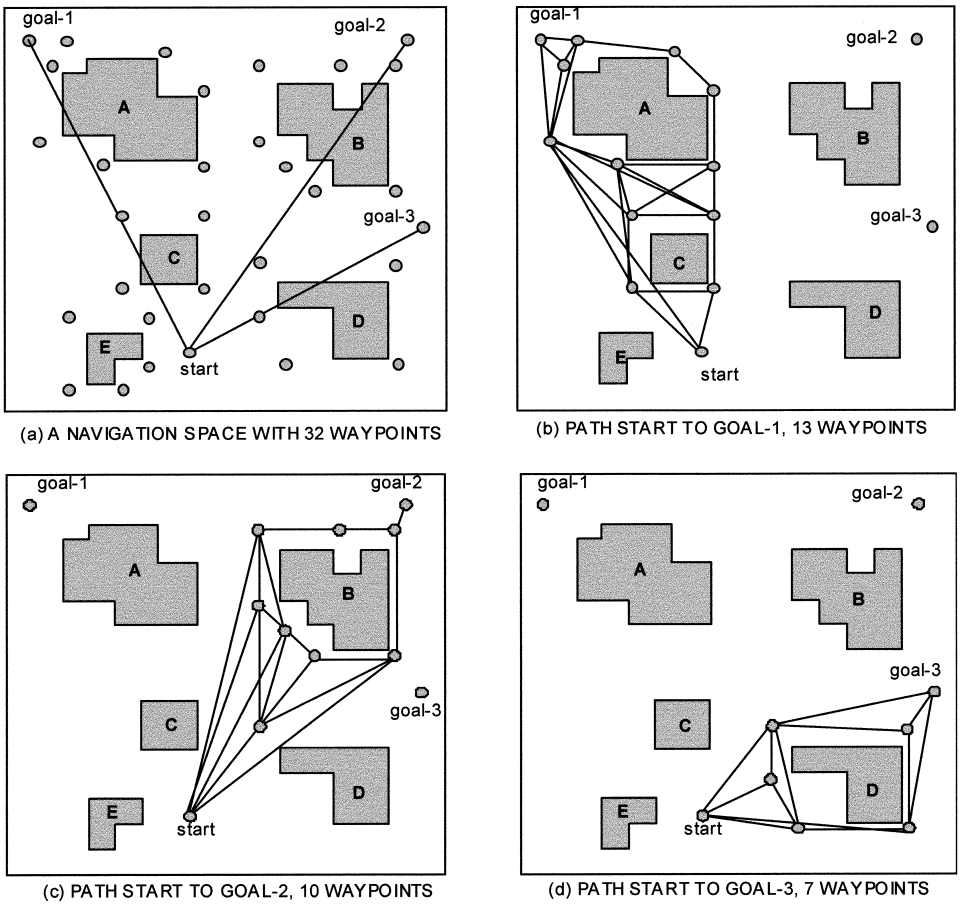

Figure 9. Obstacle regions with different start and goal points.

4. TRANSFORMATION OF THE NAVIGATION SPACE. The visibility graph is constructed from the obstacle vertices generated during the obstacle detection stage. The algorithm consists of examining all pairs of points (W_{from}, W_{to}), where W_{from} and W_{to} are the start, goal, or intermediate waypoints of obstacle regions. To determine whether W_{from} and W_{to} are the end points of a valid flight path segment, the straight line between W_{from} and W_{to} is checked for 'collisions' against all obstacles. The line between W_{from} and W_{to} forms a link of the visibility graph if, and only if, no intersection occurs in the segment joining the two points.

The criteria for collision checking are similar to the test for collisions between start and goal points. The test of a pair of waypoints is terminated as soon as a collision is detected, otherwise the test proceeds until W_{to} is reached. After all the possible combinations of waypoint pairs are tested, the result is a visibility graph where the waypoints are the nodes of a graph and the path segments formed by the waypoints are the arcs of the graph. Figure 9 shows several potential paths that consist of straight lines joining the start point to the goal point via a sequence of waypoints.

The approach used is similar to the methods used in many path-planning applications where a visibility graph of the obstacle space is constructed from a list

https://doi.org/10.1017/S037346330000103X Published online by Cambridge University Press

of polygonal obstacles. The obstacle regions are obtained by extracting the obstacle nodes along a direct path between the start and goal points where the possible obstacles are limited to those near the direct path (or current direction). With this approach, the irrelevant danger nodes can be ignored, reducing the number of potential waypoints. Because the waypoints only represent the subset of obstacles of the danger areas, this process provides a partial visibility graph of the total navigation space. Figure $9_{a–d}$ depicts a set of terrain obstacles, possible waypoints and the resultant partial visibility graphs.

During collision checking, $W^*(W\text{-}1)/2$ waypoint pairs are accessed, where W is the number of waypoints including the start point S and goal point G. The time complexity for construction of a visibility graph is proportional to $W^*(W\text{-}1)/2$. In practice, the number of waypoints is significantly less than the number of vertices.

5. FLIGHT PATH SEARCHING. Once the visibility graph is constructed, it is necessary to search the possible paths in the graph to extract the optimal path between the start point S and the goal point G. Figure 10 shows a visibility graph

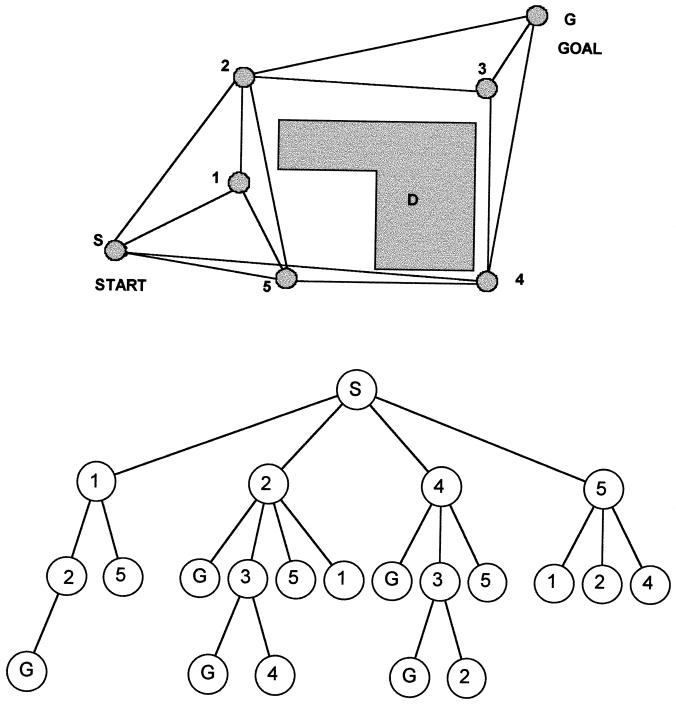

Figure 10. A visibility graph and its tree structure.

representing path segments of a navigation space. The graph is reconstructed as a tree, which comprises the non-cyclic paths from S to the other nodes of the tree, including G.

Several techniques have been developed for searching for a path in a graph, including depth-first, breadth-first and heuristic search methods. An heuristic search

method, constrained to minimise the number of connections, has been incorporated in the flight path-planning algorithm described in this paper. During the search, a path segment is chosen that is as far as possible from the current position, in the expectation that it will be closer to the destination. Cost functions are used to evaluate the effect of adding waypoints.

Finding a truly *optimal path* would require an exhaustive search. However, extraction of a sub-optimal path may be acceptable in real-time airborne applications owing to the potential degradation in performance to extract the path. For a large number of nodes in a visibility graph, the computation load of an exhaustive approach is prohibitive, and a more efficient heuristic search method was adopted, based on a variant of Dijkstra's (1959) algorithm, known as the A* method (Rich, 1986).

6. REAL-TIME APPLICATIONS. In mission management applications, where an aircraft flight plan may need to be modified in real time in order to match flight conditions and changing obstacles, the terrain database must be accessed for every flight plan change in order to re-form the visibility graph. The re-planning of a new flight path must be completed within a few seconds for real-time navigation, where this time constraint includes computation of the flight plan and also the construction of the path searching space.

One technique to ensure the real-time performance of a flight path-planning algorithm is construct a model of the time needed to execute the flight path-planning algorithms, by extensive simulation of flight path-planning algorithms for a specific terrain, affording a trade-off between performance and resolution. These timing measurements include the time to generate the waypoints, the time to construct a visibility graph and the time to find a path at an appropriate level of resolution of the terrain oct-tree. The results from random routeing provide reference computation times to 'tune' the real-time dynamic flight path-planning algorithm for a given terrain. To vary the resolution of the navigation space, a pyramid of quad-trees is used to represent the navigation space and danger nodes (Tanimoto and Pavlidis, 1975), where the k^{th} layer of the pyramid is obtained by approximating the properties of the child nodes at the lower $k+1^{th}$ layer of the pyramid. Table 1 shows the time

Table 1. Experimental results of applying the flight path planning algorithm.

Layer	Nodes	Obstacles	Waypoints	Path segment	T_{wp} (sec)	T_{vg} (sec)	T_{sp} (sec)	Total (sec)	Distance (km)
(a)									
5	6448	110	48	480	0·37	11·39	0·37	12·13	36·8
4	2938	72	22	150	0·22	2·53	0·28	2·02	36·0
3	940	37	11	50	0·11	0·50	0·11	0·72	36·2
2	253	16	7	26	0·06	0·17	0·11	0·33	37·3
1	64	8	6	20	0·06	0·11	0·06	0·22	42·7
(b)									
5	16717	250	65	844	0·77	35·97	1·39	38·13	36·8
4	8659	182	40	348	0·66	11·48	0·22	12·36	36·1
3	3217	78	20	130	0·22	2·36	0·17	2·75	36·2
2	961	37	11	50	0·11	0·55	0·11	0·77	36·5
1	253	16	7	26	0·06	0·11	0·11	0·28	37·2

https://doi.org/10.1017/S037346330000103X Published online by Cambridge University Press

taken for different stages of the path-planning algorithm at different layers of a pyramid. The figures are derived from studies using a 33 MHz 486 PC.
where:

T_{WP} is the time to locate the waypoints (sec.),
T_{VG} is the time to construct the visibility graph (sec.),
T_{SP} is the time to search for a path in the visibility graph (sec.).

In a real-time navigation environment, a new flight path is usually required in a few seconds. In practice, the extraction of waypoints and searching the visibility graph takes a relatively small amount of time in comparison with the overall path planning process. It has been observed that the majority of paths are successfully located at layer three and above of a terrain pyramid and typically, for a visibility graph containing 20 waypoints, a path is found within two seconds.

However, at upper layers of a pyramid, the path may be obscured as a result of the coarseness of the tree nodes and consequently, no path may be found at that layer. In real-time applications, it is preferable to avoid the computation time needed to form a large visibility graph at a detailed resolution layer. The determination of the appropriate processing level can be accomplished by first obtaining the waypoints at a pre-defined level, estimating the size of the visibility graph and then determining if it is necessary to switch to another layer for path planning.

7. RESULTS AND ANALYSIS. The DTED database used in this study is the OSGB 1:50000 Scale height data termed Digital Terrain Model Data (DTM), provided by the UK Ordnance Survey (OS). Each DTM file consists of height values at regular 50 m grid intervals, with values interpolated from the contours of the OS Landranger maps, containing 361201 height values for a 30 km square DTM 'tile'. Two DTM sets were used: one for an area of the Peak District and the other for an area of Port Talbot in Wales. The source file has been restricted to 512×512 grid points to simplify the encoding process (Allerton and Gia, 1996).

Dyer (1982) has shown that the average and worst case numbers of black nodes in a quad-tree representation of a $2^n \times 2^n$ image containing a single $2^m \times 2^m$ region are $O(2^{m+2} - m)$. An alternative interpretation of this result is that the number of nodes is $O(p+n)$ where p is the perimeter (in pixels) of the region. Assuming that the expanded obstacle nodes are single connected, this connected region corresponds to a polygon region of size N_{lp}x N_{lp} embedded in a $2^n \times 2^n$ area. Usually, more than one obstacle may lie along a direct path and the expanded regions are not connected. Each obstacle region also corresponds to a polygon region of size $2^m \times 2^m$ embedded in a $2^n \times 2^n$ area, where $2^m < N_{lp}$. Because the obstacle regions along a direct path are bounded by $N_{lp} \times N_{lp}$ (measured in units of the minimum resolution), the total number of nodes of an unconnected obstacle region is no more than a single connected region and of the order of its perimeter.

The collision check is based on a binary search of the elements (which constitute a path segment) in the danger nodes list. The performance of a binary search is $O(\log N_{dn})$, where N_{dn} is the number of danger nodes (Gia, 1994). The time to check each collision is determined by the number of points along a path segment. The best and worst case figures for no collision to be detected depend on the number of nodes between W_{from} and W_{to}. If the average number of points checked is N_{lp} (measured in resolution units), the cost of a collision check is $O(N_{lp} \times log N_{dn})$ comparisons.

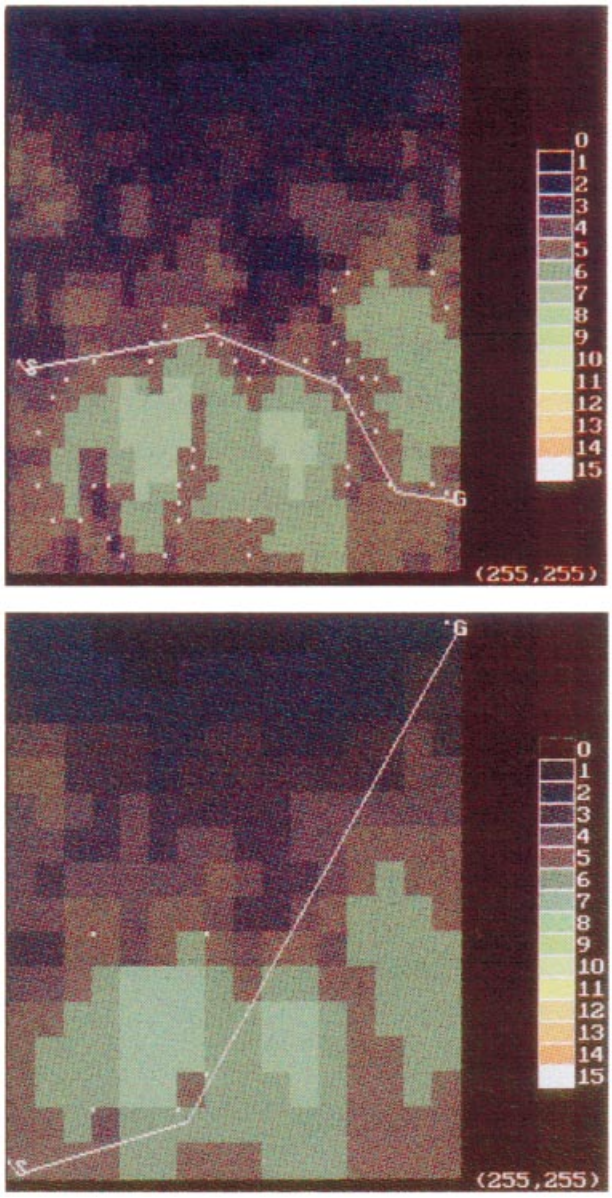

Figure 11. Flight path extraction at different resolutions of the quad-tree.

Construction of the visibility graph occupies most of the time in collision checking. A path segment with 16 nodes at layer k+1 is represented by 8 nodes at layer k. The total number of nodes in layer k may be only one quarter of the number of nodes in layer k+1. Clearly, path planning at a coarser layer can significantly reduce the number of nodes accessed and thus reduce the number of vertices generated in collision check operations. The time complexity for constructing a visibility graph is $O(W^2)$, where W is the number of nodes in the graph. In the general case, the total

https://doi.org/10.1017/S037346330000103X Published online by Cambridge University Press

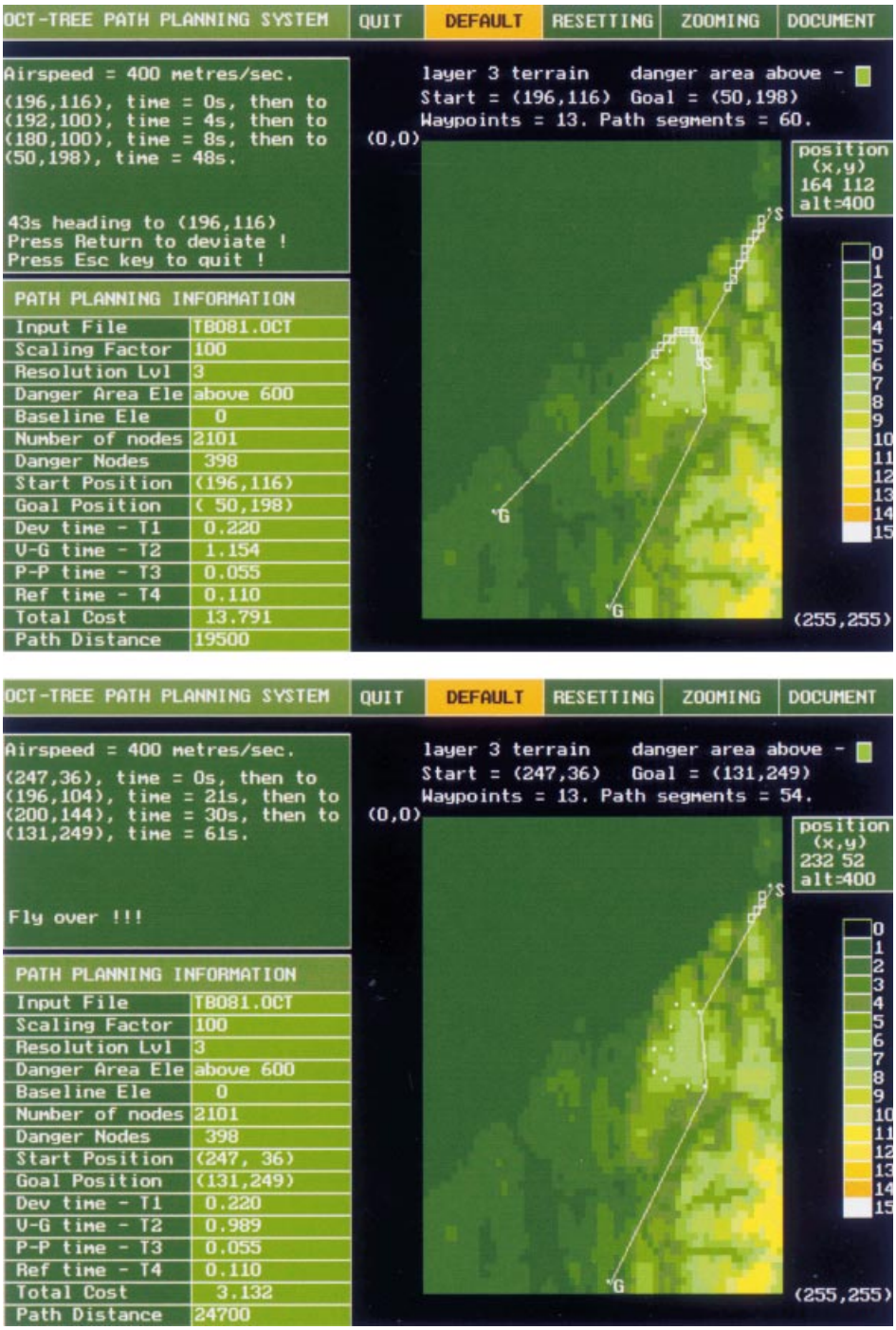

Figure 12. Real-time Routeing.

https://doi.org/10.1017/S037346330000103X Published online by Cambridge University Press

cost of construction of a visibility graph comprising W waypoints, including collision checking, is $O(W^2 \times (N_{lp} \times \log N_{dn}))$.

Generally, the actual operation layer of the path planning process is determined by the minimum flight altitude which in turn determines the number of waypoints. A low flight altitude will generate a large number of danger nodes and obstacle nodes, thus a coarser operational layer may be required to keep the number of waypoints to an acceptable level.

Figure 11 shows the extraction of a flight plan at different resolutions. The flight path for the upper illustration was routed at level 2 of the quad-tree and the waypoints, visibility graph and path extraction were computed in 1·3 s, 5·4 s and 0·2 s respectively. The flight path for the lower illustration was routed at level 1 of the quad-tree and the waypoints, visibility graph and path extraction were computed in 0·1 s, 0·2 s and 0·1 s respectively.

Figure 12 illustrates a real-time simulation where the goal point is changed during a flight mission. The initial start point is (247, 36) and the goal point is (131, 249), the airspeed is maintained at 400 m/sec and the flight altitude and operation layer are preset. During the flight, a new goal point (196, 116) is entered to select a new flight path; the algorithm predicts the new start point as (50, 198) within a predefined time constraint of five seconds.

8. CONCLUSIONS. This paper describes the development of routeing algorithms for real-time mission management. A cell decomposition approach is used to represent a DTED using oct-trees and quad-trees. In addition to the benefits of terrain compression afforded by this method, the use of tree structures facilitates access to the DTED, affording significant advantages in terms of speed.

The tree structures exploit Morton ordering to reduce the storage required for the DTED, avoiding the need for the storage for pointers to represent the tree structure. In addition, the mapping between terrain co-ordinates and tree nodes is straightforward. The obstacle nodes are extracted by locating danger nodes that intersect potential straight line path segments. The tree access operations reduce to logical operations on locational codes to determine intersections and to detect boundary nodes of obstacle regions.

The vertices of the obstacle regions constitute the set of potential waypoints avoiding the obstacles. The resulting visibility graph contains all the possible routes through the terrain that avoid the obstacles and this graph is searched for an optimal path using an adaptation of Dijkstra's A* method.

By exploiting the hierarchical nature of an oct-tree terrain model, the path-planning algorithm can operate at any layer of the terrain oct-tree. Consequently, this approach affords a trade-off between the resolution of route extraction and the time to access the DTED. For real-time applications, it is possible to access the tree at a specific level to ensure that real-time constraints are fulfilled in mission management.

An analysis of the time cost estimates of the algorithms inherent in extracting the obstacles, forming the visibility graph and determining an optimal flight path is supported by the examples presented in the paper. A real-time mission management system has been demonstrated on a standard PC where routes can be extracted from a DTED within a few seconds at an appropriate level of the oct-tree.

ACKNOWLEDGEMENT

The authors would like to thank the Ordnance Survey for providing the DTM files.

REFERENCES

Allerton, D. J. and Gia, M. C. (1996). Oct-tree terrain modelling methods for Terrain Reference Navigation Systems. *The Aeronautical Journal*, Vol. **100**, No. 995, pp. 157–164, May 1996.

Brook, R. A. (1983). Solving the find-path problem by good representation of free space, *IEEE Transactions on Systems*, *Man and Cybernetics*, SMC-13(3), pp. 190–197.

Chan, Y. K. and Foddy, M. (1985). Real time optimal flight path generation by storage of massive data bases. *IEEE National Aerospace and Electronics Conference*, pp. 516–521.

Chen, H. H. and Huang, T. S. (1988). A survey of construction and manipulation of octrees, *Computer Vision*, *Graphics*, *and Image Processing*, Vol. **43**, No. 2, pp. 409–431.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik* 1, pp. 269–271.

Dyer, C. R. (1982). The space efficiency of quad-trees. *Computer Graphics and Image Processing*, Vol. **19**, No. 4, pp. 335–348, August.

Gargantini, I. (1982). An effective way to represent quad-trees. *Communications of the ACM*, Vol. **25**, No. 12, pp. 905–910, December.

Gargantini, I. (1982). Detection of connectivity for regions by linear quad-trees. *Computers and Mathematics with Applications*, Vol. **8**, No. 4, pp. 319–327.

Gargantini, I. (1982). Linear oct-trees for fast processing of three-dimensional objects. *Computer Graphics and Image Processing*, Vol. **20**, No. 4, pp. 365–374, December.

Gia, M. C. (1994). Design of data structures for Terrain Reference Navigation, *PhD Thesis*. Cranfield University, May.

Henley A. J. and Milligan, J. (1992). Applications of terrain and feature database to aircraft operations. *RIN & DGON Digital Mapping and Navigation Conference*, *London*, paper No. 45, November.

Kambhampati, S. and Davis, L. S. (1986). Multi-resolution path planning for mobile robots. *IEEE Journal of Robotics and Automation*, Vol. **RA-2**, No. 2, pp. 135–145, September.

Klinger, A. and Dyer, C. R. (1976). Experiments in picture representation using regular decomposition. *Computer Graphs and Image Processing*, Vol. **5**, No. 1, pp. 68–105, January.

Lauzon, J. P., Mark, D. M., Kikuchi, L. and Guevara, J. A. (1985). Two-dimensional run-encoding for quad-tree representation. *Computer Graphics and Image Processing* Vol. **30**, No. 1, pp. 56–69, April.

Lizza, C. S. and Lizza, G. (1985). Path-Finder: An heuristic approach to aircraft routeing. *IEEE National Aerospace and Electronics Conference*, pp. 1436–1442.

Lozano-Perez, T. and Wesley, M. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, Vol. **22**, No.10, pp. 560–570, October.

Meng, a.c-c. (1987). Flight path planning under uncertainty for robotic air vehicles. *IEEE National Aerospace and Electronics Conference* (*NAECON*), pp. 359–366.

Mitchell, J. S. B. (1988). An algorithmic approach to some problems in terrain navigation. *Artificial Intelligence*, Vol. **37** No. 1–3, pp. 171–201.

Newman, W. M. and Sproull, R. F. (1978). *Principles Of Interactive Computer Graphics*. McGraw-Hill, Inc.

O'Dunlaing, C. andYap, C. K. (1982). A retraction method for planning the motion of a disc. *Journal of Algorithms*, Vol. **6**, pp. 104–111.

Priestley, N. (1990). Ferranti International plc, Terrain Reference Navigation. *IEEE Position*, *Location and Navigation Symposium*, pp. 482–489.

Rich, A. (1986). *Artificial Intelligence*. McGraw-Hill, Inc., 5th Ed.

Samet, H. (1990). *Applications Of Spatial Data Structures*: *Computer Graphics*, *Image Processing*, *and GIS*. Addison-Wesley, Reading, MA.

Schwartz, J. T. and Sharir, M. (1988). A survey of motion planning and related geometric algorithms. *Artificial Intelligence*, Vol. **37**, pp. 157–169.

Tanimoto, S. L. and Pavlidis, T. (1975). A hierarchical data structure for picture processing. *Computer Graphics and Image Processing*, Vol. **4**, No. 2, pp. 104–119, January.