A98-31672

ICAS-98-6,2,3

# ROUTING ALGORITHMS FOR REAL-TIME MISSION MANAGEMENT

D J Allerton and M C Gia

College of Aeronautics
Cranfield University, UK

## Abstract

Digital Terrain Elevation Data (DTED) is widely used in Terrain Reference Navigation (TRN) for terrain avoidance, terrain following and mission planning. This paper describes the use of TRN methods applied to real-time mission management to provide dynamic routing between way-points during a mission.

The DTED is converted to an oct-tree to compress the terrain and to provide fast access algorithms needed in searching functions to locate obstacles in the terrain database. Locational codes are introduced which are based on Morton ordering to provide pointerless tree structures and to simplify the transformations between oct-tree and quad-tree representations of terrain. The terrain is searched to extract the vertices of obstacles and form a visibility graph of possible routes through the terrain. An optimal route is computed according to specific mission constraints.

The paper includes examples of real-time routing running on a PC using a public domain DTED. The use of oct-trees allows a trade-off between the resolution of the route and the extraction time. Several examples are included in the paper to illustrate the capability of the method and the performance of the routing algorithms.

## Introduction

Advances in Terrain Reference Navigation[1] have resulted from three development in avionics:

- the digitisation of terrain elevation data from satellite imagery and modern surveying techniques
- the use of GPS, providing aircraft position to an accuracy better than 10m for military users
- the availability of airborne computer systems combined with avionics sensors, to provide on-board storage of terrain data and sufficient processing performance to access the terrain database and compute safe and efficient routes.

Large regions of the world have been digitised in terms of elevation data, typically at regular intervals of 50m to 100m to a vertical accuracy of 5m to 10m. In terrain following and terrain avoidance, it is possible to compute the flight path to avoid the terrain by extracting the part of the DTED[2] in the immediate flight path of an aircraft. However, for mission management, it may be necessary to search a large part of the DTED to extract an optimal route between two points in the DTED.

Raw DTEDs pose two major problems for aircraft navigation. Firstly, the volume of data may prove prohibitive in terms of the real-time computing requirements to access this data to extract routes. For example, a DTED covering a region 10000 Km by 10000 Km, with grid points spaced at 100m, will contain $10^{10}$ points. Searching and sorting operations involving this number of points is impracticable for real-time navigation. Secondly, there is no topological or structural information when the data is organised as regular grid points.

This paper focuses on two aspects of Terrain Reference Navigation: the organisation of the DTED to facilitate efficient access operations to extract obstacles in the DTED and secondly, algorithms to extract an optimal route through the terrain in real-time.

For navigation, real-time routing implies the ability to produce a series of straight line segments through the terrain passing through a series of way-points within a few seconds. With the availability of digital data links for aircraft, e.g. JTIDS[3], it is possible to update the mission requirements during the mission and for each aircraft to be able to re-route its mission in real-time. The actual optimisation criteria depends on the mission, but may include the flight path distance, exposure to threats or data links, harshness of the dynamics imposed on the aircraft and pilot and minimisation of time-on-target errors.

## Data Structures for DTEDs

In routing an aircraft through a terrain, an aircraft can climb to avoid obstacles or alternatively, it is possible to manoeuvre laterally. Although terrain is defined in three dimensions, once obstacles are extracted which are in the direct path to the target, the search for routes through the terrain becomes two dimensional.

Quad-trees[4],[5],[6] have been used to represent two dimensional regions of obstacles. The region is divided into four equal quadrants. If the characteristics of all the elements in a quadrant are common, then all the essential information is known about that quadrant and there is no need to divide the region any further. However, if the elements of a quadrant are not identical, the quadrant is then sub-divided into four quadrants and the process of subdivision is repeated until a quadrant exhibits this common characteristic or until the resolution of the subdivision matches the resolution of the DTED. The small region shown in Figure 1 can be decomposed in quadrants as shown in Figures 1a and 1b. The quadrants are numbered in a regular format as shown in Figure 1c, resulting in the quad-tree in Figure 2. In this example, the region of 64 possible squares is represented by 28 nodes.

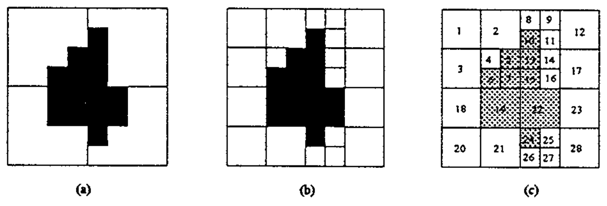


Figure 1 Quad-tree Example

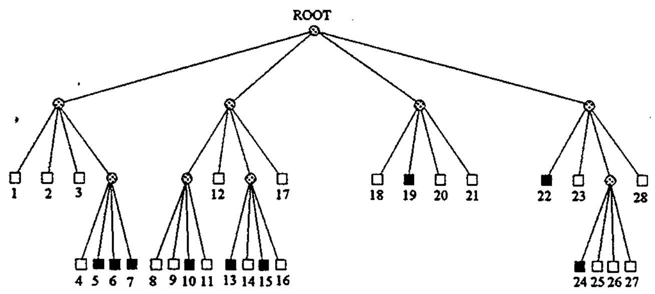


Figure 2 Quad-tree Organisation

This method can be extended to three dimensions using oct-trees[(7),(8)]. The terrain is decomposed into eight sub-octants and this recursive processes of subdivision continues until an octant contains common information. The simple shape in Figure 3a is decomposed into the terrain given in Figure 3b and the corresponding oct-tree given in Figure 4.

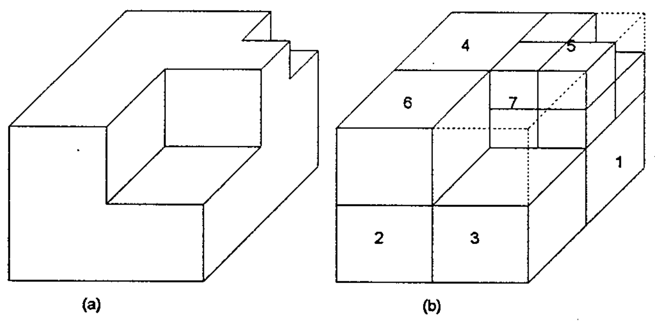


Figure 3 Oct-tree Example

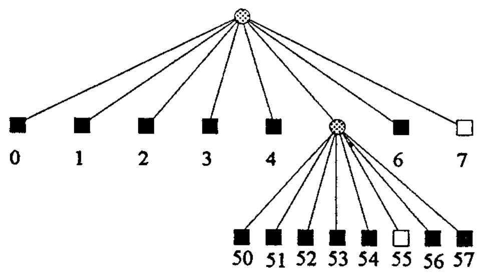


Figure 4 Oct-tree Organisation

There are two advantages with representing terrain in this form. Firstly, it is possible to exploit redundancy in the database. This leads to compression of the DTED; unnecessary topological information is discarded in the oct-tree. Secondly, the levels of the tree correspond to different levels of detail or resolution, with the most detail given by the leaf nodes. In other words, there may be sufficient detail to provide routing information at a higher level of the tree where there are less nodes to process.

## Tree Structures

By organising the terrain as a tree structure, access to the tree nodes is proportional to $\log_2 N$, where N is the number of nodes in the DTED. This is particularly important because the extraction of routes in the terrain involves searching processes. Moreover, it is possible to summarise information about sub-tree nodes within a node of the tree, for example, the highest elevation of the nodes contained in the region defined by the sub-tree.

The disadvantage of tree structures is that each tree node contains information defining both the node and also pointers to the sub-trees, adding to the storage requirements for the tree. However, Allerton and Gia[(9),(10),(11)] have shown that pointerless tree structures can be used to represent terrain. This approach provides a significant level of compression of the terrain and, with a suitable choice of node ordering, can reduce many oct-tree operations to quad-tree operations, in other words, reducing three dimensional tree access operations to two dimensional operations.

## Encoding and Decoding Oct-trees

One method of encoding both oct-trees and quad-trees is the use of Morton ordering[(12)]. This is illustrated in Figure 5 for three small quad-trees. Each subdivision of the tree introduces an extra digit of Morton coding. For example, the shaded cell in the oct-tree in Figure 5(c) is given by the locational code 213. This code represents a value where each digit represents a weighted power of 4. In this case, the location code at co-ordinate (3, 5) is given by $2 \times 4^2 + 1 \times 4^1 + 3 \times 4^0 = 39_{10}$.

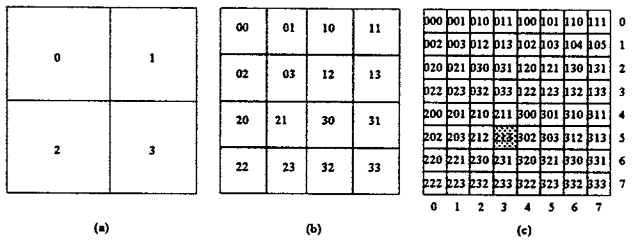


Figure 5 Quad-tree Locational Codes

In the general case, the three-dimensional co-ordinate (I, J, K), of an oct-tree of size $2^n \times 2^n \times 2^n$ is given by:

$$I = c_{n-1}2^{n-1} + c_{n-2}2^{n-2} + \dots\dots\dots + c_0 2^0$$
$$J = d_{n-1}2^{n-1} + d_{n-2}2^{n-2} + \dots\dots\dots + d_0 2^0$$
$$K = e_{n-1}2^{n-1} + e_{n-2}2^{n-2} + \dots\dots\dots + e_0 2^0$$

where the locational code Q is given by:

$$Q = q_{n-1}8^{n-1} + q_{n-2}8^{n-2} + \dots\dots\dots + q_0 8^0$$

and the coefficients $q_p$ are given by:

$$q_p = e_p 2^p + d_p 2^p + c_p 2^p \quad \{p = n-1, n-2, n-3, \dots\dots, 3, 2, 1, 0\}$$

The primary advantage of using Morton ordering is that the vertical downward projection of any oct-tree node has the same locational code as an equivalent quad-tree. Moreover, the derivation of these projection codes reduces to simple logical operations. The encoding of a terrain array into a terrain oct-tree involves the mapping of an array co-ordinate (I, J) to the corresponding locational code and includes the encoding of scaled elevation data in the corresponding locational code.

In the standard oct-tree method, a voxel is defined as a unit length cube in the I, J, K directions. For the terrain oct-tree model, the length of a cube in the K axis direction depends on the scaling factor of the elevation. In effect, the introduction of a scaling factor divides the terrain into horizontal bands. A fourth parameter S is added to represent the size of the leaf node.

A single 3-D locational code can represent an area of terrain surface by its three dimensional co-ordinates and size information. Unlike Gargantini's method, an octant is encoded with a locational code which is one of the set of digits {0, 1, 2, 3, 4, 5, 6, 7}, which can be represented by three bits. If the eight voxels belong to the same octant, they are grouped together by setting the fourth bit of the locational code. The size of a node is obtained by detecting the occurrence of these flag bits as the tree nodes are accessed.

For standard oct-trees, only odd values occur along the K axis in grouping sub-octants. In other words, the upper layer of an octant has underlying sub-octants {4, 5, 6, 7}. For terrain oct-trees, by exploiting the property that the projection of a terrain surface generates a complete quad-tree, the grouping process can be performed in quadrants instead of octants. This allows terrain surface voxels to be merged if the IJ plane projection of four blocks, with equal K values, occurs in the same quadrant of the corresponding quad-tree.

Using this method to represent an oct-tree, a block of terrain surface with equal K values is uniquely defined by the size of the block and the co-ordinates of its NW corner. Assuming a DTED given by $2^n \times 2^n$ grid points, the locational code of each leaf node of side length $2^k$ consists of n digits where the k trailing digits contain a flag bit, and the leading n-k digits contain the locational codes defining the path from the root of the tree. Note that the merging of leaf nodes is based on the quadrants of the projection plane and therefore, the locational code along the path is represented by the digits 0, 1, 2, and 3. The third and fourth bits of each digit are reserved for the scaled value K and the size value S respectively. After interleaving the indices I, J, K and S, the locational code is given by a value formed from the set of hexadecimal digits {0, 1, 2, 3, 4, 5, 6, 7, 8, C}.

The oct-tree is organised as a linear list where each element represents a leaf node of the tree. Each node is defined by a single integer known as a locational code, as shown in Figure 6. Using 32 bit locational codes, the co-ordinate of the node, the size of the node and its elevation are encoded in this single word.

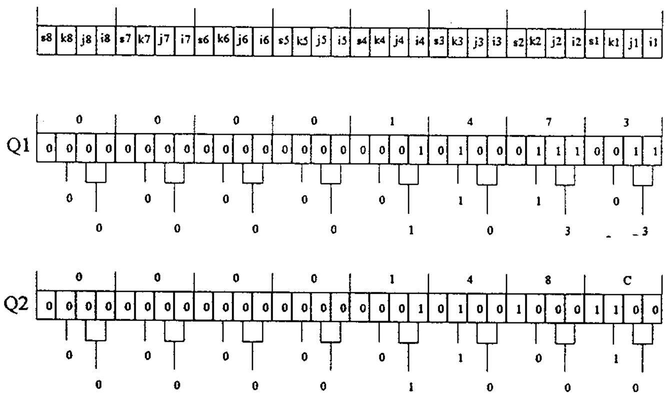


Figure 6 Locational Code Organisation

A further advantage of Morton ordering of the nodes is that terrain can be represented by an oct-tree but the two dimensional quad-tree can be extracted from the oct-tree by means of straightforward projections which reduce to logical operations on locational codes. Moreover, extraction of the DTED co-ordinates is fast and simple. In many applications, operations to be performed on a tree may be derived from DTED co-ordinates, for example, in determining a straight line to detect obstacles in the direct path. Similarly, it is necessary to provide transformations between DTED co-ordinates and a node address. Operations on the DTED where it is necessary to swap between DTED space and tree-space include neighbour finding (locating the nearest obstacle), straight-line path formulation in terms of tree nodes and distance measurements between two tree nodes.

## Flight Path Planning

The flight path planning involves two phases(13),(14). Firstly, a search space is generated containing all the possible paths between a start point and a goal point which avoid the obstacles. It is likely that this search space will contain a considerable number of paths. The second phase is to locate a path which satisfies specific mission constraints. It is assumed that the obstacle space is static(15),(16),(17),18).

The extraction of an optimal route can be based on the minimisation of an objective function defining the 'cost' of the path. Although this information can be embedded in the DTED by pre-processing(19),(20) in order to reduce the real-time searching, the obstacle space is likely to alter with changing threats and minimum clearance height above the terrain. Therefore, the methods described in this paper are based on real-time path extraction, constrained by mission requirements and consequently, no pre-processing of the DTED is assumed to assist the mission routing system.

The set of obstacle nodes is extracted from the oct-tree based on constraints of threats, distance, terrain clearance or other mission requirements. This forms a quad-tree of obstacle nodes, based on altitude(21), as shown in Figure 7. A straight line is generated, using a modified Bresenham line generation method(22) based on tree nodes, as shown in Figure 8. The tree nodes intersecting this straight line are extracted between the start point and goal point. Each node of this line is checked against the list of obstacles and when an intersecting node is encountered, it is added to the list of intersecting nodes. It is possible that more than one component of a point along the direct will intersect the same obstacle node. For example, in Figure 8, obstacle node 21 encloses four point elements, whereas node 15 encloses two point elements.

The obstacles list is a sub-set of the quad-tree representing the navigation space and may contain several isolated regions. However, the list of obstacle nodes does not provide any useful topological information, particularly connectivity or boundary conditions. For example, in Figure 7, the obstacle quad-tree contains 26 nodes but it is not clear if node 1 or node 26 belong to the same connected region.

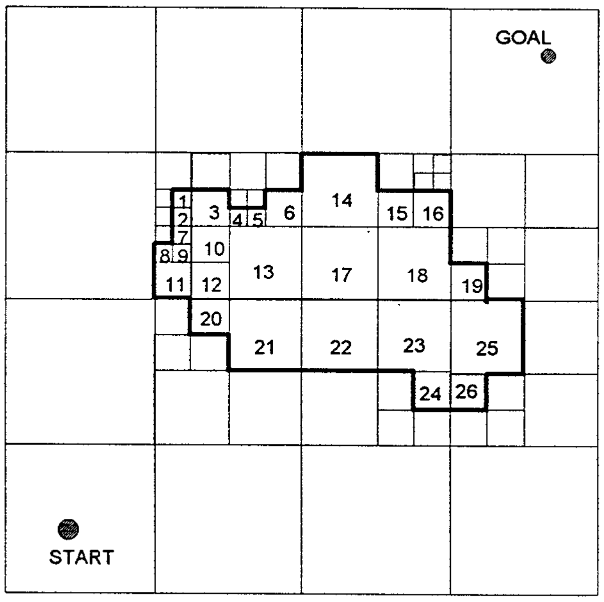


Figure 7 Obstacle Nodes

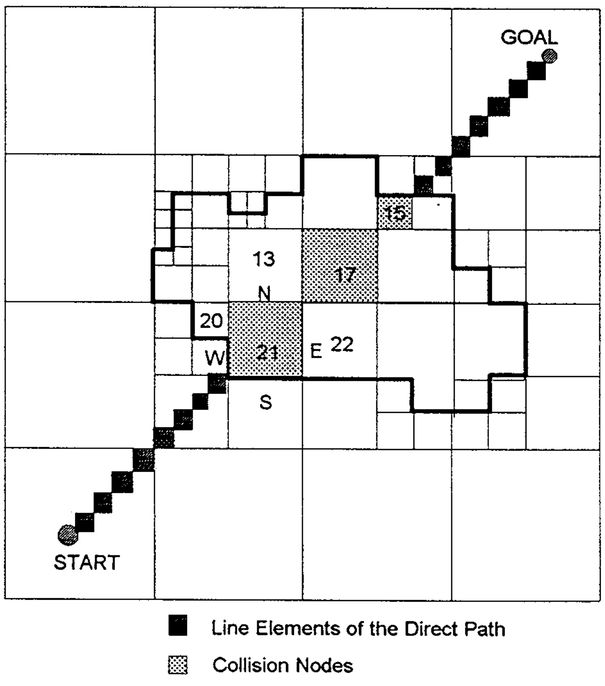


Figure 8 Collision Detection

For each node in the list of intersecting nodes, the node is expanded to detect the boundaries of the intersecting obstacle. The obstacle leaf node is formed by computing the locational codes of the neighbouring nodes in four directions. This is a recursive operation which is completed when a boundary node is reached which does not have any neighbouring nodes in the list of obstacle nodes. The size of the neighbouring node may be different from the size of the expanded node and in this case, the search process is repeated until a neighbouring node is located which is of equal size.

If this searching process does not locate a neighbouring node, then either a smaller neighbouring node may exist or the node is a boundary node. The required node may be contained in a merged node at an upper level of the oct-tree. However, the extraction of a covering node is straightforward from the logical structure of the locational codes. This is shown in Figure 8, where the expansion of node 21 encounters a boundary node in the south direction, connects to nodes in the north and east directions and detects node 20 from further inspection of nodes in the west direction.

Whenever a neighbouring node is located, it implies that further expansion is needed in that direction, otherwise the node is a boundary node. After all four main directions are explored, the boundary type of an obstacle node is obtained.

As each neighbouring node is encountered, it is expanded until a boundary node is detected. However, as this recursive search is also applied to neighbouring nodes in all directions, it is necessary to avoid re-visiting nodes which are established as boundary nodes. Figure 9 shows the expansion of the obstacles nodes from node 21 to locate the boundary nodes.

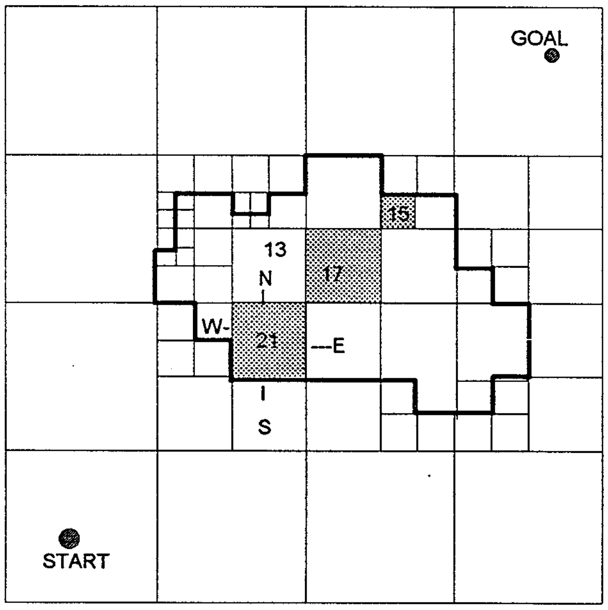


Figure 9 Obstacle Region Expansion

Figure 10 shows the resultant way-points for this obstacle, where this process has been applied to nodes 21, 17 and 15, the obstacle nodes between the start point and the goal point. The vertices are obtained from the north west corner of the boundary nodes in the quad-tree.

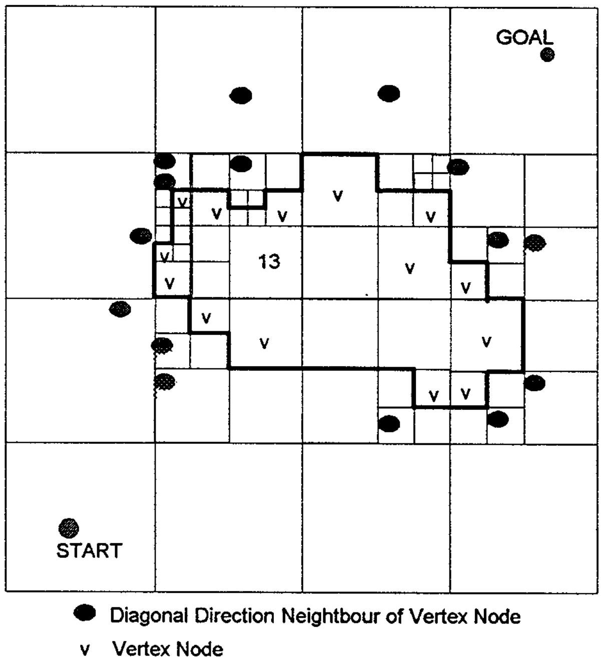


Figure 10 Way-point Extraction

## The Visibility Graph

After expansion of all the obstacle nodes, the complete list of possible way-points, known as the visibility graph, is formed as shown in Figure 11.

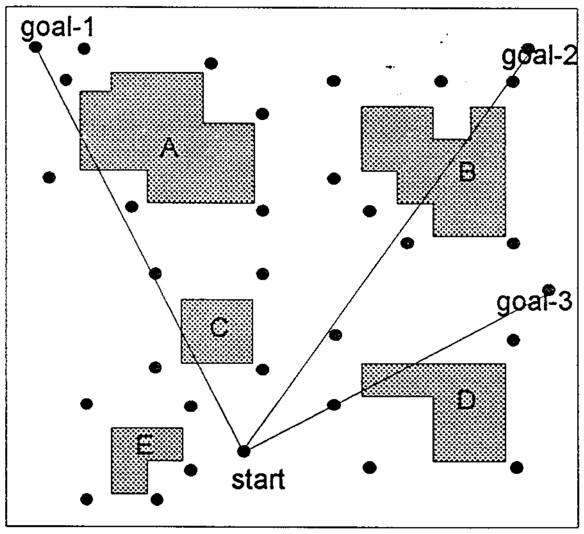


Figure 11 Navigation Space Visibility Graph

Clearly, not all the obstacles are likely to be encountered in the route, as shown in Figure 11, where the route from the start point to goal-1 is independent of obstacles B and D.

If a connected graph is found between the start point and the goal point, other way-points can be eliminated which are not in the direction towards the goal point, as shown in Figure 12, where a visibility graph of 14 way-points is extracted from the navigation space containing 32 way-points.

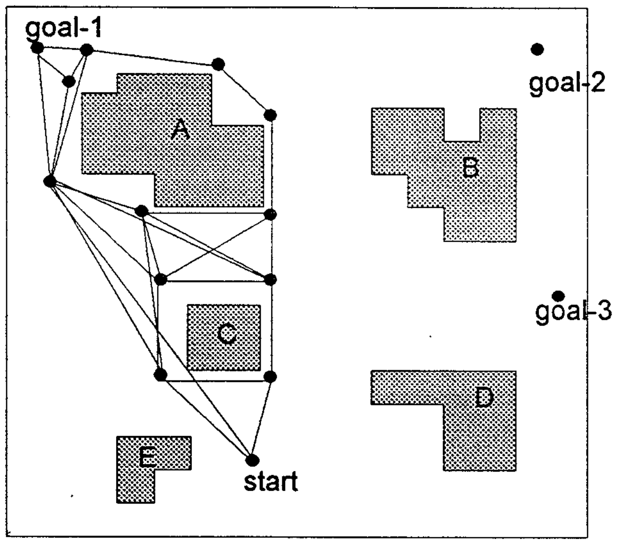


Figure 12 Partial Visibility Graph

This method is similar to the methods used in most path planning problems, where a visibility graph is constructed from a list of polygonal obstacles. The difference for flight path planning is that the possible obstacles are limited to those close to the direct path. During the collision check, w*(w-1)/2 pairs of points are examined where w is the number of points. The time complexity is therefore proportional to $w^2$.

To search the complete visibility graph for the optimal path would require exhaustive searching. For a large number of points in the visibility graph, this is clearly impractical and a heuristic version is used, based on a variant of Dijkstra's algorithm(23), known as the A* method(24). Figure 13 shows the visibility graph, with the corresponding search tree in Figure 14, which is generated to extract the optimal path. The graph of 7 nodes contains five possible paths from the start point to the goal point.

As the flight planning phase must be performed in real-time, it is necessary determine if a new flight path exists within a few seconds which requires the extraction of the obstacles, derivation of the visibility graph and computation of the set of mission way-points.

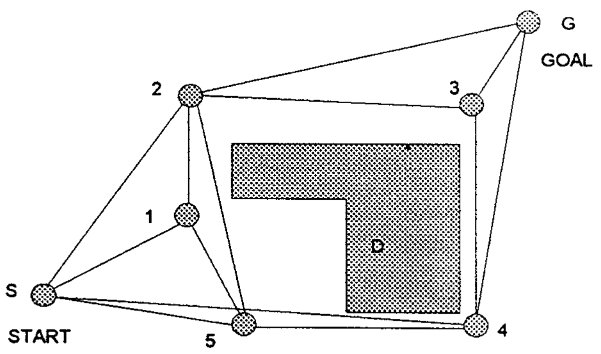


Figure 13 Visibility Graph

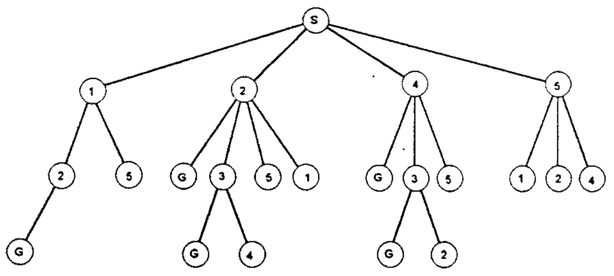


Figure 14 Search Tree for a Visibility Graph

In addition to the advantages of terrain compression afforded by oct-trees, the hierarchy of the tree representation allows the tree to be accessed at different levels of detail. In other words, the flight path extraction method described in this paper can be executed at any level of the tree. Clearly, there are less nodes towards the root of the tree and the algorithm will therefore execute faster but consequently, will produce a coarser route. Indeed, routes through the terrain which are evident at the resolution of the leaf nodes, may be obscured at higher levels of the tree.

Despite this limitation, the oct-tree method offers one particular advantage in real-time mission management which is the ability to trade-off time and resolution. By applying the algorithm to a specific terrain for randomly selected routes, the routing times were measured for tracks of different lengths at all levels of resolution. These results showed that it is possible to predict the performance of the routing algorithm with an acceptable degree of certainty and then apply this knowledge to the routing algorithm.

The real-time method predicts the time to route and selects a tree resolution to be able to route the mission within a few seconds. The time to the first way-point is then known and the remainder of the mission (from the second way-point) is then re-routed at a higher level of resolution as the mission proceeds.

## Results

Table 1 shows the performance of the routing algorithms for a 33 MHz 486 PC using two terrain databases at different levels of tree resolution from level 1 (the coarsest) to level 5 (the most detailed). The DTED files are based on two regions provided by the UK Ordnance Survey using their 1:50000 Digital Terrain Model (DTM) data, containing 36201 height values for a 30 Km square 'tile'. The source file is reduced to 512:512 grid points to simplify the oct-tree encoding process. The original DTED contains over 256000 grid points at 100m spacing. Each oct-tree is organised on 5 levels.

The DTEDs reduce the number of DTED points to 6448 and 16717 nodes respectively at 100m resolution. At Level 1, the DTEDs reduce to 64 and 253 nodes at the coarsest level. The number of obstacles reduces with the level of the oct-tree by over a factor of 10 from 110 and 250 obstacles. The number of way-points varies from 20 to 65 and the possible number of path segments is less than 1000 for these DTEDs reducing to 50 and 130 respectively at level 3 of the tree. The overall mission routing times are shown in Figures 15 and 16 for the respective DTEDs as a function of the number of quad-tree nodes extracted from the DTED.

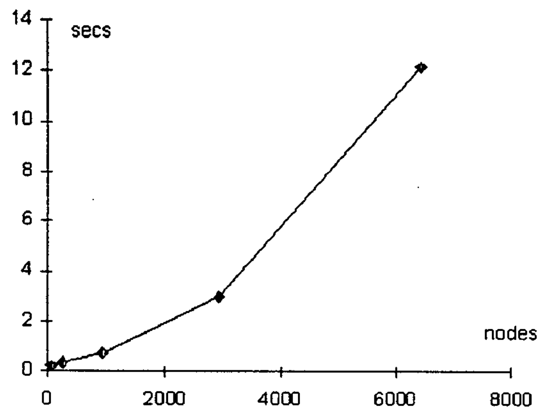


Figure 15 Mission Planning Time DTED-A

In addition, a series of random routing tests were applied to the mission planning algorithms, in order to validate the method and also to assess the algorithm performance. Table 2 shows the average time to extract the way-points, to construct the visibility graph and to extract the resultant path from the visibility graph. Clearly the performance depends on the number of potential way-points and Table 2 shows that a 33 MHz 486 PC is capable of generating a new route within 5 seconds, provided the resultant obstacle space contains less than 30 way-points.

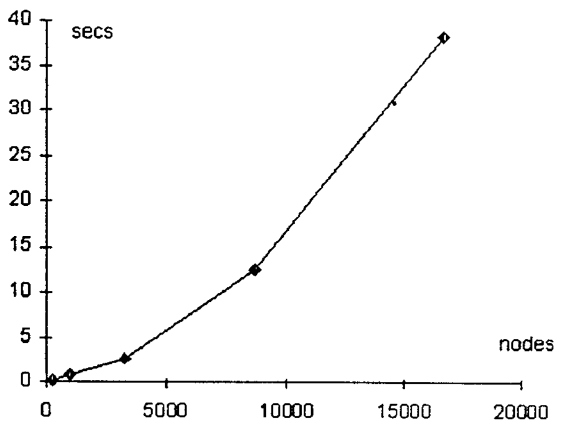


Figure 16 Mission Planning Time DTED-B

The major time is taken to construct the visibility graph. For both DTEDs this is less than 3 seconds at level 3 or below but reaches 36 seconds for one of the DTEDs at the leaf node resolution. The time to locate the way-points and to extract the path from the visibility graph is insignificant in comparison with the time to construct the visibility graph. However, these figures illustrate the concept of initial routing at a coarse level followed by subsequent routing of the remainder of the mission at a finer level of detail.

The collision check is based on a binary search of the elements and the performance is therefore $O(\log_2 N)$ where N is the number of collision nodes. The time to check each collision is determined by the number of points along the path segment. If the average number of points checked is N, the cost of a collision check is $O(N \times \log_2 N)$ comparisons.

Most of the time in collision checking is spent in constructing the visibility graph. The complexity of the visibility graph is $O(w^2)$ where w is the number of nodes in the graph. In the general case, the total cost of constructing a visibility graph of w way-points is $O(w^2 \times (N \times \log_2 N))$.

Dyer[25] has shown that a region $2^m \times 2^m$ of a quad-tree represented by $2^n \times 2^n$ grid points generates approximately $O(2^{m+2} - m)$ nodes. Alternatively, the number of nodes is $O(p + n)$ where p is the perimeter of the region in points. Assuming the expanded obstacle nodes are single connected and the obstacle region corresponds to an area N x N, then for an obstacle region $2^m \times 2^m$, $2^m < N$ and the number of nodes accessed to form the obstacle region is of the order of its perimeter. The cost of the expansion function is $O(4^{d+1} \times (p + n))$, where d is the depth of the quad-tree.

Figures 17, 18 and 19 were captured from a real-time implementation of the mission management system running on a 33 MHz PC. Figure 17 is routed at layer 3 for a quad-tree of 2101 nodes. The routing algorithm detects a path to the right of the hill shown by the vertices of the quad-tree. Figure 18 shows the same initial route with a new route computed during the mission. Figure 19 shows the same terrain and start and goal points at layer 2 where the quad-tree contains 628 nodes. Note that the number of way-points increases from 13 to 22, producing a different route.

## Conclusions

A real-time mission planning system has been developed for a PC using a proprietary DTED. The DTED is converted to an oct-tree and during the mission, a quad-tree of obstacles is extracted from the oct-tree. Routing around the obstacles is achieved by detecting collisions between the current aircraft position and the goal point. The resultant visibility graph is explored to locate the set of way-points to satisfy minimum mission criteria, such as overall distance or exposure to a threat.

The use of oct-trees allows the DTED to be organised as a hierarchical data structure with the terrain data stored at different levels of resolution. Oct-trees afford a reasonable level of compression in comparison with raw terrain data and by the use of Morton ordering, the oct-tree is stored in an efficient format which avoids the use of pointers. Operations which are common to TRN applications, such as the detection of obstacles, location of a neighbouring node, detection of obstacle boundaries and routing algorithms reduce to tree traversal algorithms which are proportional to $\log_2 N$ where N is the number of nodes in the oct-tree.

The routing is actually performed on quad-trees which are extracted from the oct-tree to produce an immediate obstacle space according to clearance criteria. It is straightforward to transform between terrain co-ordinates and locational codes, and vice-versa and operations to locate and merge items in the DTED reduce to simple logical operations on locational codes. The path planning method reduces the size of the search space by establishing a partial visibility graph, omitting regions of terrain which are independent of the possible set of way-points.

The oct-tree resolution can be varied during the formation of the tree and by limiting the access to the higher nodes of the tree, allows a trade-off between resolution of the routing algorithm and the time to route. In practice, routes were extracted within five seconds using a 33 MHz PC at the lowest level of the tree for a 50 Km square region. Simulation studies of random routing to exercise the mission management algorithms show that it is possible to predict the time to route a mission as a function of path distance and the level of the oct-tree, suggesting that this information can aid the misston management algorithms in the selection of the most appropriate level of the oct-tree to perform the routing.

The methods outlined in this paper provides a low-level set of DTED operations, including database conversion, oct-tree generation, quad-tree extraction, tree access operations, transformation between tree co-ordinates and terrain co-ordinates, neighbour finding, visibility graph extraction and optimal route extraction. Subsequent optimisation can be applied at this final stage to tailor the mission management software to mission specific requirements.

## Acknowledgement

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

## References

1. Priestley N, Terrain Reference Navigation, *IEEE Position, Location and Navigation Symposium* 1990, pp. 482-489.
2. Henley A J and Milligan J, Applications of Terrain and Feature Database to Aircraft Operations, *RIN & DGON Digital Mapping and Navigation Conference, London*, Nov 1992.
3. Fried W R, Principles and Simulation of JTIDS Relative Navigation, *IEEE Trans. Aerospace and Systems*, AES-14(1), pp. 76-84, 1978.
4. Gargantini I, An Effective Way to Represent Quadtrees, *Communications of the ACM*, Vol. 25, No. 12, Dec 1982, pp. 905-910.
5. Gargantini I, Detection of Connectivity for Regions by Linear Quadtrees, *Computers and Mathematics with Applications*, Vol 8, No. 4, 1982, pp. 319-327.
6. Klinger A and Dyer C R, Experiments in Picture Representation Using Regular Decomposition, *Computer Graphs and Image Processing*, Vol. 5, No. 1, Jan 1976, pp. 68-105.
7. Chen H H and Huang T S, A Survey of Construction and Manipulation of Octrees, *Computer Vision, Graphics, and Image Processing*, Vol. 43, No. 3, 1988, pp. 409-431.
8. Gargantini I, Linear Octtrees for Fast Processing of Three-dimensional objects, *Computer Graphics and Image Processing*, Vol. 20, No. 4, Dec 1982, pp. 365-374.
9. Allerton D J and Gia M C, The Application of Oct-trees in Airborne Terrain Guidance Systems,

*RIN & DGON Digital Mapping and Navigation Conference, London,* Nov 1992.
10. Gia M C, Design of Data Structures for Terrain Reference Navigation, PhD Thesis, College of Aeronautics, Cranfield University, May 1994.
11. Allerton D J and Gia M C, Oct-tree Terrain Modelling Methods for Terrain Reference Navigation Systems, *The Aeronautical Journal*, pp. 157-164, May 1996.
12. Morton G M, A Computer Orientated Geodetic Database and a New Technique, Ottowa, Canada, 1966.
13. Brook R A, Solving the Find-Path Problem by Good Representation of Free Space, *IEEE Transactions on Systems, Man and Cybernetics, SMC*-13(3), 1983, pp. 190-197.
14. Mitchell J S B, An Algorithmic Approach to Some Problems in Terrain Navigation, *Artificial Intelligence, Vol 37 No 1-3* 1988, pp. 171-201.
15. Kambhampati S and Davis L S, Multiresolution Path Planning for Mobile Robots, *IEEE Journal of Robotics and Automation, vol. RA-2, No. 3.,* Sep 1986, pp. 135-145.
16. Lozano-Perez T and Wesley M, An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles, *Communications of the ACM* 22,10., Oct 1979, pp. 560-570.
17. O'Dunlaing C and Yap C K, A Retraction Method for Planning the Motion of a Disc, *Journal of Algorithm,* 6, 1982, pp. 104-111.
18. Schwartz J T and Sharir M, A Survey of Motion Planning and Related Geometric Algorithms, *Artificial Intelligence* 37, 1988, pp. 157-169.
19. Chan Y K and Foddy M, Real Time Optimal Flight Path Generation by Storage of Massive Data Bases, *IEEE National Aerospace and Electronics Conference* 1985, pp. 516-521.
20. Lizza C S and Lizza G, Path-Finder: An Hueriistic Approach to Aircraft Routing, *IEEE National Aerospace and Electronics Conference* 1985, pp. 1436-1443.
21. Meng A C, Flight Path Planning Under Uncertainty for Robotic Air Vehicles, *IEEE National Aerospace and Electronics Conference (NAECON)* 1987, pp. 359-366.
22. Newman W M and Sproull R F, Principles of Interactive Computer Graphics, McGraw-Hill, Inc., 1978.
23. Dijkstra E W, A Note on Two Problems in Connexion with Graphs, *Numerische Mathematik* 1, 1959, pp. 269-271.
24. Rich A, Artificial Intelligence, McGraw-Hill, Inc., 5th ed, 1986.
25. Dyer C R, The Space Efficiency of Quadtrees, *Computer Graphics and Image Processing* 19, 4., August 1982, pp. 335-348.

| Layer | Nodes | Obstacles | Way-points | Segments | $T_{wp}$ sec | $T_{vg}$ sec | $T_{sp}$ sec | Total sec | Distance km |
|---|---|---|---|---|---|---|---|---|---|
| 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 | 3.02 | 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 |

(a) DTED A

| Layer | Nodes | Obstacles | Way-points | Segments | $T_{wp}$ sec | $T_{vg}$ sec | $T_{sp}$ sec | Total sec | Distance km |
|---|---|---|---|---|---|---|---|---|---|
| 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 |

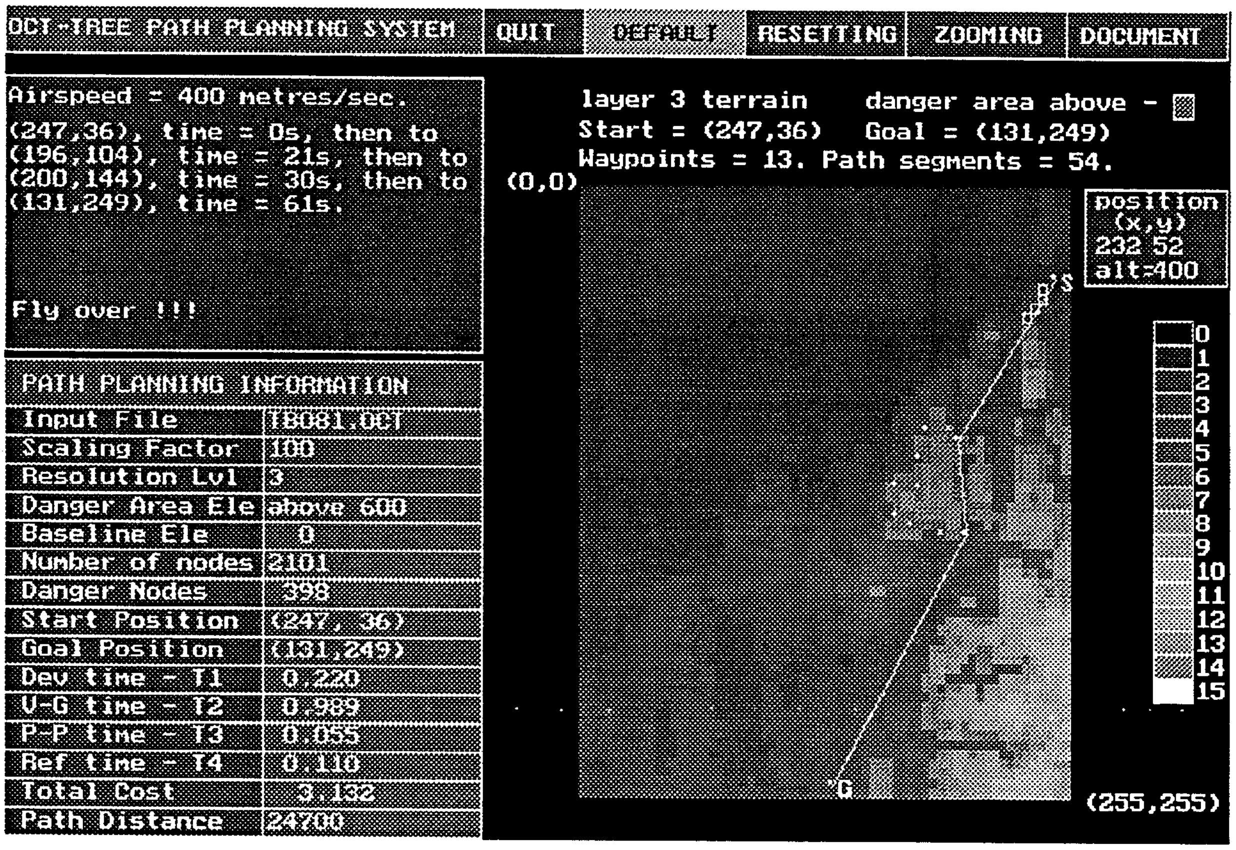
(b) DTED B

Table 1 Experimental results of applying the flight path planning algorithm

| Way-points | $T_{wp}$ sec | $T_{vg}$ sec | $T_{sp}$ sec | Total sec |
|---|---|---|---|---|
| 1- 10 | 0.06 | 0.33 | 0.60 | 0.99 |
| 11-20 | 0.17 | 1.98 | 0.71 | 2.86 |
| 21-30 | 0.44 | 3.90 | 0.71 | 5.06 |
| 31-40 | 0.44 | 7.97 | 0.71 | 9.12 |
| 41-50 | 0.55 | 11.76 | 1.35 | 14.12 |
| 51-60 | 1.81 | 30.06 | 1.81 | 33.67 |

Table 2 Average Routing Time based on 10000 Random Tests of 10 Oct-trees

| | |
|---|---|
| $T_{wp}$ (sec) | Time to locate the way-points in the obstacles quad-tree |
| $T_{vg}$ (sec) | Time to construct the visibility graph |
| $T_{sp}$ (sec) | Time to search a path in the visibility graph |
| Total (sec) | Total time to route |
| Distance (km) | Path Distance from the start to the goal points |



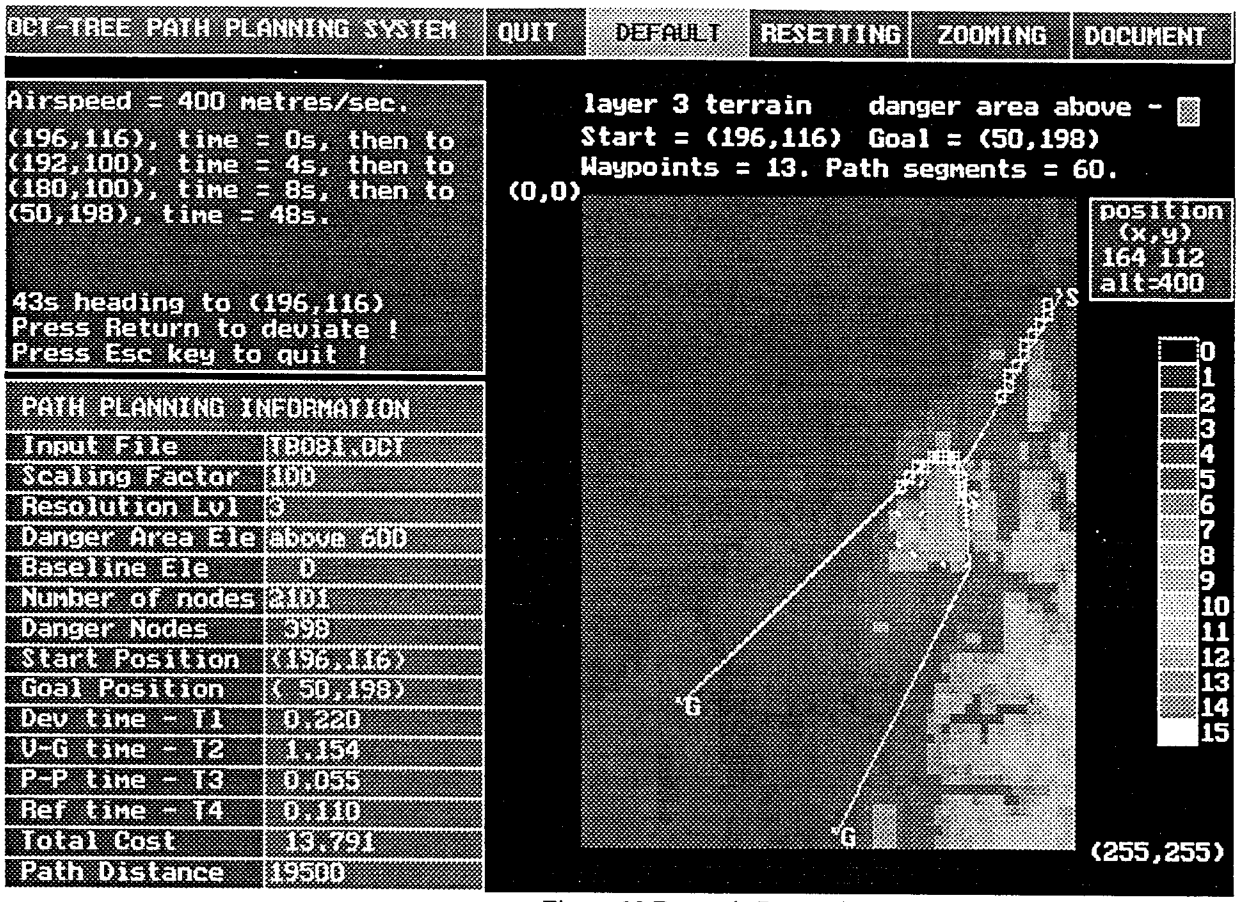
```
Airspeed = 400 metres/sec.
(247,36), time = 0s, then to
(196,104), time = 21s, then to
(200,144), time = 30s, then to
(131,249), time = 61s.

Fly over !!!
```

```
layer 3 terrain    danger area above - ▨
Start = (247,36)   Goal = (131,249)
Waypoints = 13. Path segments = 54.
```

```
position
 (x,y)
232 52
alt=400
```

```
PATH PLANNING INFORMATION
Input File          TB081.OCT
Scaling Factor      100
Resolution Lvl      3
Danger Area Ele     above 600
Baseline Ele          0
Number of nodes     2101
Danger Nodes         398
Start Position      (247, 36)
Goal Position       (131,249)
Dev time - T1        0.220
V-G time - T2        0.989
P-P time - T3        0.055
Ref time - T4        0.110
Total Cost           3.132
Path Distance       24700
```

Figure 17 Routing at Level 3 - Routing Time 3.1 seconds

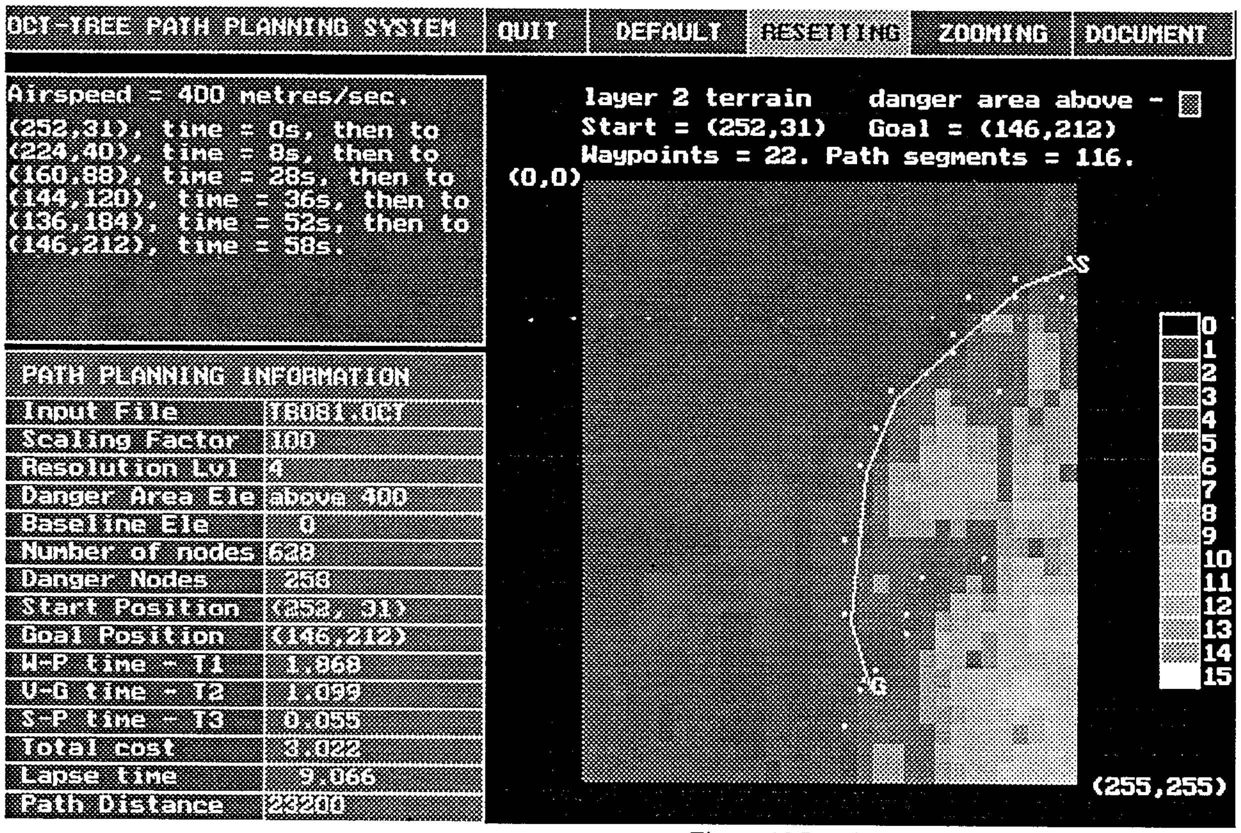


```
Airspeed = 400 metres/sec.
(196,116), time = 0s, then to
(192,100), time = 4s, then to
(180,100), time = 8s, then to
(50,198), time = 48s.

43s heading to (196,116)
Press Return to deviate !
Press Esc key to quit !
```

```
layer 3 terrain    danger area above - ▨
Start = (196,116)  Goal = (50,198)
Waypoints = 13. Path segments = 60.
```

| PATH PLANNING INFORMATION | |
|---|---|
| Input File | TB081.OCT |
| Scaling Factor | 100 |
| Resolution Lvl | 3 |
| Danger Area Ele | above 600 |
| Baseline Ele | 0 |
| Number of nodes | 2101 |
| Danger Nodes | 398 |
| Start Position | (196,116) |
| Goal Position | ( 50,198) |
| Dev time - T1 | 0.220 |
| V-G time - T2 | 1.154 |
| P-P time - T3 | 0.055 |
| Ref time - T4 | 0.110 |
| Total Cost | 13.791 |
| Path Distance | 19500 |

Figure 18 Dynamic Re-routing at level 5 - Routing Time 13.8 seconds



```
Airspeed = 400 metres/sec.
(252,31), time = 0s, then to
(224,40), time = 8s, then to
(160,88), time = 28s, then to
(144,120), time = 36s, then to
(136,184), time = 52s, then to
(146,212), time = 58s.
```

```
layer 2 terrain    danger area above - ▨
Start = (252,31)   Goal = (146,212)
Waypoints = 22. Path segments = 116.
```

| PATH PLANNING INFORMATION | |
|---|---|
| Input File | TB081.OCT |
| Scaling Factor | 100 |
| Resolution Lvl | 4 |
| Danger Area Ele | above 400 |
| Baseline Ele | 0 |
| Number of nodes | 628 |
| Danger Nodes | 258 |
| Start Position | (252, 31) |
| Goal Position | (146,212) |
| W-P time - T1 | 1.868 |
| V-G time - T2 | 1.099 |
| S-P time - T3 | 0.055 |
| Total cost | 3.022 |
| Lapse time | 9.066 |
| Path Distance | 23200 |

Figure 19 Routing at level 2 - Routing Time 3.0 seconds