Two types of algorithms for fast 3D robot path planning

van Tuijl, Frank W.M.

Award date:
1987

Eindhoven University of Technology (EUT)
Department of Electrical Engineering
Measurement and Control group

Philips Research Laboratories Eindhoven (PRLE)

TWO TYPES OF ALGORITHMS FOR FAST 3D ROBOT PATH PLANNING

Frank W.M. van Tuijl

Thesis

Supervisors : Prof. Ir. F.J. Kylstra (EUT)
Dr. Ir. F.L. Engel (PRLE)
Ir. N.G.M. Kouwenberg (EUT)

Eindhoven, Holland, August 1987

Tuijl, F.W.M. van; **Two types of algorithms for fast 3D path planning.** Thesis, Measurement and Control Section ER, Eindhoven University of Technology.

ABSTRACT

Two types of algorithms for three dimensional robot path planning were experimentally evaluated with regard to the computation time, the memory requirements and the path length, as a function of the resolution of the robot work space. The first type of algorithm, the Constrained Distance Transform, is based on the application of a masking operation on all cells of the robots mono-resolution workspace. The second type of algorithm is based on the A^* heuristic search algorithm, using a set of (not necessarily) orthogonal projections of the workspace. A quadtree representation of the projections accelerates the algorithm by enabling multi-resolution search. Two new techniques for multi-resolution search are proposed and were evaluated : local octree search and quad-lattice search.
From the experiments it was observed that the quad-lattice search incorporates a clear improvement compared to local octree search with regard to computation time and memory requirements at the expense of a slight increase in the path length. Secondly, the quad-lattice and local octree search algorithms were superior to the Constrained Distance Transform algorithm with regard to the same criteria mentioned, in workspaces with a resolution of respectively 16x16x16 and 64x64x64 cells or more.

Tuijl, F.W.M. van; **Twee soorten algorithmen voor snelle 3D robot pad-planning.** Afstudeerverslag, vakgroep ER, Technische Universiteit Eindhoven, Augustus 1987.

SAMENVATTING

Twee soorten algorithmen voor drie dimensionele robot pad-planning zijn experimenteel geëvalueerd m.b.t. de rekentijd, het geheugen gebruik en de pad lengte, als functie van de resolutie van de robot werkruimte. Het eerste soort algorithme, de 'Constrained Distance Transform', is gebaseerd op een maskeer operatie op alle cellen van de robot's werkruimte. Het tweede soort algorithme is gebaseerd op het heuristisch zoekalgorithme A^*. Het maakt gebruik van drie (niet noodzakelijkerwijs) orthogonale projecties van de werkruimte. Door de projecties in een quadtree te representeren, wordt het algorithme versneld omdat het zoekt op verschillende resoluties. Twee nieuwe 'multi-resolutie' zoek methoden zijn beschreven en geëvalueerd : de 'local octree' methode en de 'quad-lattice' methode.
Uit de experimenten is gebleken dat de 'quad-lattice' methode essentieel beter is dan de 'local octree' methode m.b.t. de rekentijd en het geheugen gebruik, echter ten koste van een kleine toename in de pad lengte. Ten tweede is gebleken dat zowel de 'quad-lattice' methode als de 'local octree' methode superieur is aan de 'Constrained Distance Transform' m.b.t. tot dezelfde criteria, in werkruimten met een resolutie van resp. 16x16x16 en 64x64x64 cellen of meer.

CONTENTS

PREFACE

This thesis is the outcome of a research project that was performed in the Control, Measurement and Automation Group of the Philips Research Laboratories in Eindhoven, Holland, from September 1986 until August 1987. The project was a cooperation with the Measurement and Control Group of the Faculty of Electrical Engineering at the Eindhoven University of Technology.

I would like to thank my supervisors, especially F.L. Engel, for their support and encouragement during my work. Finally I owe gratitude to R.C. van Ommering for his assistance for making a demonstration possible at the Concern Research Exhibition, May 1987.

Frank W. M. van Tuijl

August 1987

1. INTRODUCTION

1.1 CONTEXT

Part of the research activities of the section for Control, Measurement and Automation at the Philips Research Laboratories Eindhoven, relates to making robot control more intelligent. More precisely this implies that off-line programming of robots would become easier by using a high level of language, based on the specification of *what* should be done at task level rather than *how* it should be done at manipulator level. Moreover, the research activities are aimed at making robot task execution more tolerant for exceptions and deviations in the environment. This requires that the system has to keep a model of the expected successive stages of the robot workspace. Sensory checks have to verify whether the actual situation corresponds with the expected situation. Once a discrepancy has been detected and its cause has been diagnosed, automatic (or manual) recovery has to bring the system back towards completion of the desired task. Automatic recovery requires on-line planning of a sequence of recovery actions.

Automatic path planning is one of the subjects that fits in this scheme. On the one hand, it can be applied during off-line task level programming for generating the collision free robot movements, resulting in a less specific way of programming and therefore a shorter programming time. On the other hand, it can be applied for on-line generation of safe paths needed for recovery actions. So a higher flexibility and reliability in the task performance of a robot system can be achieved.

1.2 OBJECTIVES

Robot path planning aims at generating collision free paths for a robot with a payload from a given start position towards a given goal position, in a workspace containing obstacles. The obstacles are non-stationary, but they are assumed not to move during the planning and execution of the path.

Path planning requires a world representation in which the description of the robot workspace has been accumulated. This description can be obtained in general from various sources including manual input, object data bases and sensory information. In this case, either a CAD modeller for off-line programming or sensor information from TV cameras for on-line programming, should be used. The latter includes computer vision for identifying three dimensional obstacles. Previous approaches therefore can be classified as model-based or non-model-based. In a model-based approach, models of three dimensional objects are stored in the computer and are used as references for identifying and recognizing objects. In a non-model-based approach, 3-D objects are reconstructed using foreground/background information only. Complete reconstructions are unlikely with a finite number of views. Object approximations are often used.

In this report, a non-model-based approach is presented, using three orthogonal projections of the robot workspace. For path planning purposes, there is less interest in the precise shape of the obstacles; however, the interest here is where there are obstacles and where there is free space.

The representation in which the search for collision free paths is performed is called the search space representation. This space may differ from the cartesian world representation e.g. the robot joint space. If it is different, a procedure is required that maps the world space into the search space. In general, such a procedure will be quite time consuming. Most path planning algorithms operate with 2 or 3 dimensions only. More dimensions make the search for paths and the mapping procedure computationally too expensive, besides these procedures tend to require a lot of computer memory.

The main bottlenecks of on-line path planning are its time complexity and memory requirements. To meet these bottlenecks, the path planning problem in this investigation was simplefied to moving a rigid body, representing the gripper and payload, in a 3-D space with three translational degrees of freedom. Accordingly, the robot links were assumed not to interfere with the obstacles. No transformation from the robot world representation to the search space representation is required then. By extending the obstacles and shrinking the object to be moved, the path planning problem can be reduced futher to moving a single point through the search space.

In literature, two promising search algorithms were found that relate to 'fast' on-line path planning :

(1) **Multi-resolution search based on orthogonal projections of the workspace.** Wong and Fu [WON86] were the first to report on this approach. They decomposed the 3-D collision test into three 2-D collision tests, thus without an explicit reconstruction of the 3-D representation. A path was searched in a breadth first-fashion on multiple levels of resolution of the search space simultaneously.

(2) **Constrained Distance Transform** [DOR86], a recursive implementation of the Lee algorithm [LEE61], operating in a mono-resolution search space. The method is not very refined; however, its simplicity makes it very suitable for hardware implementation and accordingly it promises high speed.

The multi-resolution orthogonal projection approach was improved by using a more informed search algorithm called A^* (said as A-star), instead of a purely breadth-first search, as Wong and Fu [WON86] applied, and by introducing a more efficient hierarchical search space encoding. In this investigation the A^* based approach was evaluated and compared with the Constrained Distance Transform as described in literature [DOR86]. This comparison was primarily focussed on the computation time, the memory requirements and the deviation from the geometrically shortest path.

2. BASIC CONCEPTS

2.1 QUADTREES AND OCTREES

A quadtree is a hierarchical representation of 2-D silhouette images, see [SAM84]. It can be obtained by recursively subdividing non-homogeneous square regions into quadrants. Initially, the entire image is represented by a single node in the quadtree which is called the root node. If this image is homogeneous (that is completely occupied or completely empty) it is not decomposed further. Otherwise it is split into four subregions of equal size which become the children of the root. This process continues until all the regions are homogeneous or some resolution limit is reached. All terminal nodes (leaves) are either *full* (black) or *empty* (white), the non-terminal nodes are called *mixed* (grey).

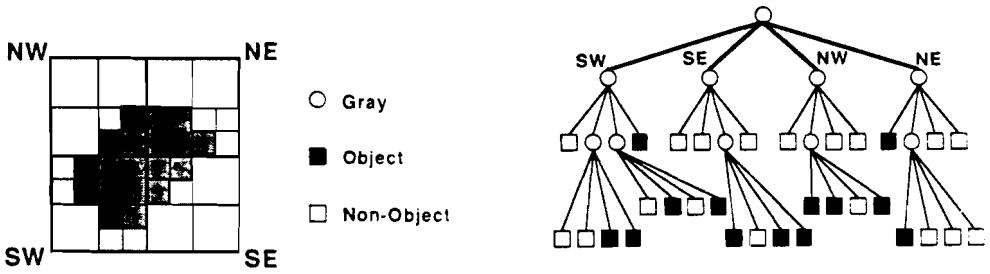


Fig. 2.1 : An image and its quadtree representation (from [CHI86]).

Fig. 2.1 shows the quadtree representation of an image.
Hunter and Steiglitz [HUN79] proved that the complexity of the quadtree representation, that is the total number of nodes in the tree, is O(p+q). Here q denotes the depth of the tree and p the total perimeter of the objects(s), expressed in the length of the edge of the smallest available squares in the description. The value of the perimeter p largely dominates the value of the depth q, so that the complexity approximates O(p), see fig. 2.2. Note that the perimeter p will increase proportional to the resolution.

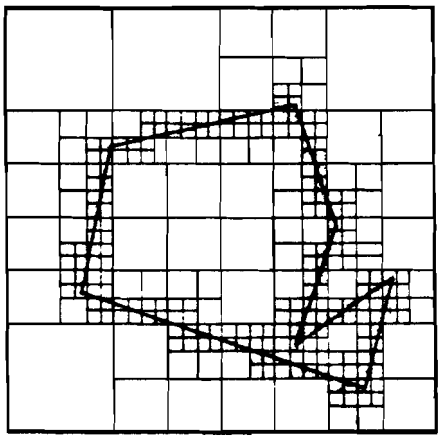


Fig. 2.2 : A quadtree showing the dominating number of smallest squares located along the object edges (from [HUN79]).

An octree is a generalization of the quadtree concept to a 3-D voxel-based space description, see [JAC80]. Each non-homogenous volume is then subdivided into eight sub-volumes of equal size, see fig. 2.3 . An octree has the same complexity bound O(p+q) as the quadtree. However, p represents the total surface of the objects measured by the surface area of the smallest cubes. Note that this expression increases quadratically with the resolution.

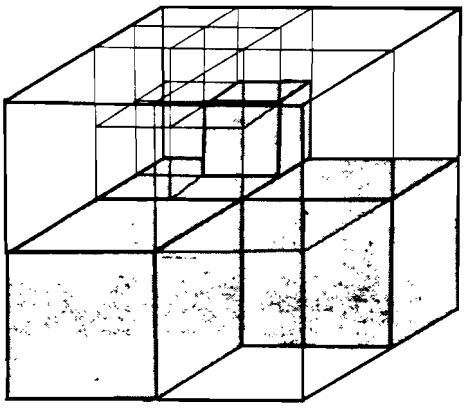


Fig. 2.3 : An example of an octree representation.

2.2 A* : HEURISTIC SEARCH ALGORITHM

*A** (said as A-star) is a heuristic search algorithm. In the following only the essentials will be indicated briefly. For a more detailed description see Nilsson [NIL80] or Pearl [PEA84].

*A** administrates *nodes*, that are states in the search space, in a *search tree*. The algorithm is based on an evaluation function f which gives the 'promise' of a node n with regard to the cost of a path in the search tree from start to goal via node n. The function f consists of adding two parts :

$$f(n) = g(n) + h(n) \qquad \text{for all nodes } n$$

where g(n) is a cost function denoting the sum of all arc costs from the start node s to node n. h(n) denotes an optimistic estimate of the path cost from node n to the goal node e. The *A** algorithm expands continuously the nodes with the best 'promise', that is the lowest f value. Expanding a node means that all its successors are made available to the search process. The effect of g in the evaluation function f is to introduce a breadth-first component. Without h, *A** reduces to purely breadth-first search. The heuristic function h adds a depth-first component.

The properties of *A** depend on the choice of heuristic function h. Nilsson [NIL80] proved that *A** finds an optimal solution if the estimated cost h is always optimistic, which means that h never overestimates the cost to be made. Then, the algorithm is said to be *admissible*, indicating that it will always find an optimal path with the lowest possible cost, provided a path from s to e exists. The closer h approximates to the real cost of the path from n to the goal, the fewer nodes will be expanded and therefore the more *efficient* the *A** algorithm will be. Martelli [MAR77] proved that if the

estimate h is *consistent* (or *monotone*) which means that the decrease in h for any descendant node never exceeds the actual arc cost(s) g, A^* requires O(N) expansions. Here, N denotes the total number of possible nodes in the search tree. Non-consistent heuristics require $O(2^N)$ expansions. Finally, Pearl [PEA84] showed that the A^* algorithm is *optimal for consistent heuristics*, meaning that it is more efficient than any other admissible algorithm.

In detail, the A^* algorithm uses two node collections called *OPEN* and *CLOSED*. *CLOSED* contains all nodes which have been expanded, i.e. their successors are available to the search process. *OPEN* contains the nodes that are generated and are awaiting for expansion. The algorithm becomes :

(1) Put the start node(s) in *OPEN*;
(2) If *OPEN* is empty, exit with failure: no solution exists;
(3) Remove node n with the lowest f from *OPEN* and put it into *CLOSED*;
(4) If n is a goal node, exit successfully by tracing back a path from n to start node s;
(5) Expand node n by generating all its successors with pointers back to n;
(6) For every successor n' of n :
    (a) Calculate f(n');
    (b) If n' is neither on *OPEN* or on *CLOSED*, add it to *OPEN*;
    (c) If n' resides on *OPEN* and the newly calculated f(n') is smaller than the previously assigned f, substitute the new value f for the old one and redirect the pointer to the new predecessor n. In all other cases discard the newly generated n';
(7) Go to step 2;

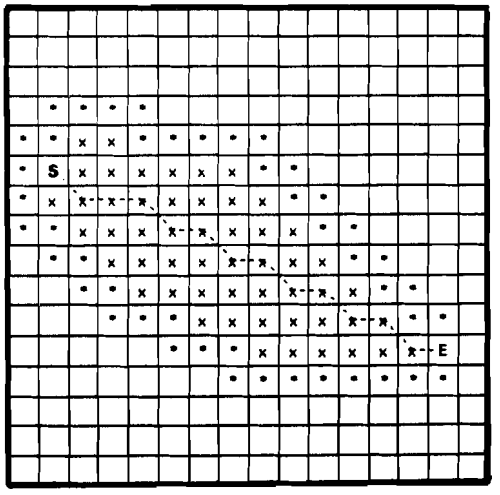


Fig. 2.4 : The nodes expanded by A^* in an empty search space during search from start node 's' to goal node 'e'. A shortest path according to the chamfer distance metric is dotted. Nodes marked 'x' are in CLOSED, those marked '•' are in OPEN.

Fig. 2.4 illustrates what nodes have to be expanded for an empty search space which is represented by a 2-D array. Every empty cell corresponds to a possible node in the search tree. If there had been any obstacle cells they could be represented by forbidden nodes. A path to a cell can continue

towards all its eight neighbouring cells. The so called *euclidian chamfer metric* was used to measure the distances. According to this metric, horizontal and vertical path segments have length 1 and diagonal path segments have length $\sqrt{2}$, see fig. 2.5. In the example of fig. 2.4, g(n) corresponds to the euclidian chamfer distance from s to n, and h(n) to the euclidian distance from n to e. A shortest path (dotted) according to this metric is found from the start node marked 's' (in the upper left part) to the goal node marked 'e' (in the lower right part). Nodes marked 'x' are all expanded (in *CLOSED*), nodes marked '*' are possible candidates for expansion (in *OPEN*). Pearl [PEA84] showed that in a 2-D array without obstacles, the expanded nodes are roughly bounded by an elliptic area satisfying $g(n) + h(n) \leq f(e)$.

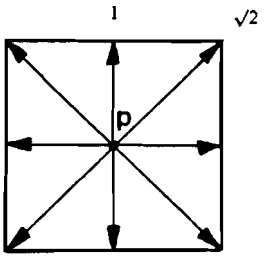


Fig. 2.5 : The permissible path segments from point P according to the euclidian 2-D chamfer distance metric. The values 1 and $\sqrt{2}$ indicate the length of the path segments.

2.3 CONSTRAINED DISTANCE TRANSFORMATION (CDT)

Distance transformation is an operation that generates for every point in a picture, its shortest distance to a given goal point (or set of goal points). The picture containing the resulting distances for all the points is called a distance image. The shortest path can be found by following the steepest distance gradient from an arbitrary starting point to the goal. Thus, the distance transformation transforms a global optimalization problem (find the shortest path) to a local problem (follow the local gradient).

The Lee algorithm [LEE61] was an early approach to generate a distance transformation in a breadth-first fashion. Initially starting from the goal point, it adds continuously a layer to the distance wavefront; however, this requires an extensive overhead of points to be treated and therefore it is rather slow.

Borgerfors [BOR84] proposed a fast method for distance transformation by an iterative 2-pass masking operation. Firstly, the distance image has to be initialized to either zero for goal points, or to infinity for the remaining points. Then, two masks W and W' are run successively over the distance image. For the euclidian chamfer metric, the masks are shown in fig. 2.6.

Fig. 2.6 : The masks W (left) and W' of the CDT for calculation of the euclidian chamfer distance.

These masks are placed over the distance image. Each mask coefficient is added to the value beneath it and the least of sums within the mask is selected. This result replaces the distance image value at the mask center ('0' position). Then, the mask is moved to the next position and the procedure is repeated. Firstly, W is run from left to right and from top to bottom, then W' in the opposite direction. In the absence of obstacles, these two passes will be sufficient for generating the distance image.

In the presence of obstacles the calculation of distance is more complicated. Dorst [DOR86] showed that it suffices to introduce constraints on the distance calculations : obstacle points are given the value infinity at initialization and are kept infinite during the distance transformation. This procedure can be seen as imposing a 'write protection' on obstacle locations. However, two passes may not be sufficient to generate the distance image, in which case the masks W and W' must be alternated until no further change occurs. This is illustrated in fig. 2.7. The equi-distance lines are given, obtained after the successive passes.

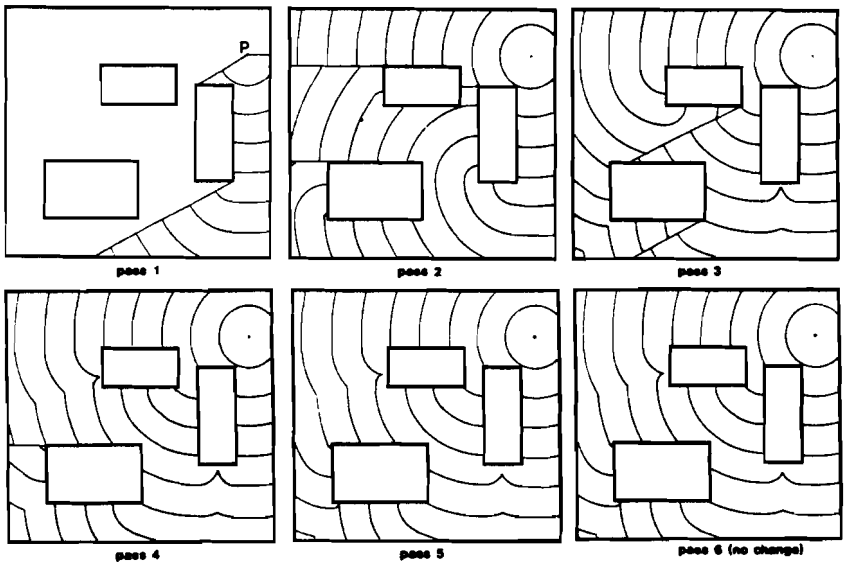


Fig. 2.7 : The equi-distance lines in a distance image after the alternate passes of the masks W and W' (from [DOR86]).

The shortest path from an arbitrary point A to the goal point P can be found by following the steepest descent in the distance image, perpendicular to the equi-distance lines, see fig. 2.8.

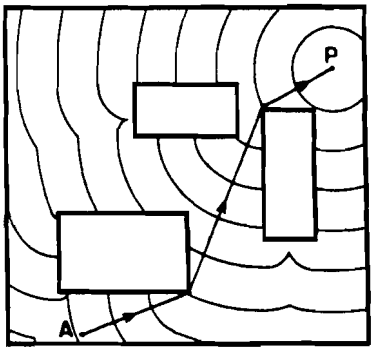


Fig. 2.8 : The shortest path from A to P in a distance image is found by following the steepest gradient, perpendicular to the equi-distance lines (from [DOR86]).

This operation is called Constrained Distance Transformation (*CDT*). Borgefors [BOR84] showed that the transformation can easily be extended to an arbitrary number of dimensions.

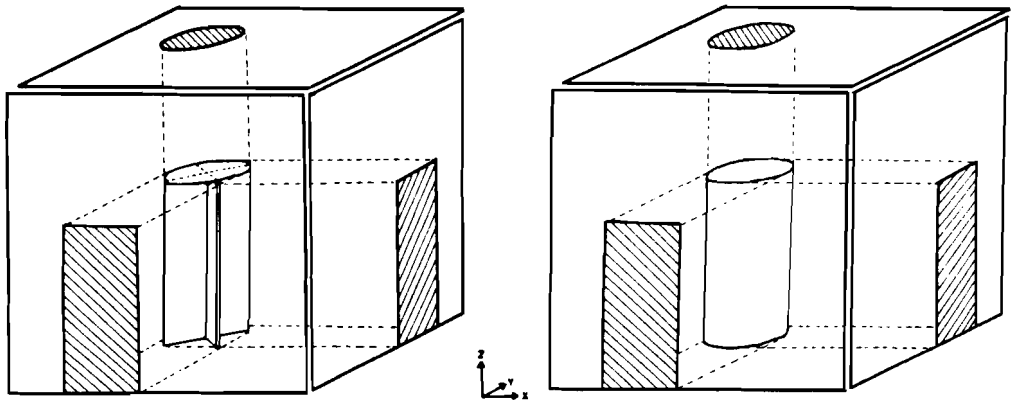
The *CDT* requires O(N) masking operations, where N is the total number of cells in the distance image. The memory usage and execution time can be reduced by using integer arithmetic instead of floating point. The 1:$\sqrt{2}$ ratio of the masks W and W' can be approximated, for example by 3:4 or 5:7.

The main advantage of the *CDT* algorithm is that it is very fast in comparison to other more conventional methods, for example the Lee algorithm. Moreover, its simplicity makes it very suitable for hardware implementation.

3. ORTHOGONAL PROJECTION APPROACH

3.1 PRINCIPLES

Analogous to Wong & Fu [WON86], three orthogonal projections of the workspace were used. The 3-D objects can be partially reconstructed by combining their projections; in this way a so-called *maximum volume* is created, enclosing the true volume of the objects. 3-D points that are not enclosed by the maximum volume do not coincide with the true object. Fig. 3.1 illustrates the reconstruction of a maximum volume.



(a) A 3-D object and its projections. *(b) Reconstruction of the maximum volume from the projections.*

Fig. 3.1 : The reconstruction of an object's maximum volume from its orthogonal projections.

In general, a complete reconstruction of the true volume of the 3-D objects in a workspace cannot be made from a finite number of views. The accuracy of a maximum volume related to the true volume depends on the shape (i.e. the presence of concavities), the clustering of obstacles, the number of projections applied and the position of the projection planes. For path planning however, the precise shape of the obstacles is less interesting than the information on where regions of free space occur. Therefore, approximations of the obstacles by maximum volumes will often be adequate.

Three dimensional collision detection can be decomposed into 2-D collision detection in the orthogonal projections with the following lemma, cf. [WON86]:

> A 3-D point in space is guaranteed not to coincide with any obstacle if it is outside the occupied area in at least one of the projections.

Thus the full 3-D world does not have to be represented in an explicit three dimensional model. A partial 3-D reconstruction of a 3-D space by means of three 2-D projections is performed by the search algorithm. Such a reconstruction is limited to the region of relevance for the search algorithm. In general, this results in considerable savings in computatation time and memory requirements for a 3-D model.

To deal with physical extensions of the payload and gripper, obstacle growing was used. Obstacle growing means expanding the obstacles in the workspace

and shrinking the objects to be moved to a point. In this case the obstacles are grown (anisotropically) in the three 2-D projections.

As mentioned in the introduction, the 2-D projections can either be provided by a geometric modeller or by TV cameras.

3.2 PATH PLANNING ALGORITHM

In this section a hierarchical path search strategy is presented, based on three orthogonal projections of the workspace. To accelerate the search and to decrease the memory requirements, quadtree representations are used for all three orthogonal projections. The algorithm contains the following sequential steps :

(1) Encode each of the three orthogonal projections containing the grown projections of the obstacles into a quadtree.
(2) Connect the given 3-D start and goal points with all corners of their associated surrounding 3-D cubes in the search space.
(3) Initiate the A^* search from the eight start corners as described under (2). The calculation of the expansion nodes is described below. The search algorithm terminates when any of the goal corners is reached or when all possible nodes are expanded and apparently no path exists.

In general, the node expansions are based on the following principles :

* A 3-D path segment is free if it is free in at least one of its 2-D projections.
* A node can be expanded in at most 26 directions, obtainable by combining a step (+1,0,-1) along each of the three main axes of the workspace, see fig. 3.2.

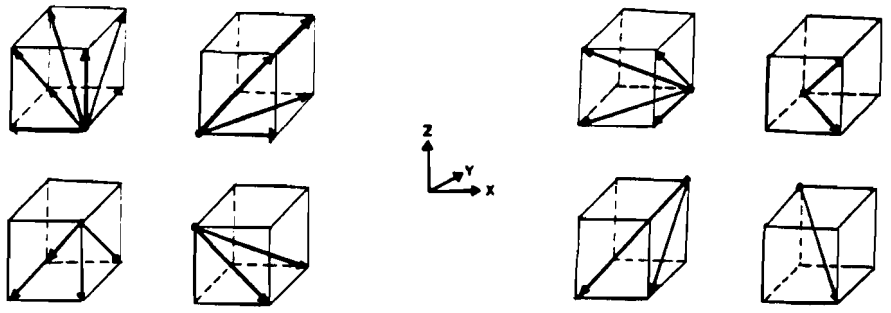


Fig. 3.2 : 26 permissible expansion directions of a 3-D point (from [WON86]).

The reason for restricting the number of expansion directions is that, in comparison to expansion in arbitrary directions, this is substantially more efficient for the calculation of coordinates of the successor nodes and for collision checking, due to the fact that the projected path segments are always located horizontally, vertically or diagonally in the projections. However, a drawback is that the length of the actual path found may be longer than the geometrically shortest path.

Another advantage of the limited number of node expansion directions is that the 3-D euclidian chamfer metric can be used to calculate the g and h component of the evaluation function f. The 3-D euclidian chamfer metric is a generalization of its 2-D metric. According to this metric a path can be continued in at most 26 directions. Path segments along resp. one, two or three axes of the search space have relative length $1:\sqrt{2}:\sqrt{3}$.

For combining the three 2-D quadtree representations into a 3-D representation, two strategies have been developed and evaluated :

- **LOCAL OCTREE** : Based on the local reconstruction of an octree.
- **QUAD-LATTICE** : A more constrained strategy than local octree that only allows those free 3-D path points for which the projections are located on quadtree division lines in all projections simultaneously.

The local octree search space representation is comparable to the search space representation described by Wong and Fu [WON86]. Quad-lattice search is newly proposed in this thesis. In the following, it will be described in more detail how the expansion nodes are calculated for both strategies.

3.3 LOCAL OCTREE STRATEGY

The local octree strategy is based on an octree reconstruction of the workspace by relating the three orthogonal quadtree coded projections. The coordinates of the neighbouring nodes in each of the 26 directions are calculated in two sequential steps:

(1) Reconstruction of surrounding cubes around the expansion point, see fig. 3.3. Determination of the size and contents (*full* or *empty*) of these cubes. Note that the sizes of the cubes are not necessarily the same.
(2) Determination of the length of the path segments and checking their *permissibility* in 26 directions. Finally the coordinates of the newly permitted search nodes are calculated.

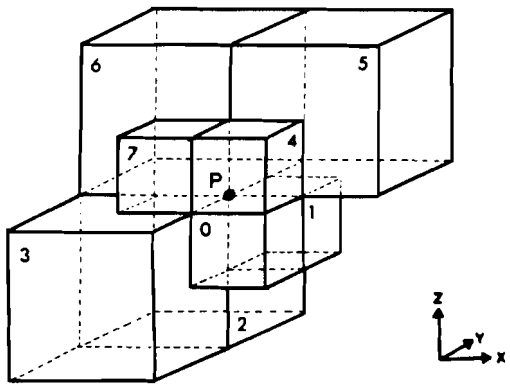


Fig. 3.3 : An example of the surrounding cubes around expansion point P.

The *permitted* nodes were defined to be located at one of the corners of an associated octree cube. The Individual path segments will always start and end

at different corners of the same cube. The cubesizes in octrees can switch from one resolution level to another at points where cubes from different hierarchy levels meet, just like squares in quadtrees. The cubesizes can change by a power of 2 in length.

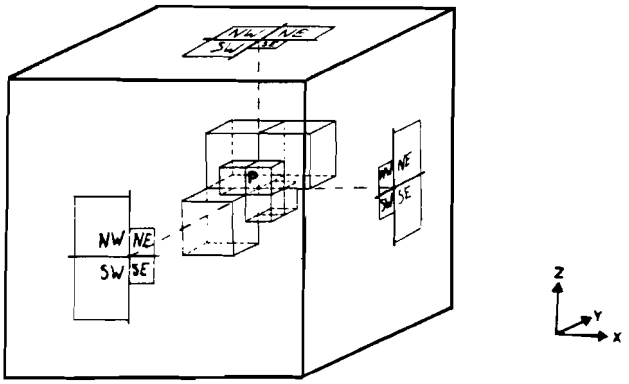


Fig. 3.4 : The reconstruction of the surrounding cubes around a 3-D point P from the orthogonal quadtree coded projections.

The reconstruction of the surrounding octree cubes is illustrated in fig. 3.4. A permitted 3-D path point P is projected on each of the 3 projection planes. Each projection of that point is surrounded by at most four different quadtree squares that usually differ in size, respectively in North-West, North-East, South-East and South-West directions. The size and contents (*empty* or *full*) of a 3-D cube is then determinated by relating it to the associated quadtree squares according to the following rule :

(1a) If at least one of the corresponding quadtree squares is empty, mark the cube as empty and set its size to the maximum size of the corresponding empty squares.
(1b) If none of the corresponding quadtree squares is empty, mark the cube as being full and set its size to the minimum size of the corresponding quadtree squares.

After having determined the sizes of the surrounding cubes, the length of the path segments can be calculated in at most 26 directions. Except for the 3-D path segments, the stepsize equals the minimum edge length of all the adjacent cubes. The minimum here is required for being able to reach all corners of the smallest cubes in the representation. For 3-D path segments there is only a single cube involved. Indeed the stepsize equals now the edge length of this cube.

For a 1-D movement, parallel to one of the 3 orthogonal main axes, four cubes are involved; e.g. in fig. 3.3 for a movement in the +x direction the cubes 0,1,4,5. The stepsize equals the length of the edge of the cubes 0,1 or 4. For a 2-D movement, along 2 main axes, two cube surfaces are involved; e.g in +x+y direction cubes 1 and 5. Now, the stepsize equals $\sqrt{2}$ times the length of the edge of cube 1. There is only a single cube involved for a 3-D move, with stepsize $\sqrt{3}$ times the length of the edge of this cube.

Finally, all path segments have to be checked for *collision* with any obstacle and for *validity*. For the *collision* check, it is necessary for at least one of the surrounding cubes involved to be empty. A path segment is only *valid* if start and end points of the path segments are located at different corners of the same octree cube. Only those path segments are permitted which are both *collision free* and *valid*. Figure 3.5 illustrates all permitted path segments in a low resolution search space.

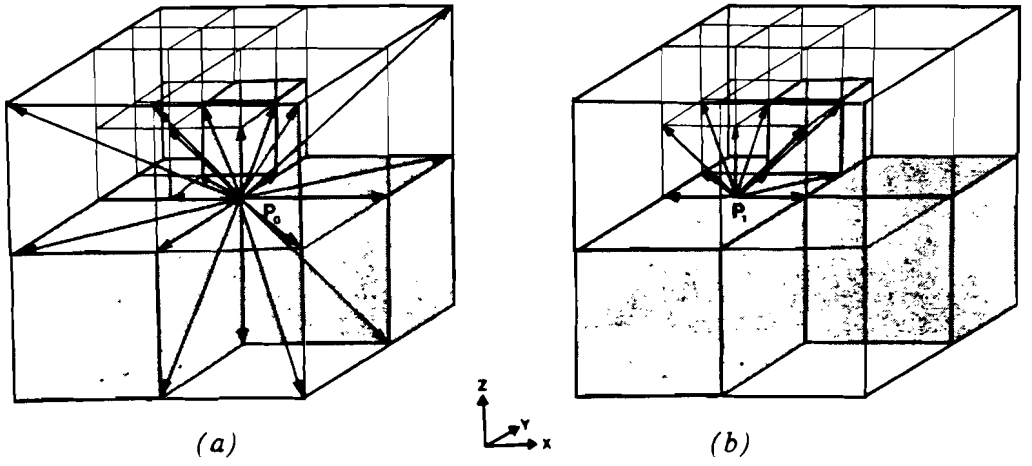


Fig. 3.5 : An illustration of the permitted 3-D path segments from points P_0 (a) and P_1 (b), according to the local octree expansion strategy.

Figs. 3.6 and 3.7 show examples of local octree search in a simple obstacle configuration, viz. a vertical wall placed diagonal to the xz-plane for two different levels of resolution. The *level of resolution* is defined as the ^{2}log of the number of smalllest subdivisions along the main axes of the search space. 's' marks the start point, 'o' the goal point and '×' positions of the expanded nodes. The shortest path found is drawn dotted. Note that the majority of expanded nodes is located at the surface of the wall. At higher resolution levels, for example in fig. 3.7, this number dominates the search process increasingly, thus, requiring a lot of computation time and memory space. A more efficient expansion strategy has been developed in regard to these two aspects. This strategy will be described in the next paragraph.

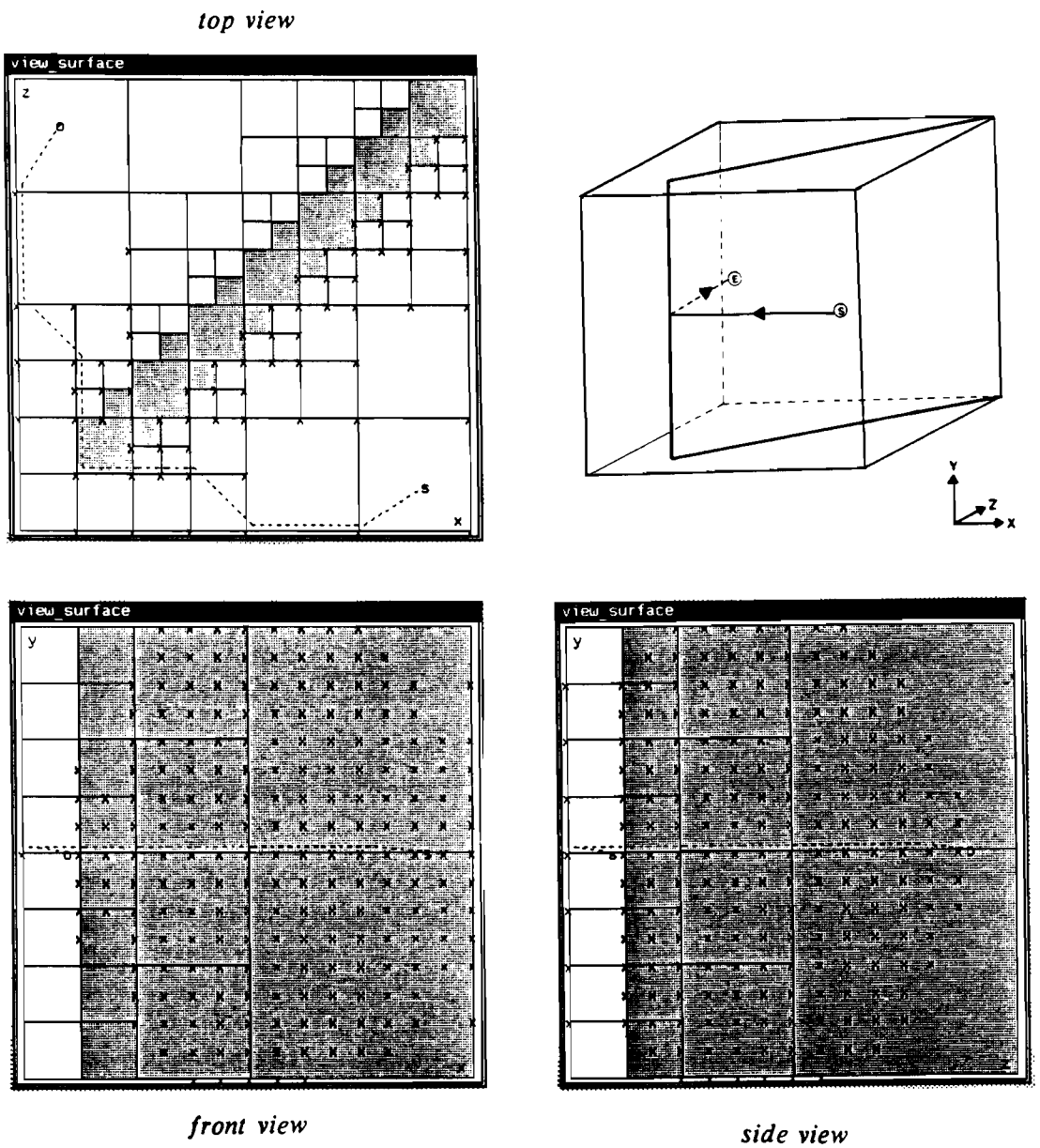


Fig. 3.6 : An example of local octree search for an obstacle configuration containing a wall at resolution level 4. 's' marks the start point, 'o' the goal point and 'x' the expanded nodes. The calculated path is dotted.

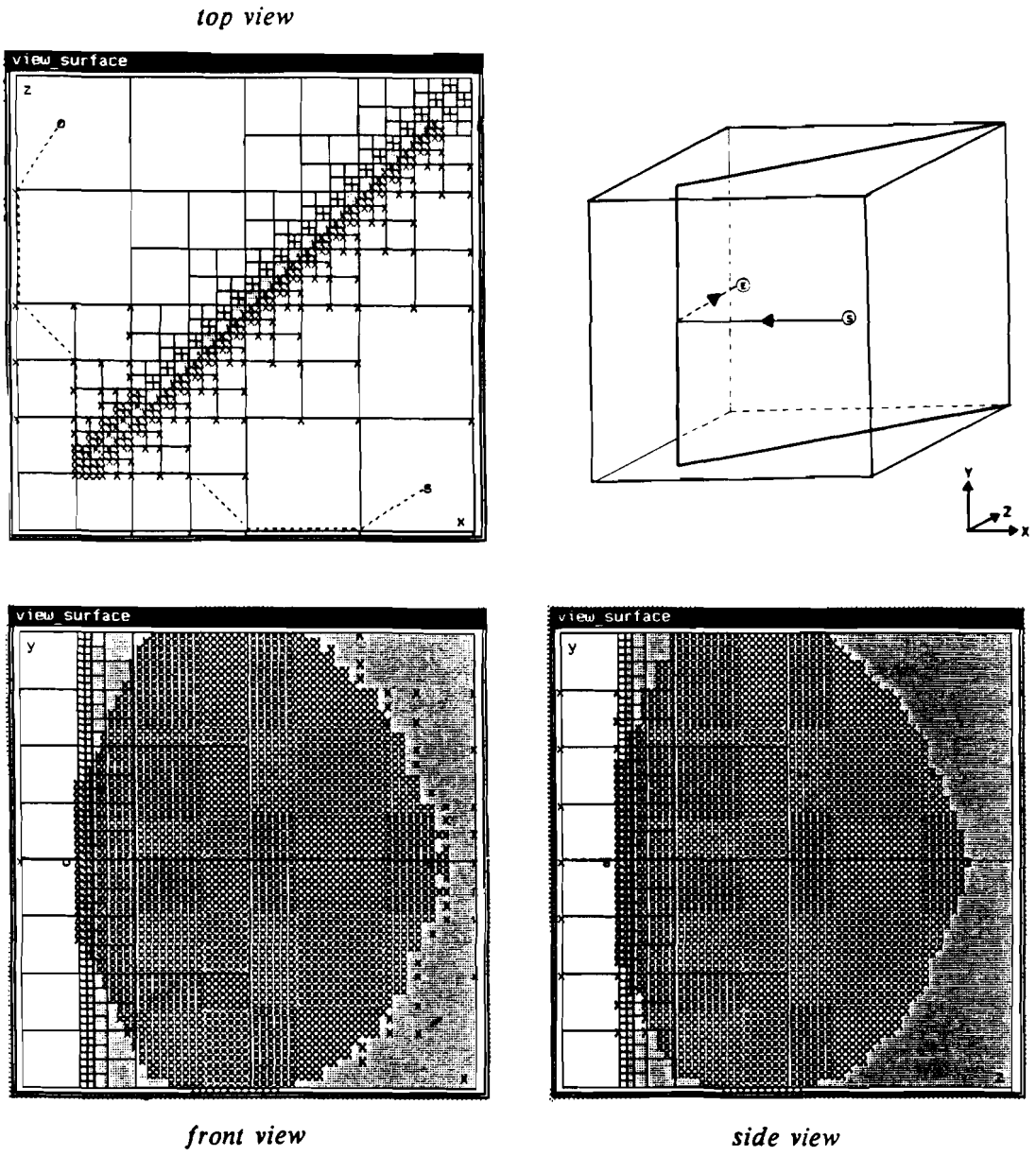


Fig. 3.7 : An example of local octree search in the same obstacle configuration as applied in fig. 3.6, however now at resolution level 6.

3.4 QUAD-LATTICE STRATEGY

In quad-lattice search, an attempt has been made to limit the nodes to be expanded futher. The strategy is to constrain the start and end points of the path segments to the quadrant division lines which are available from the three quadtree descriptions.

Fig. 3.8 illustrates the permitted path segments according to the quad-lattice strategy in the same obstacle configuration as used in fig. 3.5.

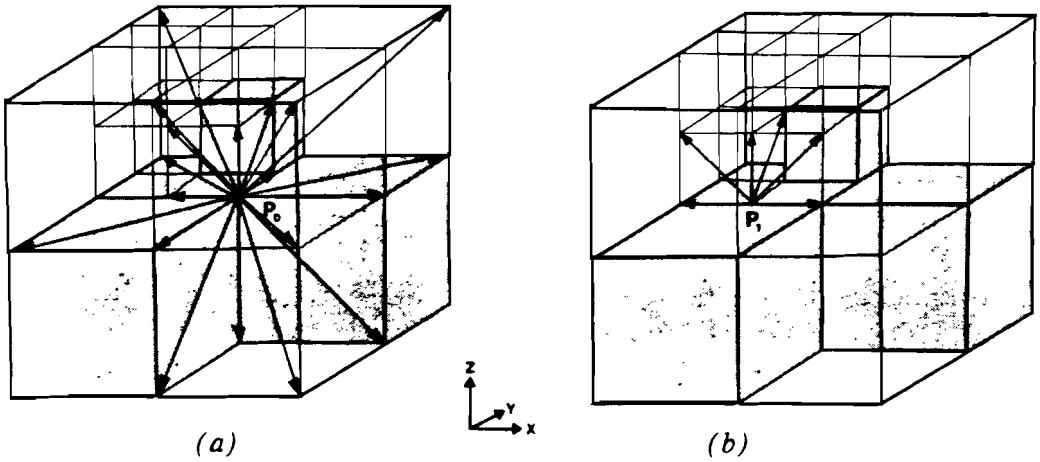


Fig. 3.8 : An illustration of the permitted path segments from points P_0 (a) and P_1 (b), according to the quad-lattice expansion strategy.

For a 1-D move, the largest free path segment is determined out of the two projections in which the path segment is located. A path segment in one projection equals the smallest adjacent quadrant. An additional constraint is that the path segments have to be along the quadtree division lines in both projections.

For a 2-D move, only the projection containing both axes is considered. The length of the path segment is equal to the size of the crossed quadrant. The additional constraint is that the path segment has to start and end at two different corners of that square.

A 3-D move is composed of a simultaneous move through squares in each of the 3 projections. To make the projections of the 3-D move end at a corner in all three projections, the length of the path segment is derived from the longest of the 2-D move projections. The longest path segment so derived must not pass through the occupied areas of the projections.

All paths segments must be checked for *collision* with any obstacle, in a similar way as for the local octree strategy. The *validity* is checked in the separate projections. In a single projection, only those path segments of which its projection passes two corners of the same square are valid. A 3-D path segment is only valid if all of its three projections are valid simultaneously. Figs. 3.9 and 3.10 show examples of quad-lattice search in the same obstacle configuration that was used in the figures 3.6 and 3.7.

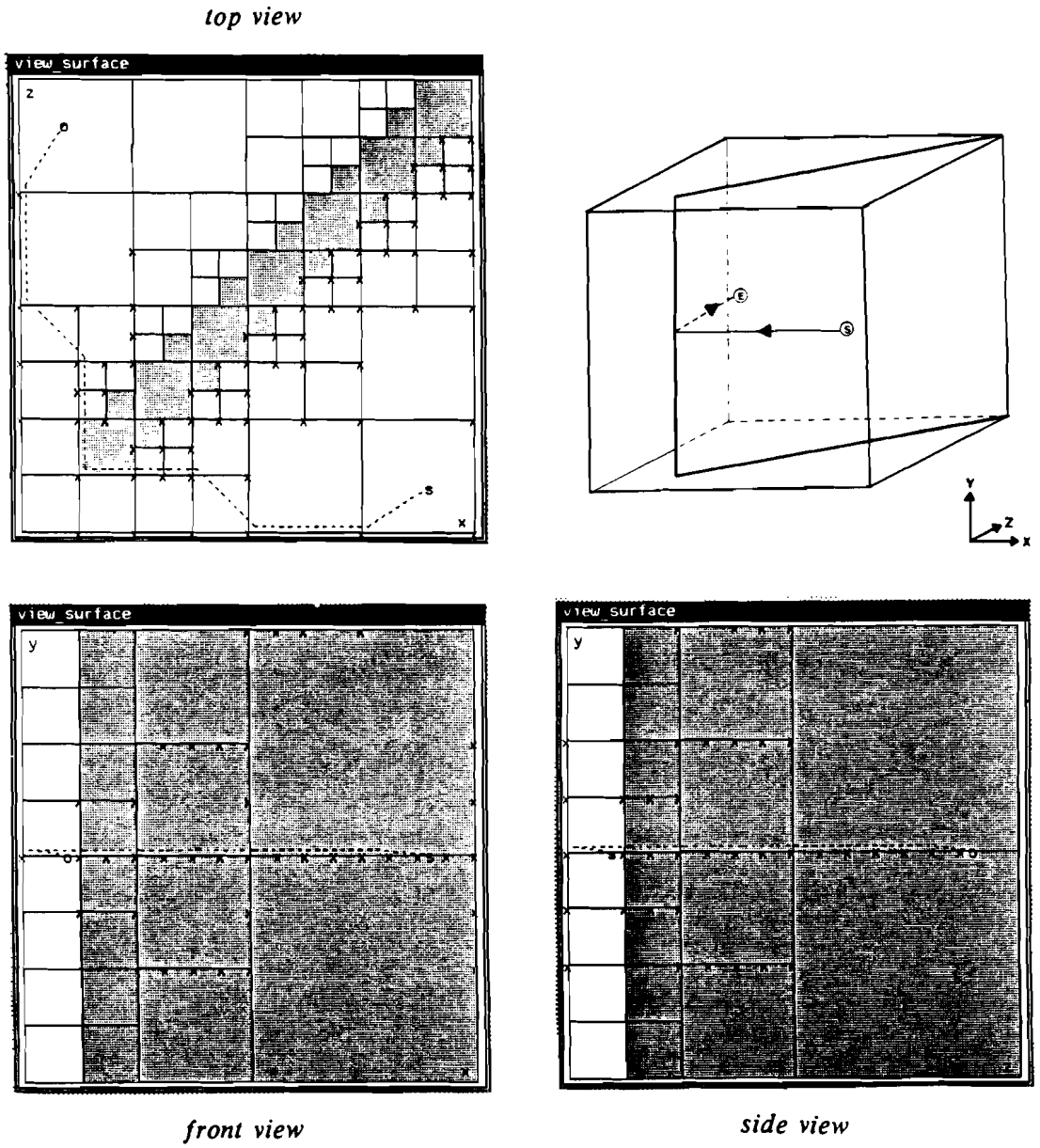


Fig. 3.9 : An example of quad-lattice search for an obstacle configuration containing a wall at resolution level 4. 's' marks the start point, 'o' the goal point and '×' the expanded nodes. The calculated path is dotted.

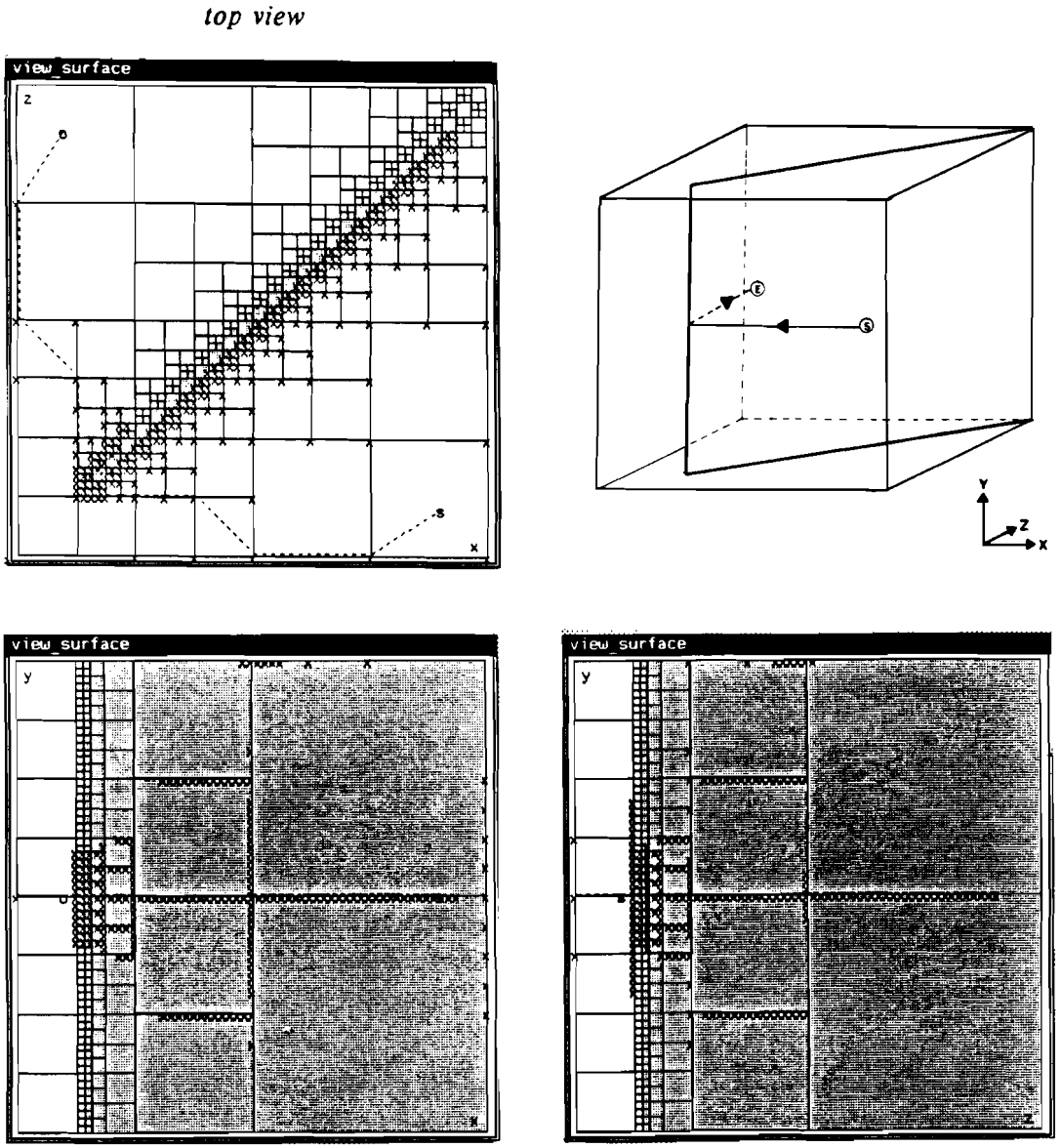


Fig. 3.10 : An example of quad-lattice search in the same obstacle configuration as applied in fig. 3.9, however now at resolution level 6.

4. COMPLEXITY MODELS

4.1 NODE COMPLEXITY

Local octree and quad-lattice strategy can be compared by means of the number of nodes expanded by A^*. We call this the *node complexity*.

As indicated in par. 2.2, A^* expands all the nodes within a fixed volume, independent of the resolution. This is due to the fact that the values of the evaluation function for these enclosed nodes does not exceed the cost of the optimal path from the start to the goal point. For a 3-D search space without obstacles, the volume is bounded by an ellipsoid. In the presence of obstacles, the volume will spread around the obstacles. The obstacle configuration of the figures 3.7 and 3.10 illustrates that a higher resolution R, defined as the number of smallest subdivisions along the main axes of the search space, causes a higher number of expanded nodes N. So, N is a function of R.

In paragraph 2.1, it was described that the total number of cubes in an octree representation is dominated by the smallest cubes which are located along the object surface and that this number increases quadratically to the resolution. As the search nodes are defined to be located at the corners of the octree cubes and the number of corners is proportional to the number of cubes, the node complexity for local octree search will be $O(R^2)$. Figs. 3.6 and 3.7 agree with the hypothesis of the dominant expansion of nodes along the surface of the wall.

In quad-lattice search the nodes are constrained to the quadtree division lines in all three quadtree descriptions simultaneously. For a single projection this means that the nodes are exclusively located in planes which are perpendicular to the projection plane and which contain a quadtree division line, see fig. 4.1.

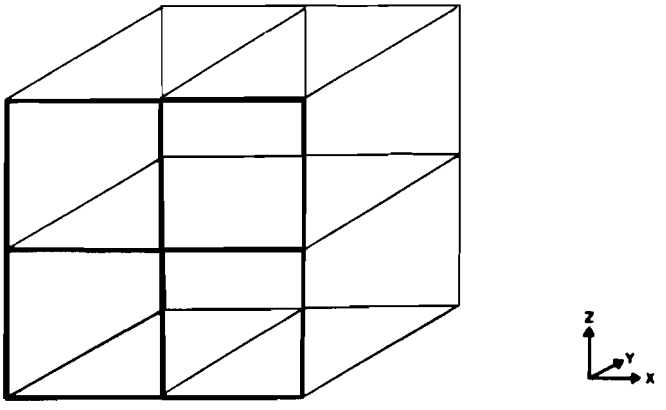


Fig. 4.1 : For a single projection (in front), the nodes for quad-lattice search are located in planes that are perpendicalar to the projection plane and contain a quadtree division line.

When three projections are used, the nodes can only be situated at the intersection points where these planes meet from all the three projections. The number of smallest path segments dominates the total number of expanded nodes, analogous to local octree search. The smallest path segments

are located at the smallest squares along the contours of the projected obstacles. When the resolution increases, only the squares along the contours will split further, thus, introducing extra quadtree division lines. If however, the path does not move along the obstacle contours in all projections simultaneously, the newly introduced nodes can only be located in a plane, perpendicular to the projection in which the path does not move along a contour. This results in a planar expansion requiring an $O(R)$ number of nodes, proportional to the number of squares in a quadtree.
However, the $O(R)$ node complexity is a lower limit. If the three projections contain squares of equal sizes, local octree search expands the same (number of) nodes as quad-lattice search. Near the object edges where the path moves along the obstacle contours in three projections simultaneously, there is locally a octree-like expansion with an $O(R^2)$ node complexity.
Accordingly, the 'best case' complexity for quad-lattice search is $O(R)$, the 'worst case' complexity is $O(R^2)$.

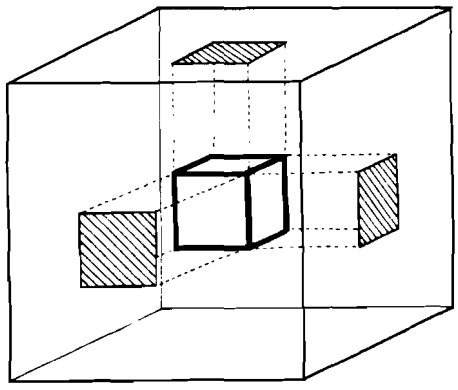
The relation between R and N for either local octree and quad-lattice search can be written as:

$$N = C_n * R^m \qquad (1)$$

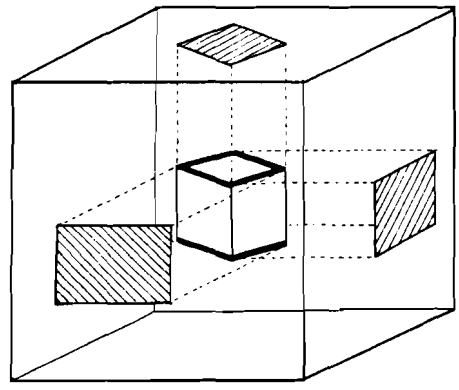
Here, C_n is a constant depending on the obstacle configuration and the position of the start and goal points. In general, the exponent m depends on the dimensions of the search space and search strategy. However, for quad-lattice search it also depends on the obstacle configuration. The exponent m for local octree search is expected to be 2, for quad-lattice search it is expected to be limited between 1 and 2.

An unfavourable obstacle configuration for quad-lattice search will include large areas along the so called *'quadratic object edges'* in the search volume. Object edges are generally called quadratic if both of the adjacent surfaces are perpendicular to a projection plane, or if the object approximates to a plane. Fig. 4.2a shows the 'quadratic object edges' (marked with bold lines) for a cube with all of its surfaces parallel to the projection planes, fig. 4.2b for a cube with only the top and bottom surfaces parallel to the projection planes and fig. 4.2c for a thin wall similar to the obstacle configurations in the figs. 3.6 and 3.9.

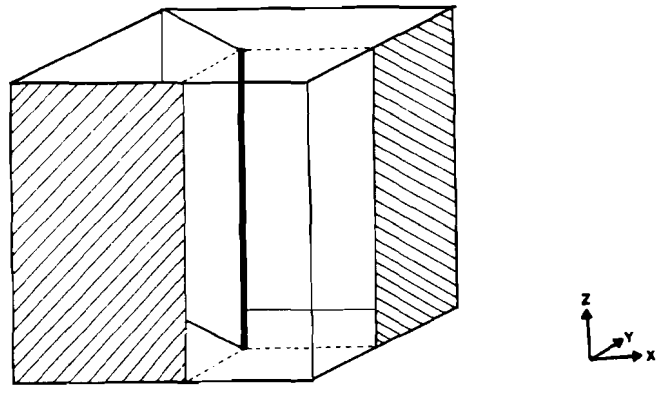
An example of partial quadratic and linear node expansion is shown in figs. 3.9 and 3.10. Along the quadtree division lines, for example in the largest occupied squares in the front and side views, the number of nodes increases linearly with the resolution like the number of smallest squares in the top view. At the vertical edge of the wall, perpendicular to the xz-plane, there is a tendency to expand nodes along the surface of the wall, cf. fig. 4.2c.



(a) A cube with all its surfaces parallel to the projection planes.



(b) A cube with only its top and bottom surfaces parallel to the projection planes.



(c) A thin wall

Fig. 4.2 : 'Quadratic object edges' (bold) in several obstacle configurations.

4.2 TIME COMPLEXITY

For A^* search the computation time T is a function of the number of expanded nodes N. In this approach, the processing time A_t per node is assumed constant, so :

$$T = A_t * N \tag{2}$$

By combining eq. (2) with eq. (1) it follows:

$$T = C_t * R^m \tag{3}$$

where C_t denotes the constant $A_t * C_n$.

The *CDT* runs two masks in passes over all cells of the mono-resolution search space. In a 3-D search space, the total number of cells amounts R^3. A search space without obstacles requires 2 passes to generate the distance transform. In the presence of obstacles, the number of passes will be greater than 2. If an average constant processing time per cell is assumed, the computation time T required for performing the distance transform is:

$$T = p * B_t * R^3 \tag{4}$$

where p denotes the number of passes and B_t the average processing time per cell per pass.

5 EXPERIMENTS

5.1 GOALS

The primary objective described in the introduction was to compare the A^* and *CDT* algorithms with respect to the computation time, the memory requirements and the path length relative to the geometrically shortest path. In chapter 3, an additional criterium was introduced, viz. the number of expanded nodes. The experiments described in this chapter aimed at comparing the criteria mentioned and, moreover, they had to verify the expected polynomial functions described in chapter 4.

In the experiments, the A^* and *CDT* algorithms were implemented in 'C' language on a SUN 3/160 workstation. This workstation contained a 68020 (16 Mhz) processor with 4 MB of working memory. Both algorithms were implemented in 3-D using the chamfer distance metric. In order to save memory space, the integer arithmetic was used for the *CDT* algorithm. In these experiments the $1{:}\sqrt{2}{:}\sqrt{3}$ ratio of the euclidian chamfern metric was approximated by 3:4:5. The experimental object configurations were constructed by means of a CAD modeller.

5.2 EXPANDED NODES

For either of the A^* based local octree and quad-lattice search methods, it follows from eq. (1) :

$$\log(N) = \log(C_n) + m \log(R) \tag{5}$$

with N being the total number of nodes expanded, C_n being a constant depending on the obstacle configuration and location of start and goal point, and R the resolution. On a double logarithmic scale, N becomes a linear function of R with slope m.

Fig. 5.1 gives the total number of expanded nodes N, measured for the obstacle configuration shown in the figs. 3.6 and 3.9. The horizontal axis denotes the level of resolution defined by ${}^2\log(R)$. The marks indicate the measured values. They were connected by straight line segments. As the relationship of eq. (2) was assumed, the best fitting straight line through the data was calculated for comparative purposes. The slope of this line corresponds to the average value of the exponent m. For the local octree curve in fig. 5.1, an exponent with a value of 1.9 was determined which denotes almost a quadratic relationship between between the number of nodes N and the resolution R. For the quad-lattice a value of 1.1 was found which approximates a linear relationship. Accordingly, the influence of a quadratic expansion component is small in this obstacle configuration. It follows from fig. 5.1 that quad-lattice search expands considerably fewer nodes than local octree search in the applied obstacle configuration, varying from a factor of 3 at resolution level 3 up to a factor of 20 at resolution level 7.

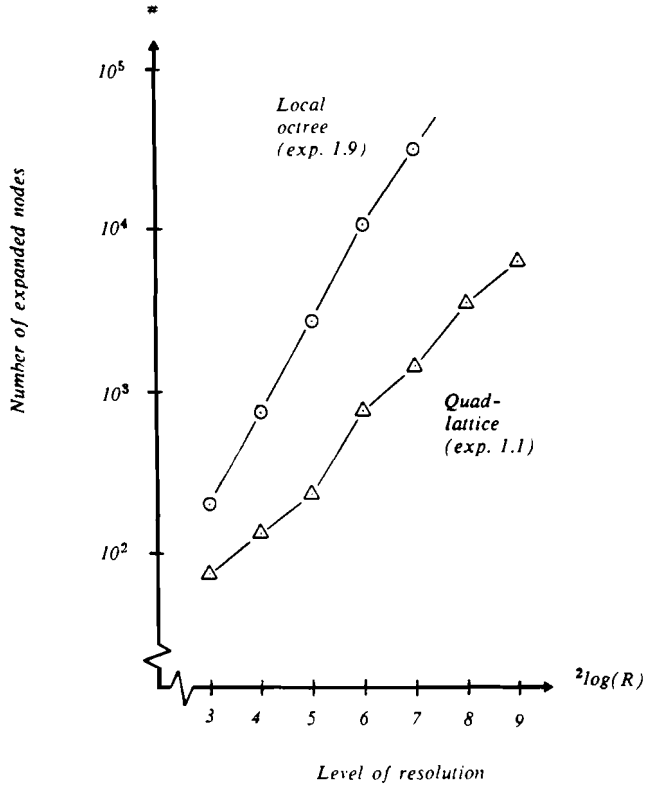


Fig. 5.1 : The number of expanded nodes as a function of the level of resolution for the obstacle configuration of the 'wall'. o indicates the results obtained from local octree search, Δ the results from quad-lattice search.

Experiments have been done in several obstacle configurations. These obstacle configurations were subdivided in two categories. The first category contained the obstacle configurations in which deliberately was attempted to measure a quadratic exponent for quad-lattice search by forcing a path along the 'quadratic obstacle edges'. The obstacle configurations in the second category were supposed to be examples of 'average case' configurations, in which the path was not forced along the 'quadratic object edges'. The experimental obstacle configurations are described in table 5.1.

The linear regression line through the data was calculated according to eq. (5) for all the experimental obstacle configurations. Table 5.2 gives the results for the exponent m, the constant C_n and the correlation coefficient ρ. This correlation coefficient indicates how well the data agrees with the linear regression line. When $|\rho|$ equals 1, the data agrees perfectly, when $|\rho|$ approaches 0 there is no correspondence at all.

Cat.	*Config.*	*Shown in*	*Description*
I	cube	append. 1	A cube with all its surfaces parallel to the projection planes.
	wall_1	append. 2	A path along the 'quadratic edge' of a vertical thin wall.
	sphere	append. 3	A sphere in the center of the search space.
II	wall	figs. 3.6/3.9	A path perpendicular to a vertical wall.
	maze	append. 4	Four vertically placed walls.
	cylinders	append. 5	Five cylinders placed at increasing distances of each other.

Table 5.1 : The description of the experimental obstacle configurations.

Config.		*Local octree search*			*Quad-lattice search*		
		m	C_n	ρ	m	C_n	ρ
I:	cube	2.4	2.4	0.999	1.5	2.5	0.970
	wall_1	2.1	1.0	0.999	1.6	1.0	0.981
	sphere	2.2	0.5	0.992	1.3	3.9	0.979
II:	wall	1.9	3.7	0.999	1.1	6.2	0.996
	maze	2.1	7.6	1.000	1.0	27	1.000
	cylinders	1.6	3.2	0.998	0.9	8.8	0.966

Table 5.2 : The values of the exponent m and constant C_n according to eq. (5) for the experimental obstacle configurations in regard to the number of expanded nodes. ρ denotes the correlation coefficient.

The 'worst case' exponent value of 2 for quad-lattice search was not found for the unfavourable obstacle configurations from the first category. This was due to the fact that the contours of projected obstacles were not entirely described by the smallest available squares in all three projections simultaneously, for example in the configurations of the 'cube' and the 'wall' in which the contours were parallel to the main axes of the projection planes. The contours of the 'sphere' were entirely described by the smallest squares, but the 'quadratic regions' were quite small in which this demand was satisfied for all three projections simultaneously. For the obstacle configurations in the second category, the exponent m approached the value of 1 rather than 2.
For local octree search, the values of the exponent m approximated the expected value of 2 quite well in the obstacle configurations of table 5.1. In the 'cylinder' configuration a relative low value (1.6) of the exponent was determined due to the fact that new passage regions became 'visible' at increasing resolution and so, a considerably shorter path could be found.

The values of the constant C_n for quad-lattice search ranged from 1 to 30 in the experimental obstacle configurations. For local octree search these values ranged from about 0.5 to 10. The obstacle configurations with a large surface area like the 'maze' had relatively a high value of C_n.

The values of the correlation coefficient ρ equalled almost 1 in all experimental obstacle configurations. So, the experiments agreed with the polynomial relation of eq. (1) between the number of expanded nodes N and the resolution R for either local octree and quad-lattice search.

From the experiments described, it can be concluded that quad-lattice search expands considerably fewer nodes than local octree search. The number of nodes approximated the expected quadratic function of the resolution quite well for local octree search. In several cases, the expected lower bound value on the exponent m was found for quad-lattice search which denotes a linear relationship to the resolution. An obstacle configuration confirming the expected worst case exponent value of 2 was not found. A tendency was noticed for the exponent m to approximate to 1 rather than 2.

5.3 COMPUTATION TIME

Analogous to eq. (5), it follows from eq. (3) for either local octree search and quad-lattice search :

$$\log(T) = \log(C_t) + m \log(R) \qquad (6)$$

Fig. 5.2 illustrates the experimentally determined computation times for the A^* based search algorithms and the *CDT*. This relationship has been measured in the obstacle configuration of the 'wall' from the figs. 3.6 and 3.9. The differences in computation time between local octree and quad-lattice search were considerable in this figure, varying from a factor of 10 at resolution level 4, up to a factor of 100 at resolution level 7. A 2.3 value of the exponent m was determined for local octree search, a 1.2 value for quad-lattice search. For the *CDT* algorithm, a 3.0 value of the exponent was calculated according to eq. (4). The *CDT* required less computation time than either local octree and quad-lattice search at low resolution levels. The turn-over point between quad-lattice search and the *CDT* was approximately located at resolution level 4 in fig. 5.2. At this resolution level the computation time was about 1 s.. Local octree search became faster from resolution level 9 when it required about 10^4 s. of computation time.

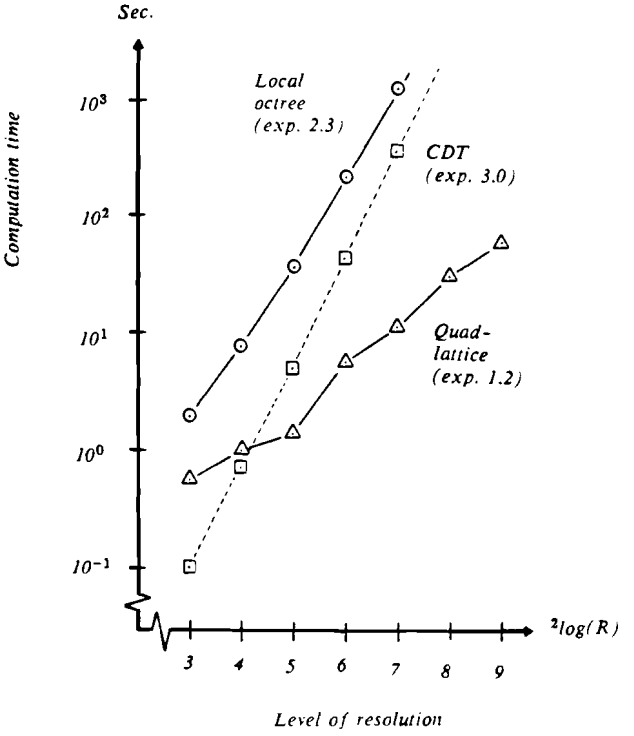


Fig. 5.2 : The experimentally determined computation time for the obstacle configuration of the 'wall'. ○ indicates the results obtained from local octree search, Δ the results from quad-lattice search, □ the results from the CDT.

The computation times of the other experimental obstacle configurations from table 5.1 are shown in the appendices 1 to 5. For these obstacle configurations, the time constant C_t and the exponent m were determined by means of a linear regression line, according to eq. (6). The results are given in table 5.3. This table contains also the correlation coefficient ρ.

Config.	*Local octree search*			*Quad-lattice search*		
	m	C_t *(mS)*	ρ	m	C_t *(mS)*	ρ
I: Cube	2.6	1.9	0.996	1.4	3.9	0.977
wall_1	2.6	3.2	0.997	1.6	7.5	0.980
sphere	2.0	7.8	0.996	1.1	6.7	0.995
II: wall	2.3	1.6	0.998	1.2	3.7	0.990
maze	2.5	1.7	0.998	1.1	8.9	1.000
cylinders	1.9	1.1	0.995	1.0	6.4	0.935

Table 5.3 : The valus of the exponent m and constant C_t for the experimental obstacle configurations, according to eq. (6). ρ denotes the correlation coefficient.

The correlation coefficients ρ in table 5.3 are all almost equal to 1; so, this indicates a high correlation between the data and the regression lines. The values of the exponent m in this table did not accurately correspond with the values that were found for the exponent in table 5.2 regarding the number of expanded nodes. The differences between the values of the time and node exponent were smaller for quad-lattice search (< 0.2) than for local octree search (< 0.5). These differences between the exponents were caused by an increasing average processing time per node at higher resolution levels. In fact, the number of candidate nodes for expansion increased and therefore, more time was required on average, for selecting the best node from the collection *OPEN*.

To illustrate the magnitude of the increase, some experimental results are given that were obtained from the the obstacle configuration of the 'wall' that was shown in the figs. 3.6. and 3.9. For local octree search about 20% (9 ms. total average processing time per node) of the computation time was spent on node selection at low resolution levels. For higher resolution levels, this percentage increased to about 80% (20 ms.). For the more restrictively expanding quad-lattice search, the percentages varied from 10% (6 ms. average processing time per node) at low resolution levels up to 20% (7 ms.) at higher resolution levels.

Accordingly, the assumption of a constant processing time per node, as expressed by eq. (2), was not very accurate for local octree search at high levels of resolution. For quad-lattice search, the results agreed better with eq. (3). For practical purposes, the relationship of eq. (3) will do.

Table 5.4 shows the results that were obtained from the *CDT* algorithm. This table contains the experimentally determined values of the exponent (*exp.*), the number of passes p, the average processing time per cell per pass B_t and the correlation coefficient ρ, according to eq. (4).

Config.	*exp.*	p	B_t (μS)	ρ
cube	3.0	4	34	1.000
wall_1	3.0	6	35	1.000
sphere	3.0	4	34	1.000
wall	3.0	6	35	1.000
maze	3.0	8	35	1.000
cylinders	3.0	4	40	1.000

Table 5.4 : The results obtained by application of the CDT algorithm for the experimental obstacle configurations. p denotes the number of passes required, B_t the average processing time per cell per pass and ρ the correlation coefficient.

In table 5.4, there is a very high correlation between the data and the regression line in all the experimental obstacle configurations. The values of the exponent and the average processing time B_t, respectively 3 and 35 μs,

were constant in all the experimental obstacle configurations. Therefore, it can be concluded that the experimental results agreed with eq. (4).

An 'unfavourable' obstacle configuration for the *CDT* is characterized by many passes. In general such a configuration contains free space regions which are largely enclosed by obstacles, for example the obstacle configuration of a maze. However, this was not investigated any further. In these experiments 4 to 8 passes were sufficient to generate the distance image.

Due to the large cell/node processing time ratio of 1:200 to 300 and the differences between the exponents of the A^* based methods and the *CDT*, there was a turn-over point for these algorithms. At low resolution levels the *CDT* was superior to the A^* based methods in computation time, at higher resolution levels this relationship was reverse. Table 5.5 shows at what resolution levels the turn-over points were located for local octree search and quad-lattice search vs. the *CDT*. These turn-over points were calculated by intersecting the regression lines with the parameters from tables 5.3 and 5.4.

Config.	*Local octree/CDT*	*Quad-lattice/CDT*
cube	10.2	4.8
wall_1	10.1	6.2
sphere	6.2	4.9
wall	9.2	4.2
maze	13.3	4.7
cylinders	5.4	4.2

Table 5.5 : The resolution levels of the turn-over points for local octree search and quad-lattice search vs. the CDT in regard to the computation time.

The turn-over points for quad-lattice search vs. the *CDT* in this table are generally located at resolution level 4 to 5 in table 5.5. For local octree search the location of the turn-over points ranges more widely, namely from level 6 to 13.

5.4 MEMORY USAGE

A^* required the temporary storage of a search tree that contained the expanded nodes (in *CLOSED*) and the candidate nodes for expansion (in *OPEN*). The *CDT* used a fixed amount of memory i.e R^3 cells for a 3-D search space with resolution R. A *CDT* cell required considerably less memory space than an A^* node; in these experiments, a cell was stored in 2 bytes, a search node in 48 bytes, that is a 1:24 ratio.

For the obstacle configuration of the 'wall' that was shown in the figs. 3.6 and 3.9, the total amount of required memory is illustrated in fig. 5.5. In this obstacle configuration, the *CDT* used less memory than quad-lattice search up to resolution level 4. From resolution level 6, local octree search required less

memory space than the *CDT*. Quad-lattice search is considerably more efficient in memory usage than local octree search.

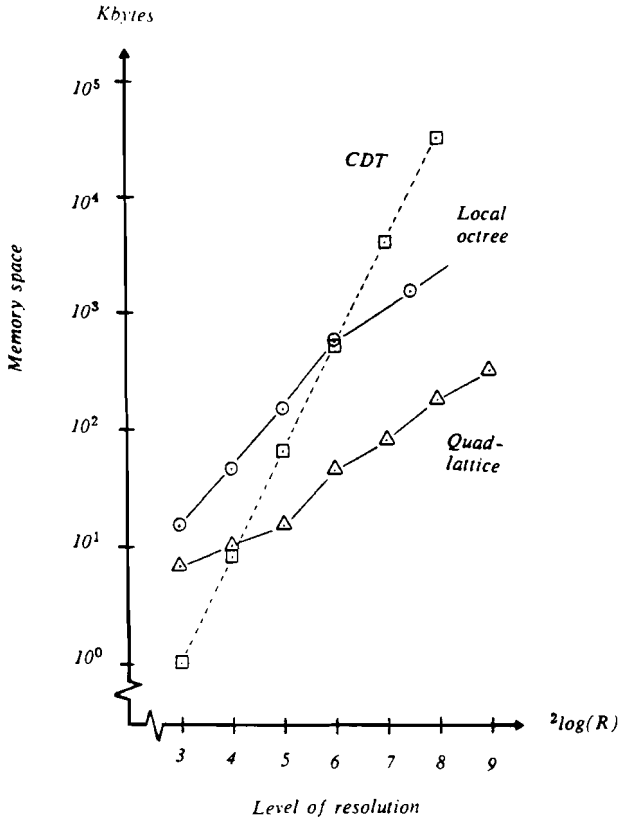


Fig. 5.5 : The total memory required for the obstacle configuration of the 'wall'. o indicates the results obtained by local octree search, Δ the results from quad-lattice search, □ the results from the CDT.

The results from the memory requirements in the other experimental obstacle configurations from table 5.1 are given in the appendices 1 to 5. In these obstacle configurations, the number of expanded nodes dominated the number of '*OPEN*' nodes. This dominancy was explained by the fact that the expanded nodes were all located in a search *volume*, whereas the '*OPEN*' nodes were located at the *surface* of that volume. Due to this dominancy, the total memory space required was approximately proportional to the number of expanded nodes. The large cell/node memory space ratio made the *CDT* to be preferred to either quad-lattice search and local octree search at low resolution levels. At higer resolution levels, the exponenential part in eq. (1) made local octree search and quad-lattice search to superior to the *CDT*.

The resolution levels of the turn-over points in regard to the memory space are given in table 5.6 for the experimental obstacle configurations.

Config.	*Local octree/CDT*	*Quad-lattice/CDT*
cube	3.8	3.8
wall_1	5.3	4.0
sphere	5.5	4.8
wall	6.0	4.1
maze	7.5	4.7
cylinders	4.6	3.9

Table 5.6 : The resolution levels of the turn-over points for local octree search and quad-lattice search vs. the CDT in regard to the required memory space.

In this table, the turn-over points for quad-lattice search vs. the *CDT* are located approximately at resolution level 4 to 5; for local octree search vs. the *CDT*, the locations range from resolution level 4 to 8.

For practical purposes, it is necessary to apply an upper limit to the memory required for both of the A^* based search strategies. For local octree search the total number of cubes in the octree description of the search space is the upper limit. However this number is not explicitly available from the quadtree projections. The memory for quad-lattice search is definitely limited by the number of octree cubes; however, a more accurate estimation needs to be found.

5.5 OPTIMAL DISTANCE

Fig. 5.6 shows an obstacle configuration which illustrates the different paths found by multi-resolution local octree search (dashed lines), quad-lattice search (dots and dashes) and mono-resolution *CDT* search (dotted lines). The relative increase of path length compared to the geometrically shortest path is given as a function of the level of resolution in fig. 5.7.

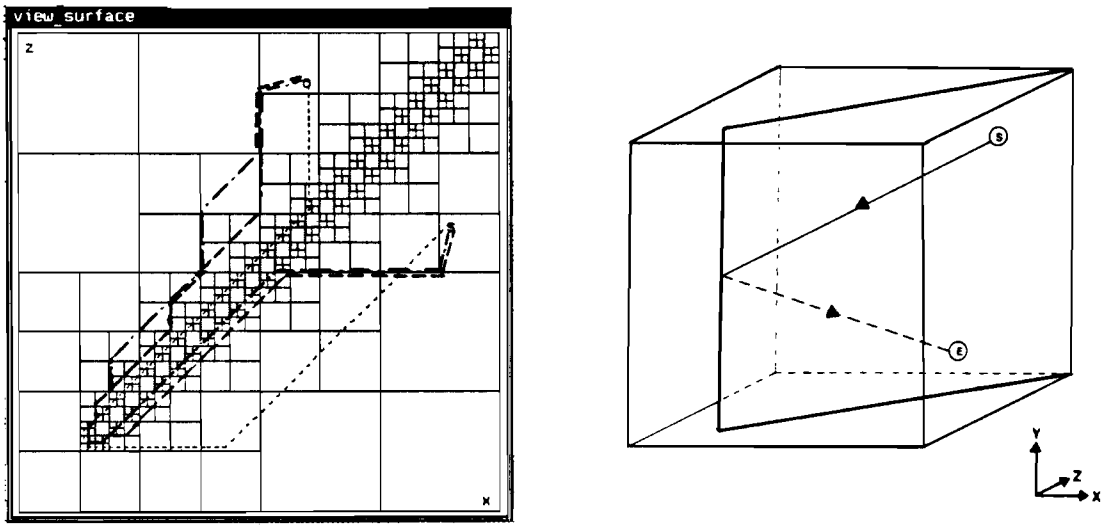
In the obstacle configuration of fig 5.6, the *CDT* found the shortest path of all three search techniques on every resolution level. Quad-lattice search produced shorter paths than local octree search. There is hardly a decrease in path length from resolution levels approximately greater than 6.

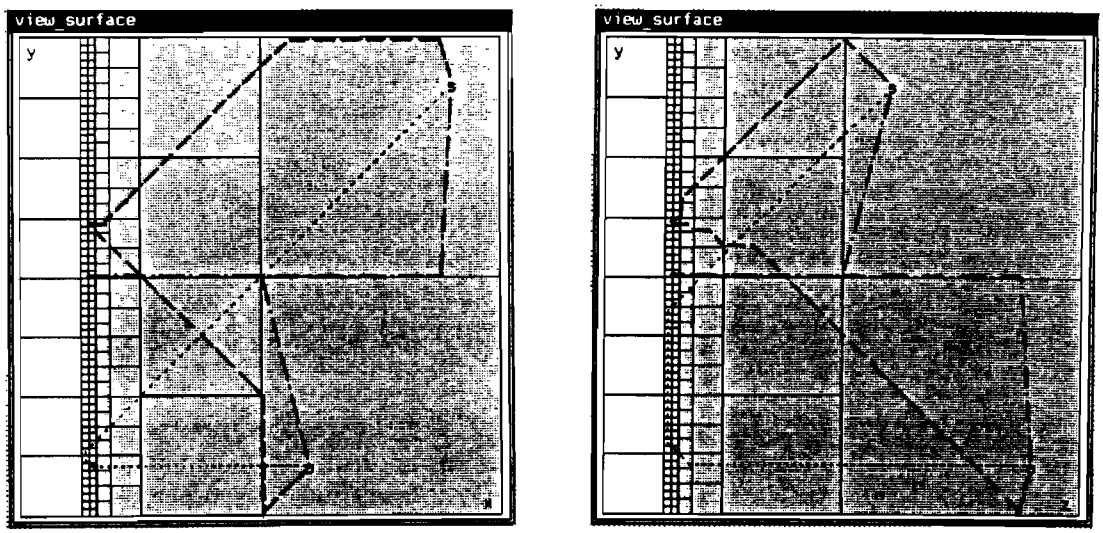
In general, the length of the solution paths compared to the geometrically shortest paths depended on five factors :

(1) **the resolution.** In low resolution search spaces, the freedom of motion was limited due to the coarse quantization of the space. Besides, the obstacles tended to grow because partially occupied cubes of the smallest available size were considered as fully occupied to prevent collisions. This made the path to pass more widely around the obstacle surfaces.
(2) **the metric.** Borgefors [BOR84] showed that the length of the calculated path could be overestimated at most 14.7 % compared to the geometrically shortest path if the euclidian chamfer metric is used. For the chamfer

(3,4,5) metric as used here for the *CDT* algorithm, this percentage amounted to 11.8 %.

top view





front view *side view*

Fig. 5.6 : The different paths found by local octree search (dashed lines), quad-lattice search (dots and dashes) and the CDT (dotted lines).

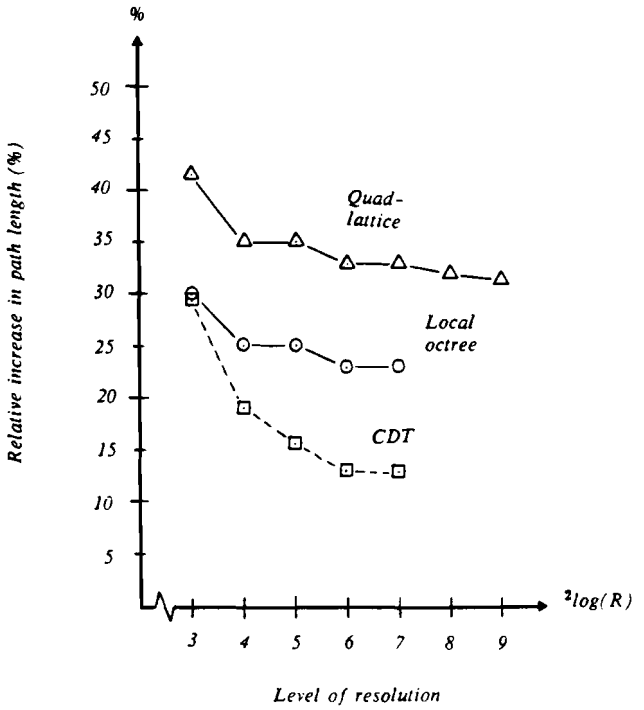


Fig. 5.7 : The relative increase in path length, compared to the geometrically shortest path. o indicates the results obtained from local octree search, Δ the results from quad-lattice search, □ the results from the CDT.

(3) **the constraints on the permissible path segments.** The A^* based algorithms used a multi-resolution search space representation. The permissible path segments in such a representation were more constrained than in a mono-resolution representation as used by the *CDT*. These constraints resulted in a rather meandering path with greater length than the mono-resolution path. As quad-lattice expansion strategy was more constrained than the local octree strategy, its paths could be longer than those produced by local octree search.

(4) **path segments along the corners or along the centres of the cubes.** As the A^* path moved along the corners of the cubes, it could pass more closely along the obstacles than a *CDT* path which moved from cube centre to cube centre.

(5) **connection of the start and goal points with the search nodes.** The start and goal point were connected with straight path segment to the nearest cube corners (in case of A^*) or cube center (in case of the *CDT*). These path segments could move in arbitrary directions and they were not constrained to the chamfer distance metric. So, in a multi-resolution search space with large start or goal cubes, the path could be shorter in length than the mono-resolution path if any of the cube corners was located into the direction of the 'optimal' path.

The importance of each of these factors depended on the specific obstacle configuration.

The path lengths in the experimental obstacle configurations from table 5.1 are shown in the appendices 1 to 5. From these results, it was observed that at increasing resolution, the decrease of the path length was greater for the *CDT* than for both of the A^* based methods. This could be explained by the fact that the effect of an increasing resolution in a multi-resolution search space is only local, namely at the surface of the obstacles only; the remaining parts of the search space were not subdivided any further. At the resolution levels 5 to 6 and higher, the *CDT* path was generally shorter than the A^* paths. In all of the experimental obstacle configurations, the quad-lattice path was longer than, or at most as long as the local octree path. At resolution levels higher than 6, the decrease in path lenth was marginal in nearly all the experimental obstacle configurations, except for the 'cylinders' configuration. In this obstacle configuration, a small passage way became 'visible' at resolution level 7. This caused a considerably shorter path.

5.6 CONCLUSIONS FROM THE EXPERIMENTS

Experiments have been done with the *CDT* algorithm and both the A^* based local octree and quad-lattice search algorithms. From these experiments it followed that:

* Local octree search had a node and memory complexity of approximately $O(R^2)$. In several experiments, the node and memory complexity of quad-lattice search approached rather $O(R)$ than $O(R^2)$.
* The computation time for quad-lattice search was aproximately proportional to the number of expanded nodes. For local octree search, this proportional relationship deteriorated at high resolution levels; then, the exponent tended to increase slightly. It can be concluded that quad-lattice search incorporates a clear improvement in computation time and memory requirements compared to local octree search, at the expense of only a slight increase in path length.
* The *CDT* algorithm had a time and memory complexity of $O(R^3)$. Despite its higher complexity compared to A^* search, the *CDT* algorithm required less memory space and computation time at low resolution levels, due to the large overhead to be kept by A^*. At relative high levels of resolution, in these experiments approximately level 7 and higher, local octree search became superior to the *CDT* algorithm with regard to computation time and memory usage. Quad-lattice search became to be preferred at relatively lower levels of resolution, in these experiments approximately at level 4-5.
* The decrease of the path length at an increasing resolution is greater for a path found in a mono-resolution search space as used by the *CDT* than in a multi-resolution search space as used by A^*. So, at relative high levels of resolution, the *CDT* could find shorter paths than both of the A^* based search methods. In the absence of tiny 'critical' free passage regions, the decrease of the path length was found to be marginal at resolution levels higher than approximately level 6.

6. EXTENSIONS

In this chapter, some extensions are described that were tried in order to improve the performance and applicability of the orthogonal projection approach, namely, the optimization of various criteria (par. 5.1) and an acceleration using A^*_ϵ, a semi-admissible variation on A^* (par. 5.2).

6.1 VARIOUS OPTIMIZATION CRITERIA

In the experiments described in the previous chapters, only the distance was optimized. However, for robot applications, this single criterium may not give optimal results. The geometrically shortest path moves often close to the object surface and it is therefore very susceptible to model inaccuracies and positioning errors. To minimize the positioning errors, the movements have to be performed at low speed for preference. For robot assembly operations, a minimal time optimization is usually preferred. Formally, this implies extending the path planning algorithm with the dimensions of velocity and acceleration for all degrees of freedom. For robots with a few degrees of freedom, planning with such a complexity can hardly be handled.

If a sub-optimal solution is allowed, the dimensionality of this problem can be reduced by splitting it into a separate topological optimization viz. the actual path planning, and a dynamic optimization or trajectory planning, related to optimization of velocities and accelerations of the robot links. For the benefit of the trajectory planning, additional criteria can be included in the optimization function. In this case, these criteria are topological characteristics that were assumed to be favourable for high speed motion. Experiments were done with two additional criteria:

(1) **Sharpness of angles between path segments.** Long straight path segments can usually be performed at high speed. Corners in the path cause decelerations.
(2) **Proximity to obstacles.** As argued above, motions near obstacles have to be accurate and are therefore preferred to be performed at a certain distance from the objects.

These two criteria were included in the evaluation function f of the A^* algorithm, i.e. in the g component, representing the actual cost of the path from the startnode to the current node, cf. [KAM86]. The new g for node n was now defined as:

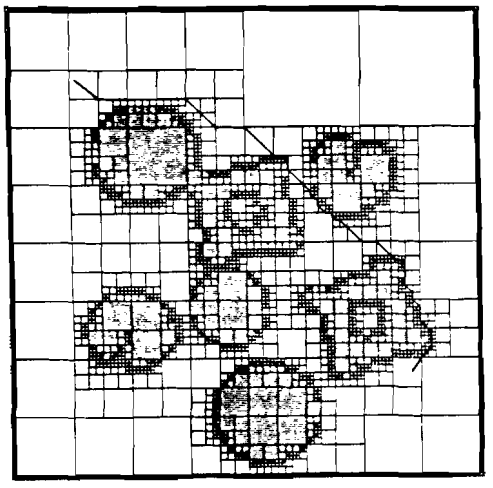
$$g(n) = g(n') + \tilde{g}(n',n) \tag{7}$$

where g(n') is the cost from the startnode to n's predecessor n', and $\tilde{g}(n',n)$ the incremental cost of the path segment between n' and n. This function was defined as:

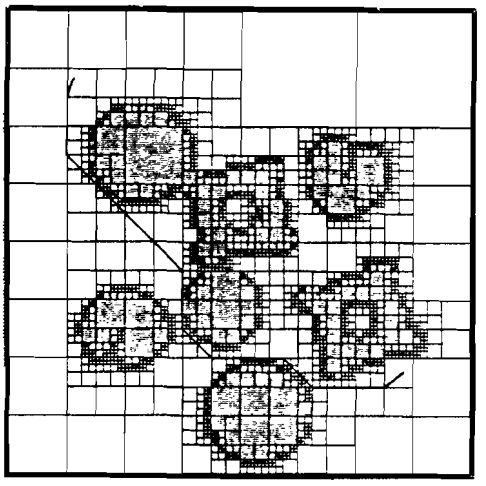
$$\tilde{g}(n',n) = D(n',n) + \alpha\,(1-\cos(s_{n'},s_n)) + \beta\,P(|s_n|) \tag{8}$$

with D(n',n) representing the euclidian chamfer distance between n' and n. The second term on the right represents the corner component. s_n and $s_{n'}$ are the path segments by which node n and n' are respectively reached from their predecessors n' and n'', α is a weighting factor. Note that straight path

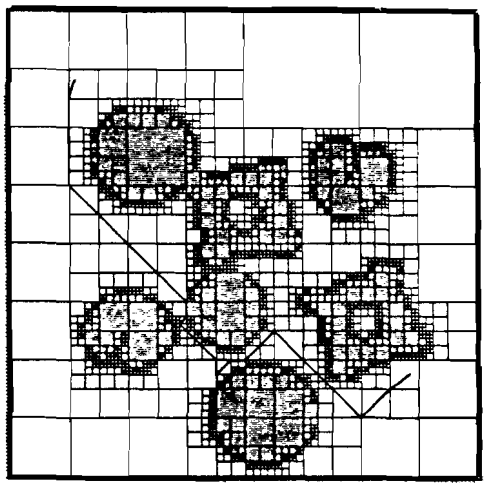
segments do not increase of the corner component of the cost function g. The most right handed term is an approximation for the proximity of obstacles. As the smallest path segments are located at the obstacle boundaries, the factor $|s_n|$ which is the length of path segment s_n, was used as an argument for the proximity penalty function P. This function was chosen to have a value 1 for the smallest path segments and decreased its value by a factor 4 for every power of 2 increase of $|s_n|$. The factor β was used for weighting. The result of including these additional optimization criteria is shown in fig. 5.1 for a 2-D quadtree search space representation.



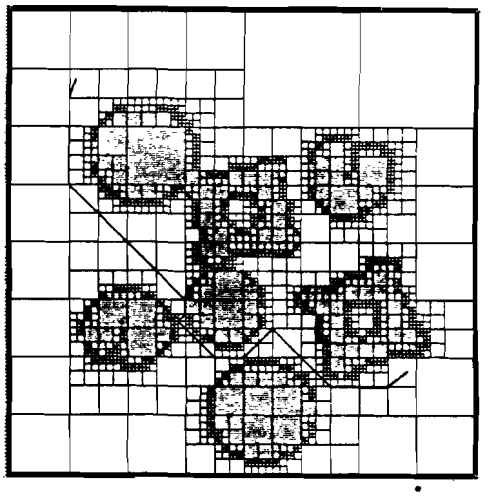
(a) shortest distance
($\alpha = \beta = 0$)



(b) including minimal number of corners
($\alpha = 0.26$; $\beta = 0$)



(c) including proximity
($\alpha = 0$; $\beta = 0.25$)



(d) including corners and proximity
($\alpha = 0.50$; $\beta = 0.10$)

Fig. 6.1 : The paths optimized to various criteria in a 2-D quadtree coded search space.

Fig. 6.1a shows the minimum distance path ($\alpha = \beta = 0$) according to the euclidian quadtree chamfer metric. (This implies that the path segments have to pass through two different corners of at least one of the associated quadtree squares.) There was a tendency to move along the obstacle boundaries. In fig. 6.1b, the corner component has been included in the optimization function ($\alpha = 0.26$; $\beta = 0$). The distance increased slightly in exchange for a lower number of corners in the path. In fig. 6.1c, the proximity criterium ($\alpha = 0$; $\beta = 0.25$) has been included only. There was a tendency to stay away from small squares, which describe the obstacle boundaries and, thus, a path was followed somewhere in between the surrounding obstacle boundaries. As the sizes of the boundary squares were not always minimal, the path was not guaranteed to stay away from the boundaries. However, differently sized blocks could cause acute changes in the direction of the path. Finally in fig. 6.1d, both the corner and the proximity criterium have been included ($\alpha = 0.50$; $\beta = 0.10$). In comparison with figure 6.1b, the path tended to be further away from the obstacles. Compared with figure 6.1c, certain acute angles have been eliminated.

For time optimization, the proximity information could be used in addition to define a volume of free space around the path. All points inside this volume are guaranteed to be collision free. The diameter of the volume can vary locally. A trajectory planner or path-smoother which smoothes 'useless' corners due to a local low resolution in the octree/quadtree description and can optimize the trajectory within the free volume further.

6.2 ACCELERATED SEARCH BY A^*_ϵ

During each iteration of the A^* algorithm, the best node with the lowest evaluation function value was selected for expansion. This selection required scanning and (partially) sorting of all the candidate nodes (in *OPEN*). At higher resolution levels, an increasingly number of nodes was stored in *OPEN*. As it was already indicated in paragraph 5.2, on average, there was relatively more time spent per node on selection at higher resolution levels than at lower levels.

As a safeguard against the increasing processing time per expanded node, A^*_ϵ (said as A-epsilon-star) was applied, see Pearl [PEA82][PEA84]. A^*_ϵ is a semi-admissible variation of the A^* algorithm, searching for a sub-optimal solution path. A^*_ϵ does not only select the best node for every iteration, but it also selects all the nodes that do not exceed the cost of the best node by more than a factor $1+\epsilon$ ($\epsilon \geq 0$). Pearl [PEA82] showed that the solution path found is ϵ-admissible. It means that A^*_ϵ is guaranteed to find at least a sub-optimal solution that does not exceed the cost C^* of the optimal path by a factor greater than $1+\epsilon$. During each iteration, the A^*_ϵ algorithm selects a set of nodes with sub-optimal values of the evaluation function f, instead of only the node with the lowest f value. This results in a reduced number of iterations, thus making fewer calls on the selection mechanism. So, the time spent on node selection will be reduced. However, the number of expanded nodes can increase as A^*_ϵ expands all the nodes n statisfying $f(n) \leq (1+\epsilon)C^*$ instead of $f(n) \leq C^*$ for A^*. The increase can be greater at greater values of ϵ.

To illustrate the impact of the A^*_ϵ algorithm, some results are given that were obtained from the obstacle configuration of figs. 3.6 and 3.9. For ϵ equal to 0.5 divided by the number of divisions along the main axes, about 5% of the computation time was spent on selection for either quad-lattice (6 ms. total average processing time per node) and local octree search (7 ms.). This percentage depended to some extent on the level of resolution. Besides, a 50% increase of the number of expanded nodes by A^*_ϵ was noticed compared with A^*, at low resolution levels. It decreased to 5% more at higher resolution levels.

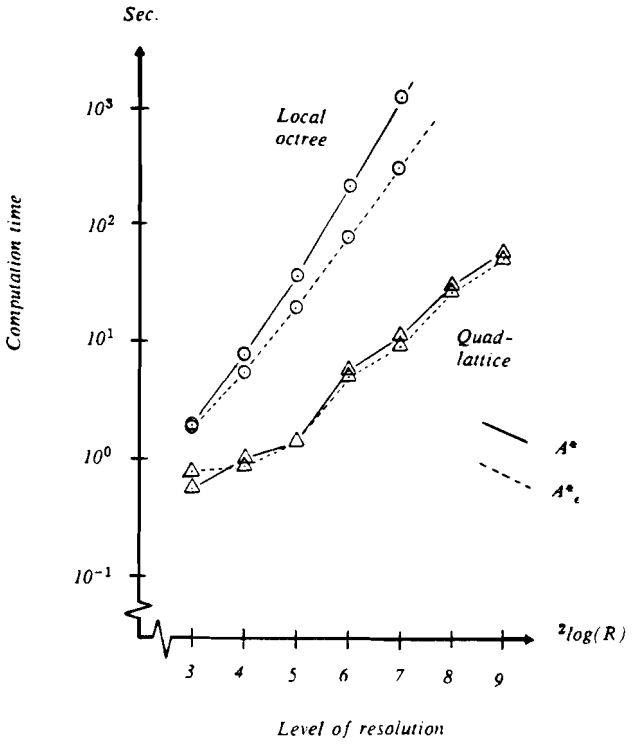


Fig. 6.2 : Computation time required for A^* and A^*_ϵ in the obstacle configuration of the 'wall'.

Fig. 6.2 shows the computation time experimentally determined for either A^* search (solid line) and A^*_ϵ (dashed line). The savings determined for local octree amounted to 6% at low resolution levels, up to 60% at high resolution levels. For quad-lattice search 30% more time was required for A^*_ϵ than for A^*, at the lowest level of resolution, due to the increased number of expanded nodes by A^*_ϵ. At higher resolution levels, A^*_ϵ gradually required up to 20% less time.

It can be concluded that A^*_ϵ was usually a time saving variation of A^* for small values of ϵ, at the expense of an increase of memory size. The savings in computation time were greater for high resolution levels than for low resolution levels and for local octree search greater rather than for quad lattice search. In general, A^*_ϵ was favourable to A^* for applications where a large number of slightly differing solution paths existed, cf. [PEA82].

7. FURTHER IMPROVEMENTS

In this chapter, some topics are described for further improving the performance and applicability of the path planning algorithms. These improvements were not implemented.

7.1 HIERARCHICAL SEARCH

From the experimental results, it can be seen that limiting the resolution of the quadtree descriptions, accelerates the search, while the path length decreases a little with further increase of the resolution. Moreover, with an increased resolution, the path tends to pass closer to the obstacles with more risk of undesired collision due to the inaccuracy in the execution of the movements. Therefore, it is desirable to search for a path at a fairly low level of resolution and to extend the search to higher levels of resolution omly if no path can be found.

Brooks and Lozano-Pérez [BRO83] suggested the following search strategy:

a- Start the path search at a rather low level of resolution, while keeping the mixed areas inaccessable for passing. The mixed areas correspond to the grey nodes in the quadtree/octree description at the maximum allowed depth.

b- If no path can be found from start to goal points, repeat the path search. However, this time allow the grey nodes to be passed. If a path can be found, expand the appropriate grey nodes and search for a path through them.

c- If any of the expanded grey nodes turn out to be impassable, the search has to be repeated with this grey node made inaccessible.

However, this strategy turned out to be very expensive computationally. Kambhampati and Davis [KAM86] improved this strategy by including the grey nodes directly in the search process; however, at a high cost to the evaluation function f of the A^* algorithm. After a path had been found, the areas corresponding to the grey nodes had to be checked whether they could safely be crossed. Thus, less fruitless effort would be spent on details during the search process.

A second method of hierarchical search is applying a hierachy of search techniques, as described by Herman [HER86]. In his solution, search started with a simple algorithm such as *hypothesize and test*, where simple solutions like linear or circular paths from start to goal were tested. If this method was not successful, hill climbing had to be tried. Once this method got stuck in a local minimum, the computationally more expensive A^* technique was used. The disadvantage of the simple hill climbing strategy was that it could not guaranteed to find the shortest possible path (*non-admissible*).

7.2 EXTENSIONS TO THE CDT ALGORITHM

In this paragraph, some suggestions will be given for extending the *CDT* algorithm.

Optimization of the path with additional criteria, such as proximity to obstacles, requires a separate application of the *CDT* algorithm for each

individual criterium. The separate transforms may be performed in parallel, but they require storage of the different images. The optimization function may be an arbitrary function of the criteria. However, this function may have several local minima in the associated images. *Irrevocable* search strategies such as hill climbing (steepest descent) can not be applied for tracing the optimal path, because they can get stuck in a local mimimum. More *tentative* search strategies are required, incorporating backtracking facilities in order to continue from local minima. A^* for example, could very well be used because the distance images provide a highly informative heuristic function, resulting in a very efficient search.

The *CDT* algorithm can offer several opportunities for accelerating the search. Some suggestions are:

* **hardware implementation**, due to the simple operations required.
* **parallel processing**; the forward and backward masks W and W' from fig. 2.6, can be run in parallel over the images.
* **application in a subspace**; if a subspace could be determined which is assumed to enclose the solution path, using some rule of thumb, the algorithm could be applied only within this subspace. Thus, a lot of effort can be saved by not running the masks in the entire space. Indeed the difficulty is knowing to define an appropriate rule of thumb.

7.3 ACQUISITION OF THE PROJECTIONS

7.3.1 PROJECTIONS FROM NON-ORTHOGONAL PLANES

Orthogonal projections can easily be obtained from a workspace described by a CAD modeller regardless of the desired viewing directions. However, when TV cameras are used to obtain the projections, the orthogonal view angles and the scaling, due to the perspective, may be more difficult to achieve. For projections taken from non-orthogonal view angles, two methods are described below.

The first method is to reconstruct a maximum volume from the non-orthogonal projection planes in a similar way as was indicated in par. 3.1. The larger the number of projections taken, the closer the approximations of the true objects. Then, the maximum volumes can be projected on to the desired orthogonal projection planes.

Another approach is to allow the search algorithm to use a non-orthogonal coordinate frame with the three viewing directions along the main axes. Then, the collision test can be performed and the length of the path segments can be calculated in the same way as for orthogonal images. However, additional scaling of the images is required.

In fig. 7.1 a non-orthogonal coordinate frame in a 2-D space is illustrated. The 2-D workspace is projected on to the non-orthogonal main axes v and w. The viewing directions are perpendicular to the main axes.

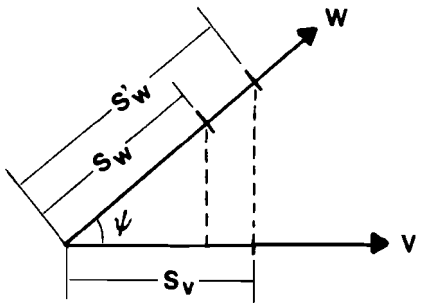


Fig. 7.1 : A non-orthogonal projection frame in a 2-D space.

For strictly 2-D points that are located at some multiple of the unity steps in v and w directions respectively s_v and s_w in fig. 7.1, these steps must have different lengths. If not, a movement over a multiple of s_w results in general in a corresponding 'intermediate' position on the v-axis, which is not a multiple of the unity s_v step. It cannot be asssumed that the steps s_v and s_w are the smallest possible unit steps.

If both images have the same scale originally, they can be matched by scaling the pixels of the w-image with a factor $1/\cos\psi$, where ψ is the angle between the two projection axes (or view angles). The new stepsize is indicated by s'_w in fig. 7.1. The orthogonal situation is special in the sense that movements in v and w directions are independent. Then, no additional scaling is required.

The method described here for 2-D, can be used generally for 3-D with an appropriate scaling of the image planes in width and height.

7.3.2 PERSPECTIVE PROJECTIONS

TV cameras provide a perspective view of the workspace. Strictly speaking, parallel projections are required. Perspective projections cause obstacles in the foreground to appear larger in the image plane than when these obstacles are placed in the background. This is critical when accurate dimensions of the obstacles are required.

The relative size in the image planes of objects placed at different positions in the workspace is called the *perspective distortion*. It can be shown that the perspective distortion is inversely proportional to the corresponding object distance from the optical centre of the camera. For minimizing the perspective distortion, the camera has to be positioned far away from the workspace. The use of zoom lenses with variable focal distances does not provide any relief.

A way to eliminate the perspective is to construct a 3-D voxel-based space representation from the perspective projections is by tracing rays in the viewing direction, starting from the image planes. Orthogonal parallel projections can be obtained by projecting the 3-D workspace on the desired projection planes.

8. CONCLUSIONS

Two different types of algorithms for 3-D robot path planning were evaluated: the A^* heuristic search algorithm and the Constrained Distance Transform algorithm (*CDT*). The A^* algorithm used three orthogonal views of the workspace only, without reconstruction of a full 3-D representation of the search space has to be performed. A quadtree encoding of these views enabled multi-resolution path planning. Two techniques for multi-resolution path planning have been presented: *local octree search* and *quad-lattice search.* Quad-lattice search incorporates a newly developed search space representation.

In the experiments it was observed that local octree search has time and memory complexity of approximately $O(R^2)$, where R was the resolution, i.e. the number of smallest subdivisions along the main axes of the workspace. Quad-lattice search had a time and memory complexity less than $O(R^2)$. The experiments suggested that the complexity approaches rather $O(R)$ than $O(R^2)$ for quad-lattice search in most experimental obstacle configurations. So, quad-lattice search clearly reduces the computation time the and memory usage compared with local octree search; however, at the expense of a slight increase in path length.

The *CDT* algorithm operated in a mono-resolution search space and therefore it had $O(R^3)$ time and memory complexity. However, the characteristic operations required for this algorithm were considerably faster than those for the A^* algorithm. It was shown that quad-lattice search is superior to the *CDT* in the experimental obstacle configurations, with regard to computation time and memory space in workspaces with a resolution approximately greater than 16. At the turn-over point, the algorithms required a few seconds of computation time. Local octree search became superior to the *CDT* only in high resolution workspaces (approx. 64 and greater). At the turn-over point, the algorithms required at least a few hundred seconds of computation time. The *CDT* found shorter paths than both of the A^* based search methods at increasing resolution.

Minimum distance paths were often less suited for robot applications. They passed closely along surfaces of the objects and therefore they were very susceptible to model inaccuracy and manipulator position errors. More suitable paths could be found by including additional criteria in the optimization function. such as the proximity to obstacles and the sharpness of angles in the path. In contrast to the A^* algorithm, the *CDT* algorithm required a large amount of additional memory for this improvement.

The A^* algorithm could be accelerated by using semi-admissible variations, searching for a sub-optimal path. It was shown that the A^*_ϵ algorithm improved the computation time by a factor up to 2, at the expense of slightly increased memory size. Suggestions were made for improving the speed of the *CDT* algorithm. It is very suitable for hardware implementation. Also, a more intelligent evaluation than in purely a breadth-first fashion could lead to considerable savings in computation time.

It was shown that it is possible to incorporate views taken from non-orthogonal view angles in the projection approach. If TV camera images were directly used, accurate positions were limited by perspective distortion.

REFERENCES

[BOR84] Borgefors G.,"Distance transformations in arbitrary dimensions", *Computer Vision Graphics and Image processing*, 27, 1984, pp321-345.

[BOR86] Borgefors G., "Distance transformations in digital images", *Computer Vision Graphics and Image processing*, 34, 1986, pp344-371.

[BRO83] Brooks R.A. and Lozano-Pérez T., "A subdivision algorithm in configuration space for findpath with rotation", *Proc. Eighth Int. Joint Conf. Artificial Intelligence*, 1983, pp799-806.

[CHI86] Chien C.H. and Aggerwal J.K., "Identification of 3D objects from multiple silhouettes using quadtrees/octrees", *Computer Vision Graphics and Image Processing*, 36, 1986, pp256-273.

[DOR86] Dorst L. and Verbeek P.W., "The constrained distances transformation: a pseudo-euclidian, recursive implementation of the Lee-algorithm", *Proceedings Eusipco-86*, Sept. 2-5, the Hague, The Netherlands, 1986, pp917-920.

[HER86] Herman M., "Fast planning in unstructured dynamic, 3-D worlds", *Proc. of SPIE*, vol. 635, Applications of Artificial Intelligence III, 1986, pp505-512.

[HUN79] Hunter G.M. and Steiglitz K., "Operations on images using quad trees", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, 1979, pp145-153.

[JAC80] Jackins C.L and Tanimoto S.L., "Oct-trees and their use in representing three-dimensional objects", *Computer Vision and Image Processing*, 14, 1980, pp249-270.

[KAM86] Kambhampati S. and Davis L.S., "Multiresolution path planning for mobile robots", *IEEE J. of Robotics and Automation*, vol RA-2, 1986, pp135-145.

[LEE61] Lee C.J., "An algorithm for path connections and its applications", *IRE Transactions on Electronic Computers*, Sept., 1961, pp346-365.

[MAR77] Martelli A., "On the complexity of admissible search algorithms", *Artificial Intelligence*, 8, 1977, pp1-13.

[NIL80] Nilsson N.J., "Principles of Artificial Intelligence", Palo Alto, California: Tioga Publishing Company, 1980.

[PEA82] Pearl J. and Kim J.H., "Studies in semi-admissible heuristics", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-4, no. 4, july, 1982, pp392-399.

[PEA84] Pearl J., "Heuristics: Intelligent Search Strategies for Computer Problem Solving", Reading, Mass.: Addison Wesley Publishing Company, 1984.

[SAM84] Samet H., "The quadtree and related hierarchical data structures", *Computing Surveys*, Vol 16, 1984, pp187-260.

[WON86] Wong E.K. and Fu K.S., "A hierarchical orthogonal space approach to three-dimensional path planning", *IEEE J. of Robotics and Automation*, Vol RA-2, 1986, pp42-53.

APPENDICES

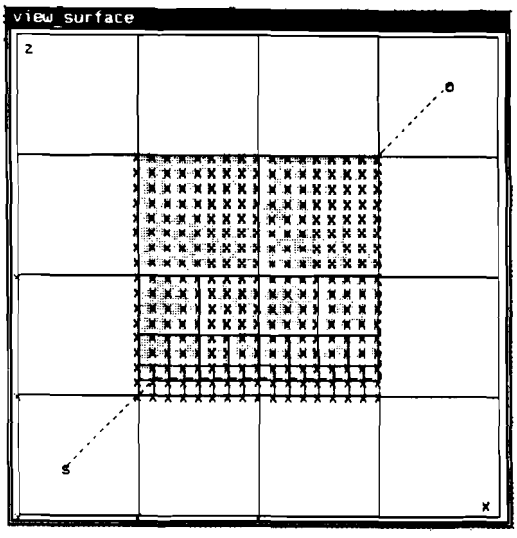
The appendices contain examples of the experimental obstacle configurations and the results from the measurements.

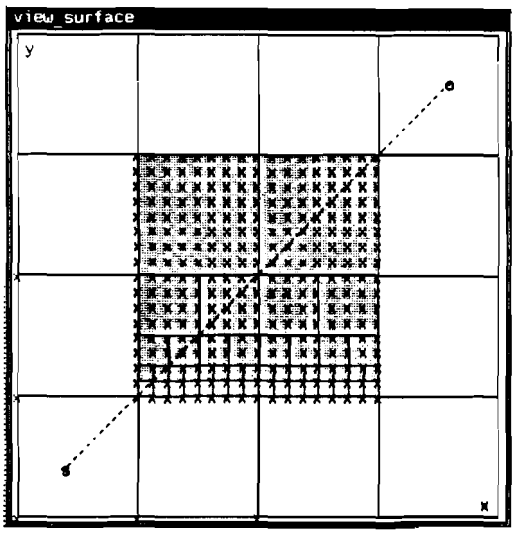
Appendix 1 : The 'cube' obstacle configuration;
Appendix 2 : The 'wall_1' obstacle configuration;
Appendix 3 : The 'sphere' obstacle configuration;
Appendix 4 : The 'maze' obstacle configuration;
Appendix 5 : The 'cylinders' obstacle configuration;

APPENDIX 1 : *the 'cube' obstacle configuration*

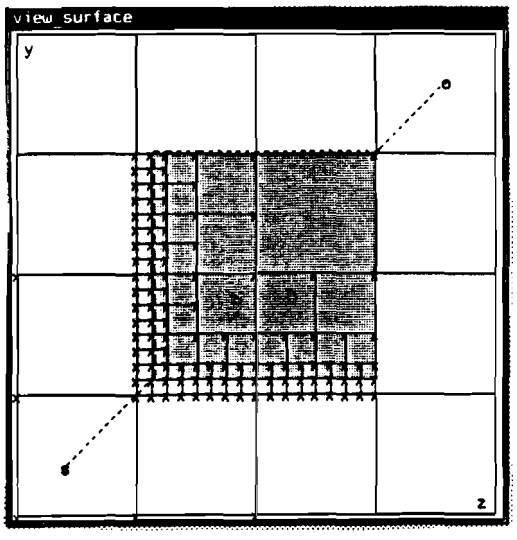
Example of Local octree search at resolution level 5.

top view



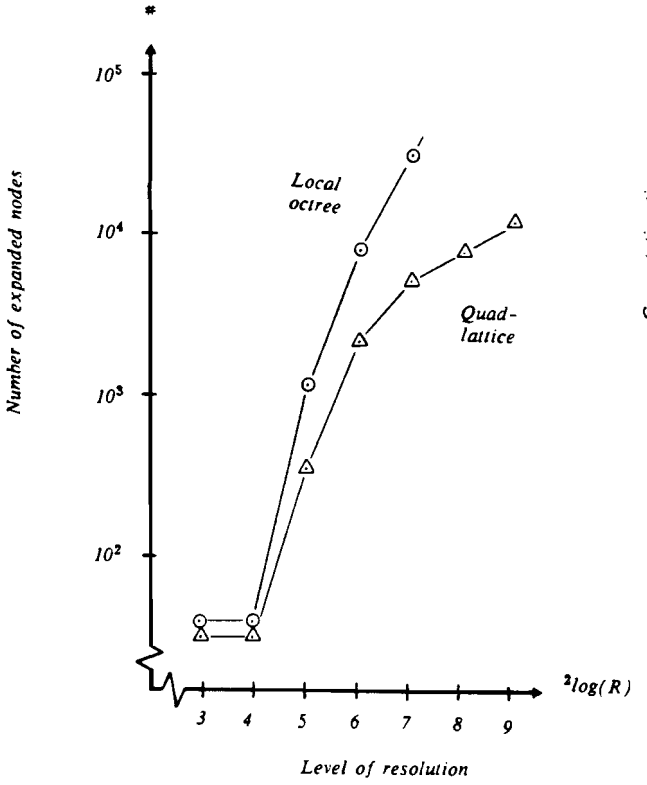


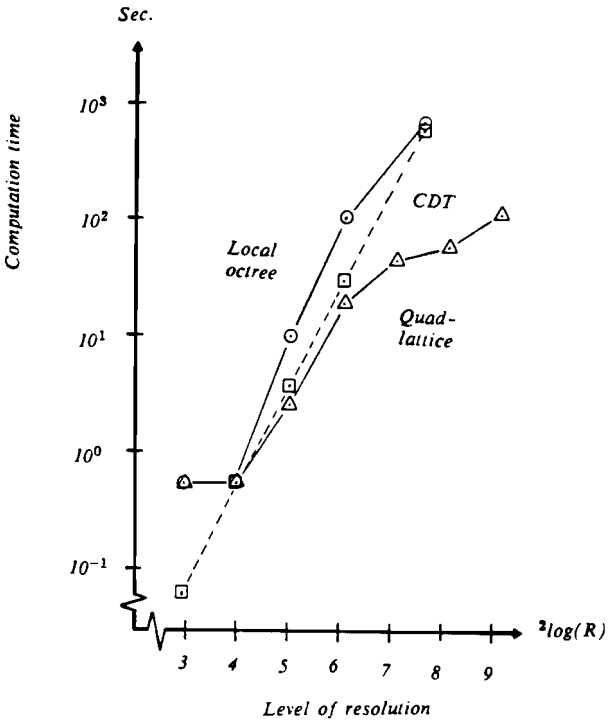
front view

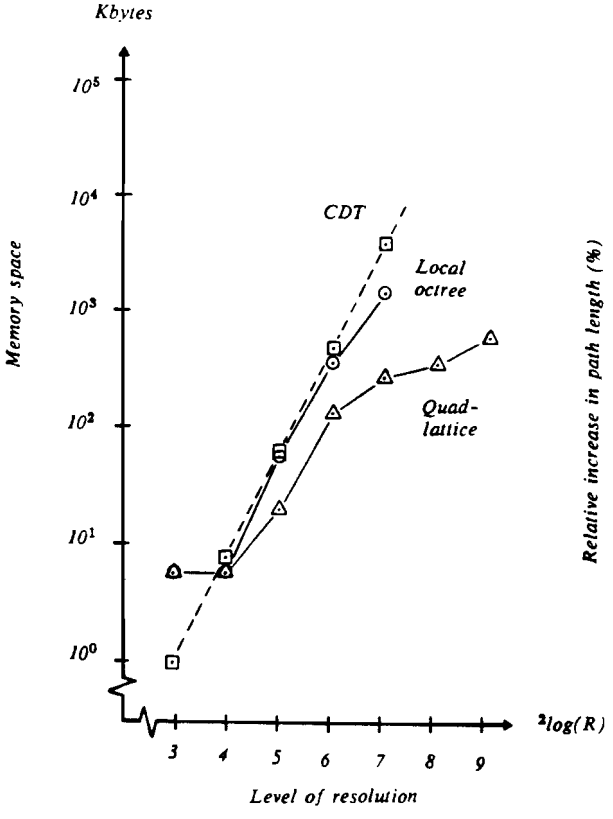


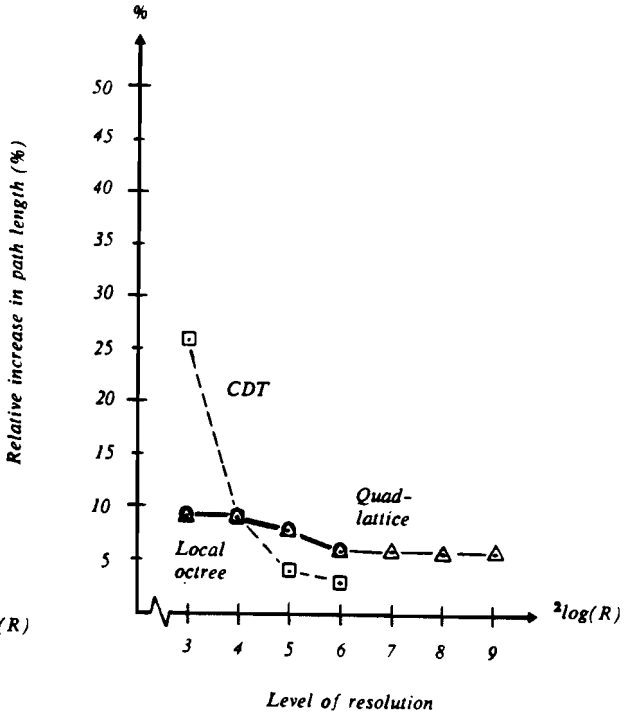
side view

Data measured for the 'cube' obstacle configuration.





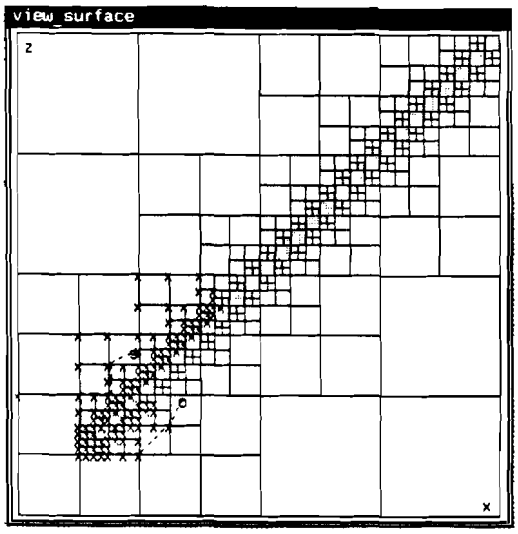


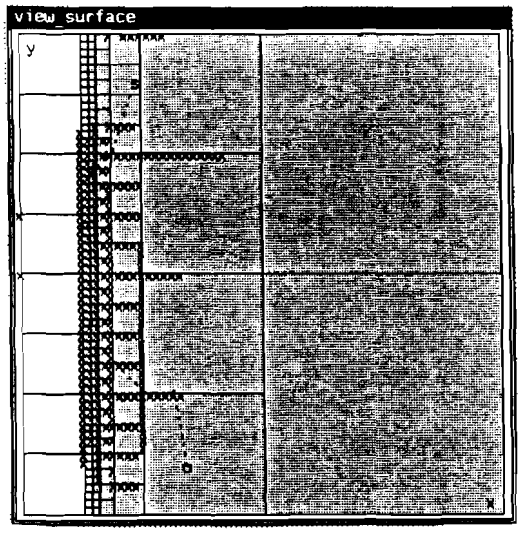


APPENDIX 2 : *the 'wall_1' obstacle configuration*

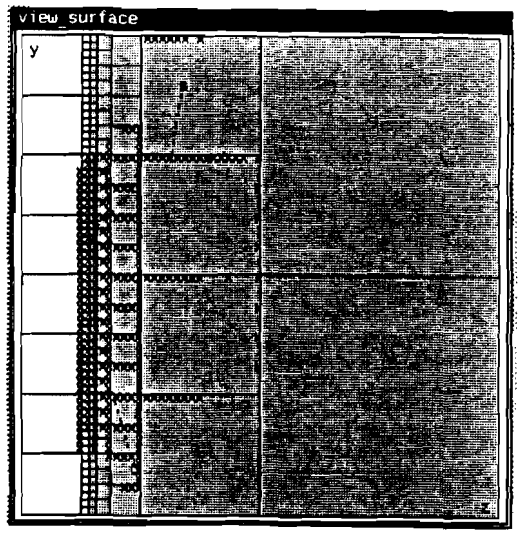
Example of Quad-lattice search at resolution level 6.

top view



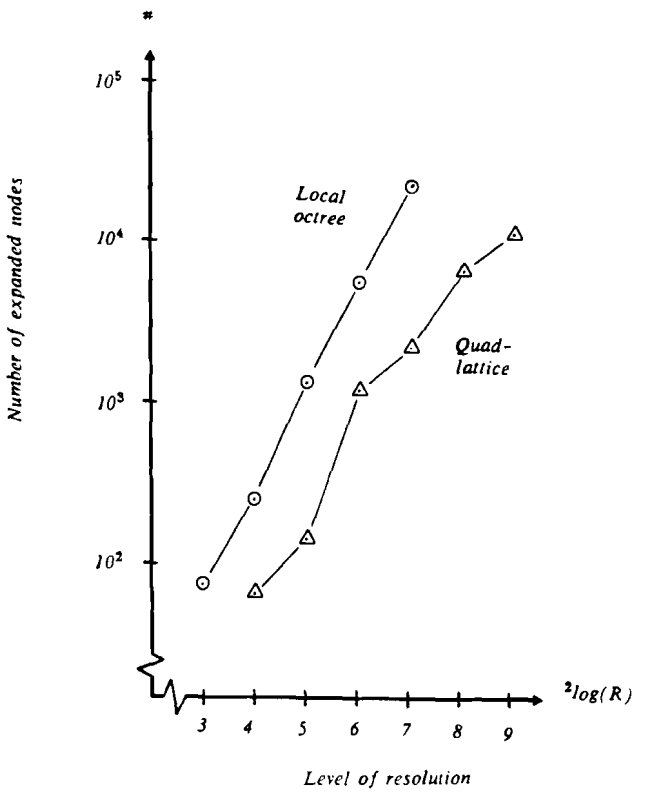


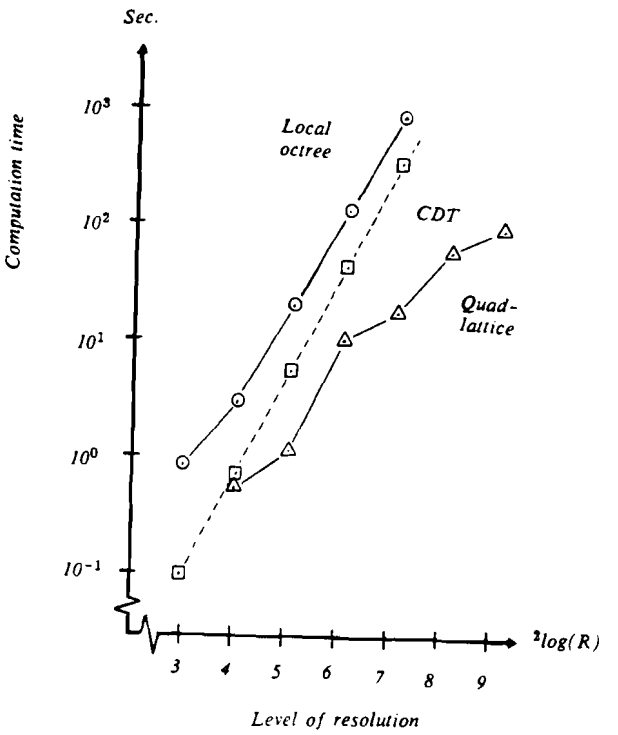
front view

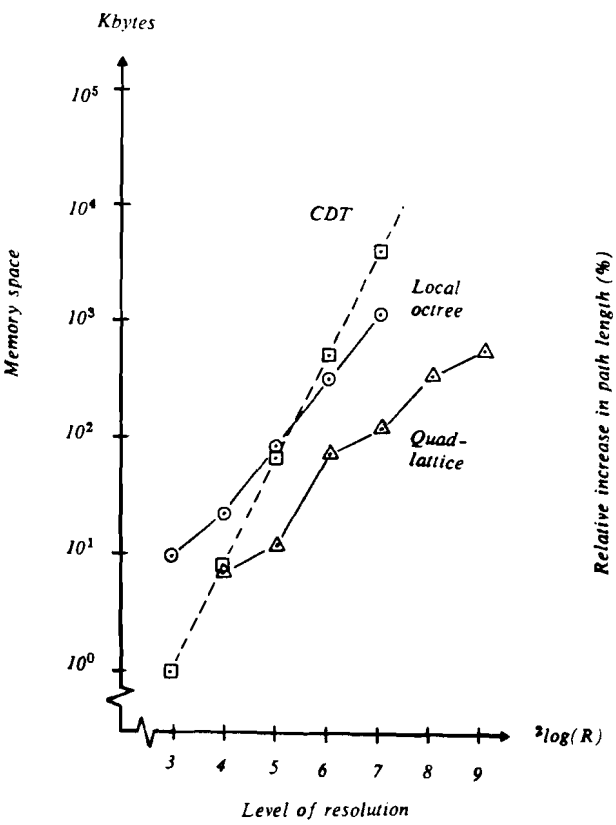


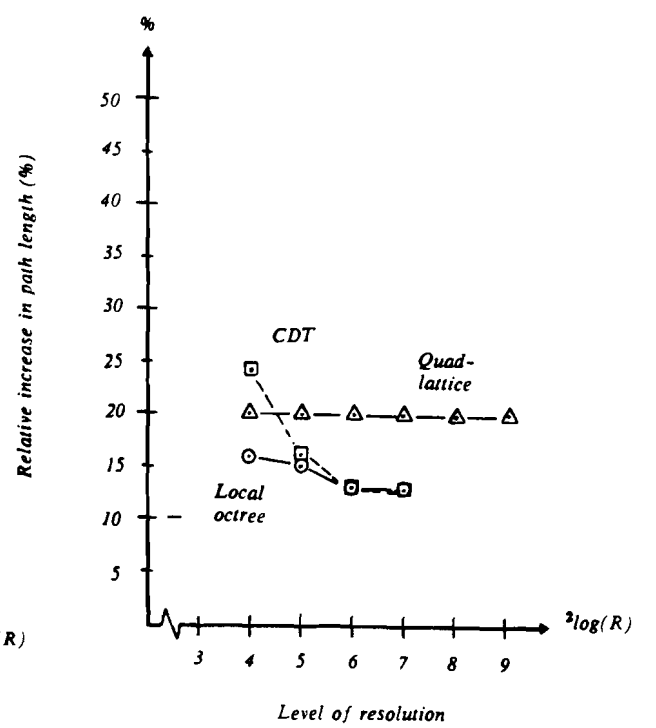
side view

Data measured for the 'wall_1' obstacle configuration.





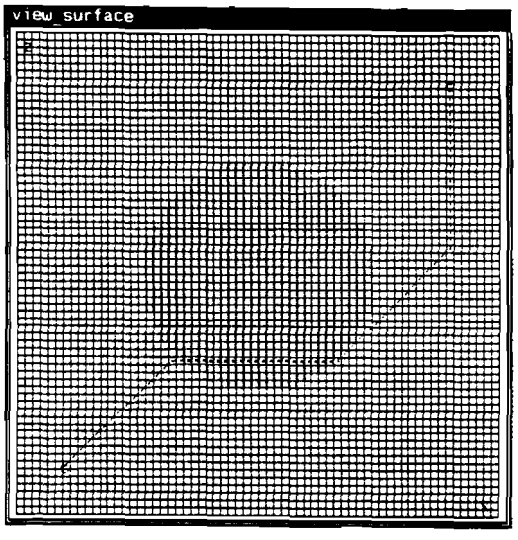


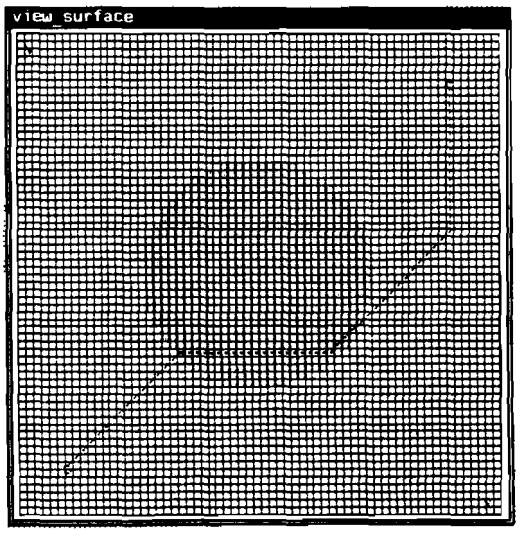


APPENDIX 3 : *the 'sphere' obstacle configuration*

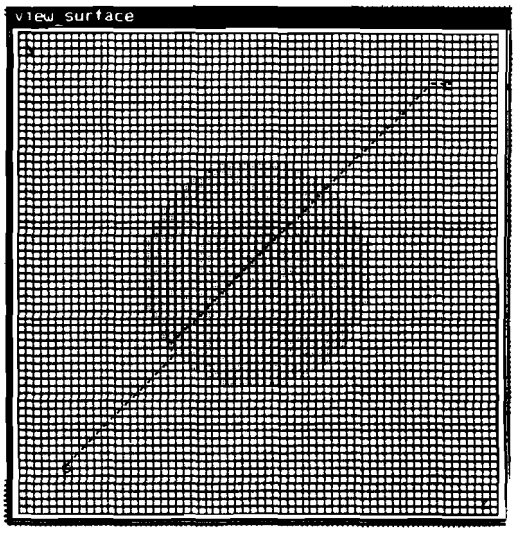
Example of the CDT at resolution level 6.

top view



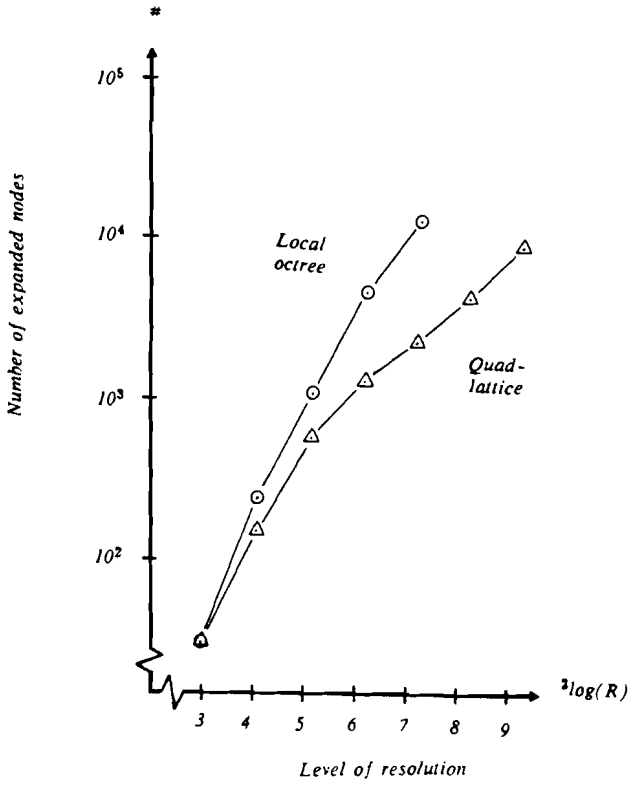


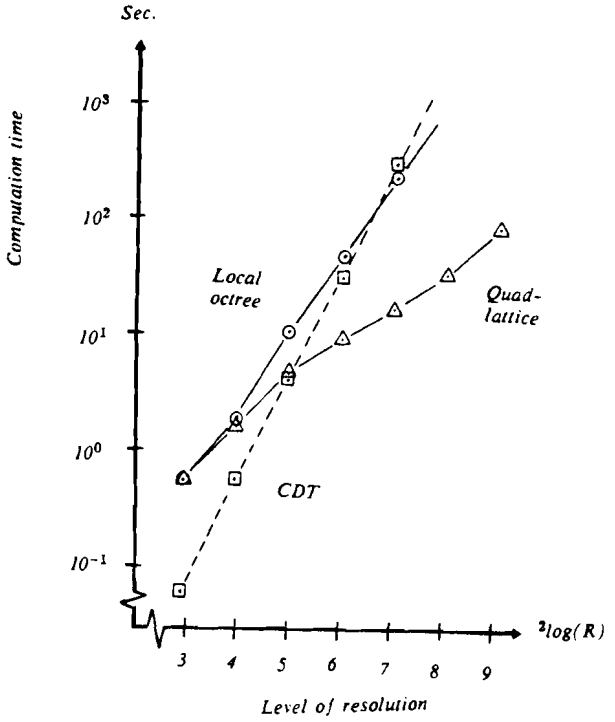
front view

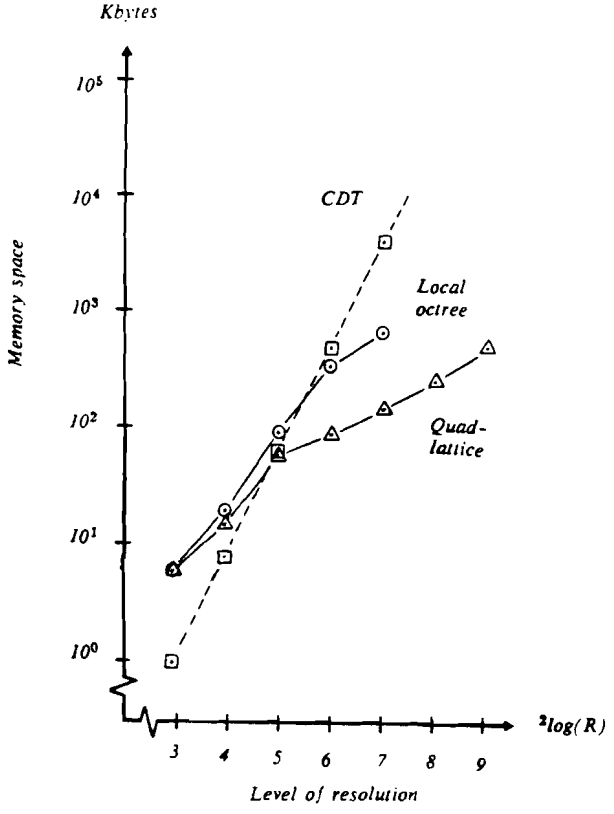


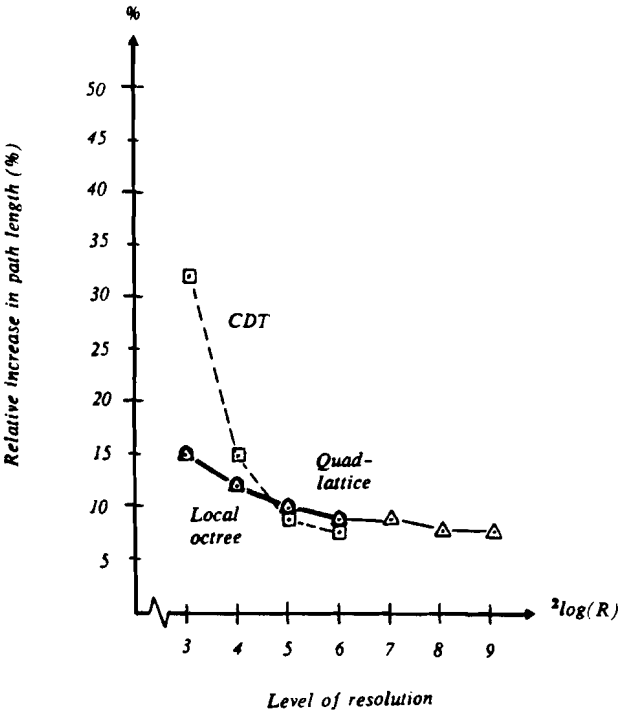
side view

Data measured for the 'sphere' obstacle configuration.





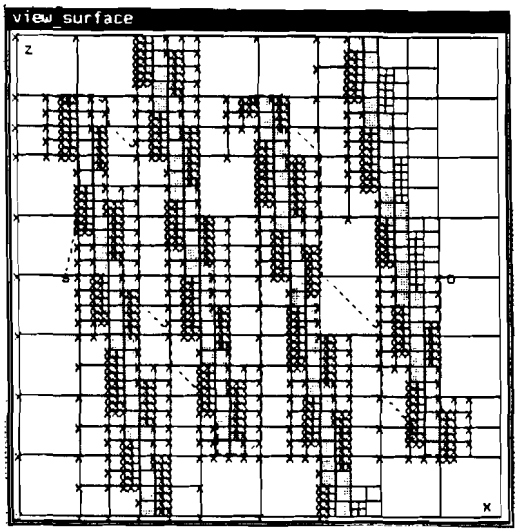


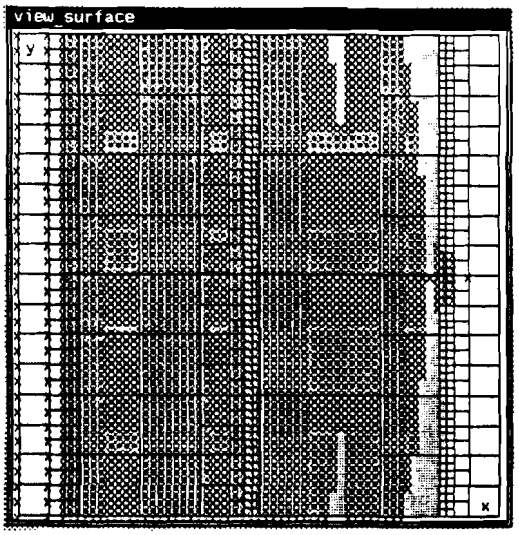


APPENDIX 4 : *the 'maze' obstacle configuration*

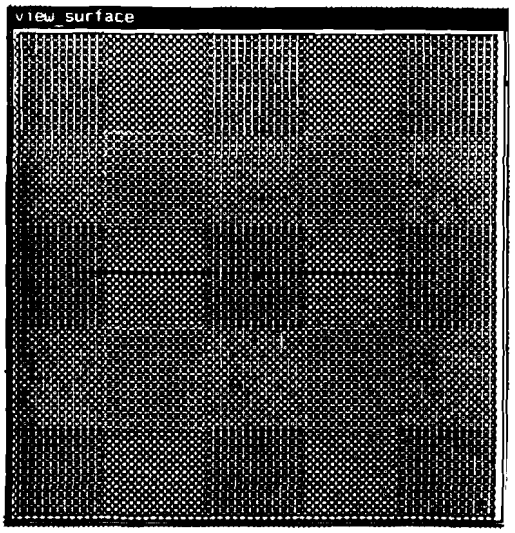
Example of Local octree search at resolution level 6.

top view



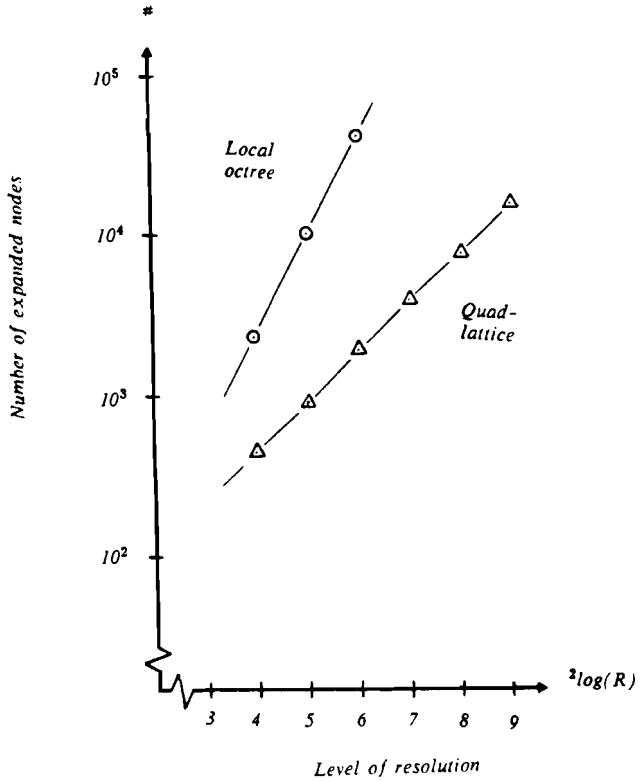


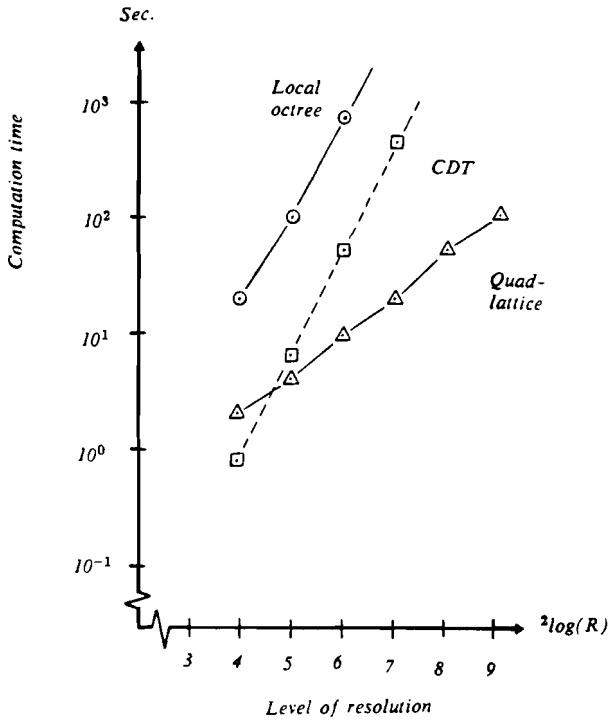
front view

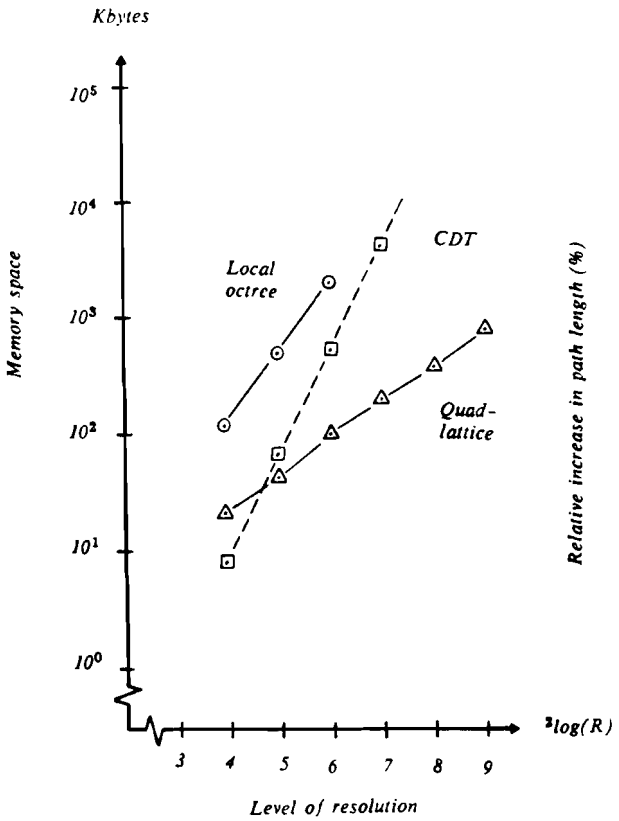


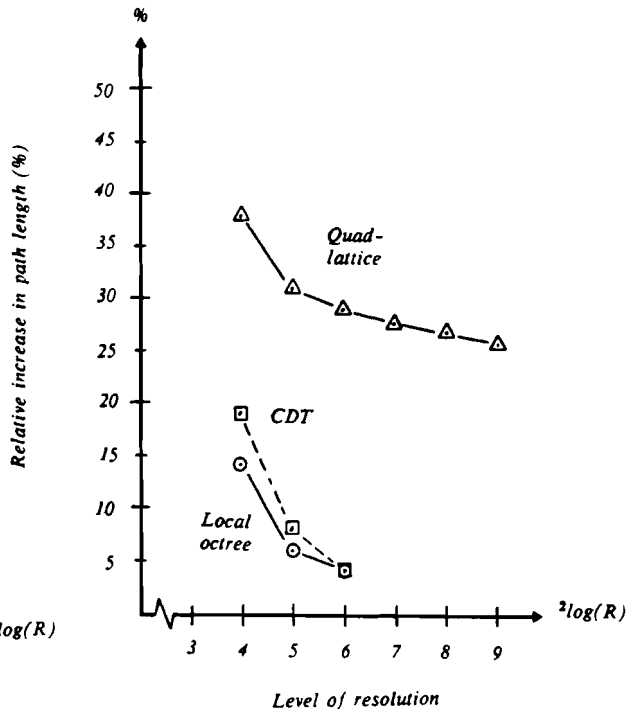
side view

Data measured for the 'maze' obstacle configuration.





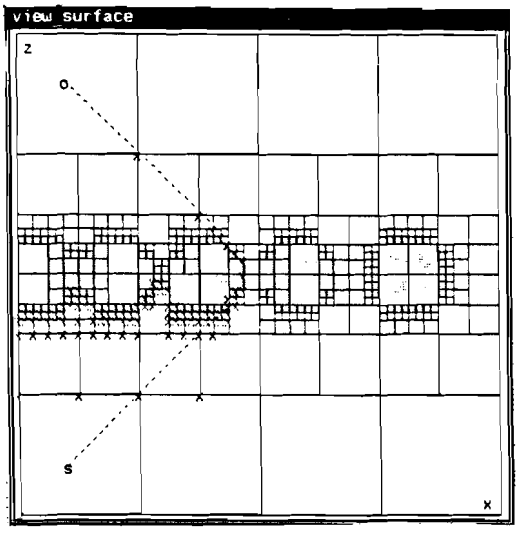


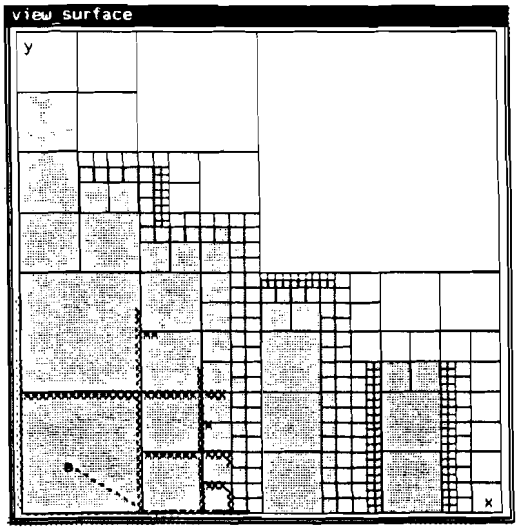


APPENDIX 5 : *the 'cylinders' obstacle configuration*

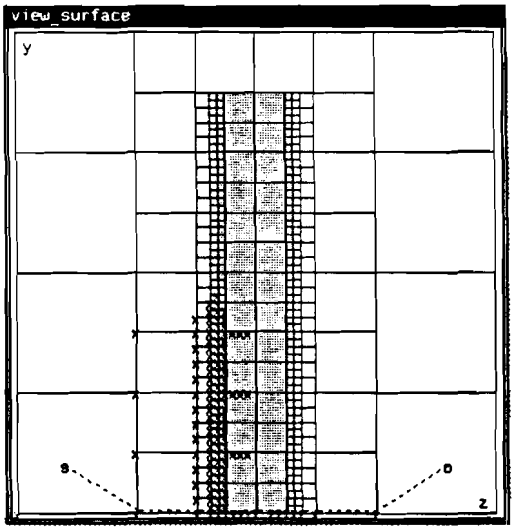
Example of Quad-lattice search at resolution level 6.

top view





front view



side view

Data measured for the 'cylinders' obstacle configuration.