A Computational Technique for Free Space Localization in 3-D Multiresolution Probabilistic Environment Models

Pierre Payeur, *Member, IEEE*

***Abstract*—Probabilistic modeling of two-dimensional or three-dimensional (2-D or 3-D) objects and working environments as quadtrees and octrees encoded with multiple resolutions represents a new trend with numerous applications in computer vision and robotics. The development of neighbor-finding techniques adapted to these tree structures appears as a critical issue for such models to be used properly, especially for path planning and collision avoidance where free space localization is essential. In this paper, a generic neighbor-finding framework that is based on a recursive addressing scheme directly operating on a hierarchical tree structure without the need for preprocessing of raw occupancy measurements generated by range-sensing devices is presented. Neighboring cell addresses are processed in a way similar to basic arithmetic operations with carry given a displacement direction and the address of a starting cell. Neighboring rule sets are derived for a quadtree and extended to an octree. Special cases resulting from multiresolution maps are handled, while the algorithm complexity is kept low to ensure good performances. The approach is developed and validated in the context of collision avoidance for autonomous robotics.**

***Index Terms*—Multiresolution maps, neighbor finding, path planning, probabilistic modeling, virtual navigation.**

I. Introduction

MODELING the environment in which an autonomous system operates is a critical issue in robotic applications. The availability and fast access to information about the cluttering of space are primordial for solving many problems in this field. The selection of a model structure must thus be made carefully while considering the faithfulness of the model to the real world and the efficiency of data retrieval from the model. Quadtrees and octrees demonstrate important advantages to satisfy these requirements mainly because of their compactness and their capability to represent multiresolution models that provide higher performance in three-dimensional (3-D) space representations.

Quadtrees or octrees are often confounded with Cartesian grids. Indeed, both representations contain the same information. The difference resides in the encoding approach. The Cartesian grid is a bounded region of two-dimensional (2-D) or 3-D space that is recursively subdivided in two along each of its axes. As the subdivision level increases, the grid resolution also increases. If each cell is tagged with the occupancy state of the corresponding space, this results in a representation of increasing precision. Models can also be built with multiple levels of resolution depending on the uniformity of space state.

The concept of quadtrees has been introduced by Klinger [1] in the early 1970s and thereafter extended to 3-D space. Quadtrees and octrees are directly associated to Cartesian grids as they provide more compact encoding structures [2]. The principle consists of associating one branch of the tree to each cell of the Cartesian grid. This branch can be recursively subdivided into multiple subbranches (four for a quadtree and eight for an octree). The mother branch that corresponds to the entire modeled volume is seen as the root of the tree. When a branch does not need to be further subdivided, it is considered as a terminal leaf to which the occupancy state of a given region of space, or any other type of information, can be attached.

From the standpoint of geometry, the analogy between Cartesian grids and quadtrees or octrees is shown in Fig. 1. However, an important difference between these encoding schemes is that tree structures do not contain any direct information about the Cartesian position of a given cell. All cells are defined in relationship with the origin and the size of the mother cell that represents the entire volume that is modeled. The traversal path from the mother cell to a cell of interest in the tree structure is then the only way to locate a given area or volume.

Several applications of quadtrees and octrees require means of computing spatial displacements based on the geometric cell connectivity in the model as if it were encoded in a Cartesian grid. Split-and-merge segmentation in computer vision, objects' properties estimation in biomedical imaging [3], and collision-free path planning for robots [4] are all important examples of such applications. In the later case, a neighbor-finding technique is needed to identify free paths among a set of objects. In the case of a manipulator where the entire structure of a robot needs to be moved without colliding with the environment, the number of cells to validate is significantly increased in 3-D space in comparison with the problem of mobile robot path planning that is often limited to the displacement of a single point on a planar surface. An efficient neighbor-finding technique is therefore essential to quickly locate connected free space areas in 3-D volumes.

Manuscript received October 30, 2004; revised June 16, 2006. This work was supported by the Canadian Foundation for Innovation, the Ontario Innovation Trust, and the Natural Sciences and Engineering Research Council of Canada.

The author is with the Vision, Imaging, Video, and Autonomous Systems Research Laboratory, School of Information Technology and Engineering, University of Ottawa, Ottawa, ON K1N 6N5, Canada (e-mail: ppayeur@site.uottawa.ca).

Digital Object Identifier 10.1109/TIM.2006.881028

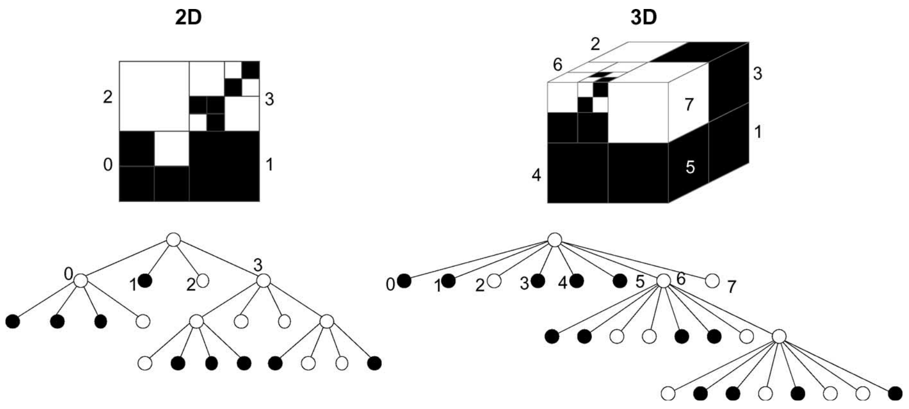

Fig. 1. Two-dimensional and 3-D Cartesian grids and corresponding quadtree and octree.

Neighbors are easy to identify in a Cartesian grid (even 3-D) since the origin and the size of each cell are known. It is then easy to jump from a given cell to one of its neighbors with the help of standard Cartesian coordinates. On the other hand, in a tree structure model of the workspace, neighboring leaves no longer coincide with geometrically adjacent cells in space, and steps from a leaf to its neighbors must then be defined without referring to Cartesian coordinates.

Several research works can be found on building and using tree-based structures [5]–[7]. The majority of these publications are concerned with 2-D space. The classical neighbor-finding approach proposed by Samet is based on the search of a common ancestor cell [8], [9]. This technique has been revisited by Besançon [10], who designed a neighbor-finding algorithm that relies on the encoding structure introduced by Ballard and Brown [11]. The idea consists of backtracking in the tree, starting from the initial cell, until a common ancestor of this cell and its neighbor of interest is reached. From this point, the algorithm goes down the tree and reaches the desired neighbor cell. This approach takes advantage of the recursive structure of the tree. Logical functions are introduced to verify the neighboring status between two given cells, and the algorithm is designed to minimize backtracking. For this reason, it stops as soon as the first common ancestor is found. Nevertheless, it tends to generate long traversals of the tree before a valid ancestor can be found, especially in 3-D models. In addition, neighbor finding between corner neighbors appears to be more difficult to process than between face neighbors.

Some straightforward approaches exploit the size of the model rather than the tree structure. Geometric data are encoded in the cells, such as the coordinates of the origin and the size of each cell. With this information available, the identity of a neighbor cell in a given direction is easily computed. In retrospect, such a technique is equivalent to encoding the model as a Cartesian grid rather than a quadtree or an octree. The advantages of tree structures, such as their compactness, are then lost. Variations of such strategies include the work of Klinger and Rhodes [12], who propose to identify a sister cell in a given direction in a quadtree by means of a sequence of primitive displacements. For this purpose, some complex primitives that lead to an important computational load are used. Other researchers propose to add some data inside of each branch or leaf of the tree in order to keep the identity of every neighbor cell. Hunter and Steiglitz [13] call this supplementary information ropes that are inserted in the model during its building. These ropes allow us to directly access the neighbor cell in a given direction without the need for a neighbor search while the model is in use. On the other hand, extra memory space is required to store the ropes, and the neighbor-finding process complexity is not reduced since all neighbors must be computed for every cell while the model is built. Because only a small percentage of ropes are actually used in a given application, the computing time is wasted in preprocessing many ropes that will never be used.

Address-based computational search strategies have also been proposed as extensions to the original encoding schemes primarily aiming at reducing storage space. Gargantini [14], [15] introduces the concept of linear quadtrees that encode discrete data (black or white pixels) and associates generic addresses to a 2-D binary array of cells of equal size. An algorithmic procedure is proposed for adjacency determination that manipulates the intermediate address representation, achieving a computational time proportional to the number of resolution levels. The addressing framework is robust to variations in the number of resolution levels, but the constraints imposed on the discrete nature of the data being encoded in an array of cells of equal size preempt the direct use of linear quadtrees with multiresolution probabilistic maps. Schrack [16] proposes the use of dilated integers to simplify the classical addition operation on the coordinates of a cell to determine its location code or address, achieving constant-time operation. Although this framework is extended from quadtrees to octrees, it is also designed for equal-sized neighbors.

Binary trees (bintrees) that recursively subdivide space until uniform regions are achieved have also been proposed to provide more compact models than classical quadtrees and octrees [17], [18]. From this representation, neighbors can be identified through a combination of algebraic and logical operations with a complexity proportional to the size of the model encoded as a bintree. This approach allows us to handle multiresolution models and finds neighbors of different sizes as long as the neighbor cell is equal or larger than the starting cell, but the recursive subdivision operation implies that discrete data are encoded to create uniform regions, and the preprocessing

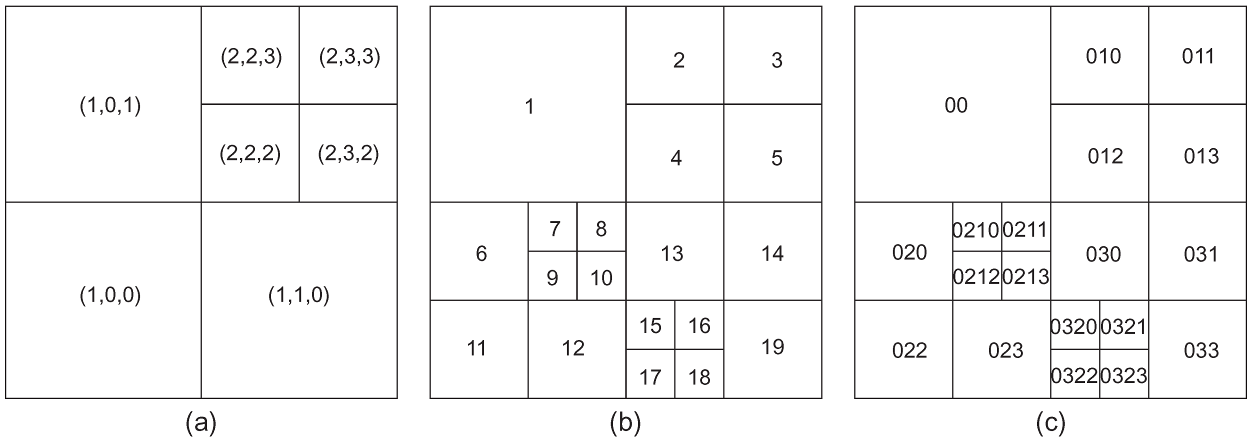

Fig. 2. Various classical addressing schemes for 2-D Cartesian grids. (a) Shu and Kankanhalli. (b) Samet. (c) Major *et al.*

phase required to convert an occupancy grid into a bintree representation is inappropriate for the dynamic environment mapping often found in robotic applications.

This paper, which is an extended version of [19], proposes a neighbor-finding approach operating directly on a multiresolution tree structure encoding nondiscrete data (probabilistic) and implies neither any backtracking in the quadtree or octree nor the encoding of any supplementary data in the model. This method is validated for 2-D and 3-D models (quadtrees and octrees). The algorithm is based on an addressing scheme that identifies each cell with a specific logical tag. Starting from a given address, the addresses of neighbor cells are computed using simple algebra, which is defined in lookup tables, that depends on the displacement direction and on the address of the starting cell. Special cases resulting from different cell sizes are also handled efficiently by the framework.

Section II provides details about the selected addressing scheme of cells in quadtree and octree structures. Section III describes the proposed address-based neighbor-finding algorithm for a 2-D model (quadtree). In Section IV, the approach is generalized to 3-D models (octrees). Section V presents considerations about model encoding and neighbor finding in multiresolution models and examines the effect of model borders on the neighbor search algorithm. Section VI discusses the complexity of the approach. Finally, Section VII examines applications to free space localization in multiresolution probabilistic occupancy maps for robot path planning and presents some experimental results.

II. Cell Addressing

The basis of the proposed free space localization algorithm in quadtrees and octrees consists of an addressing scheme that associates a numerical identity with each branch and each leaf of the tree structure. Various classical addressing schemes have been considered. For instance, Shu and Kankanhalli's addressing method [3] relies on the subdivision level L and on the relative coordinates (x, y) of a given cell with respect to the origin, as shown in Fig. 2(a). Here, the concept is presented on a 2-D multiresolution grid for clarity. The address associated with each cell is of the form (L, x, y), where the mother cell receives the address (0, 0, 0). This type of addressing creates a tree for which it is not required to use pointers that connect leaves and branches to each other. It results in a simple structure but at the expense of a complex set of logical rules to process the addresses. Especially, a complete update of all addresses is required if the subdivision level of a part of the model is changed, making this encoding inadequate for dynamic modeling.

Samet [6] proposes a continuous addressing scheme, as shown in Fig. 2(b). The address grows with the number of independent cells in the grid. In spite of the relative simplicity of this addressing approach, it appears to be difficult to define a standard numbering strategy since the subdivision level of each region of space is not known *a priori* in a model in evolution like that found in autonomous robotics. Major *et al.* [20] propose a structured addressing method, as shown in Fig. 2(c). Cells are incrementally numbered from 0 to 3 following a fixed scanning pattern of the four children (for a quadtree) resulting from the subdivision of a mother cell. The number of digits varies with the resolution level as one digit is associated with each level starting from the lowest one (the entire model). As in the example of Fig. 2(c) for a 2-D map, the mother cell is tagged 0 while its four children are, respectively, tagged 00, 01, 02, and 03 in a predefined order. In accordance with this pattern, the four children of the second child (01) are, respectively, tagged 010, 011, 012 and 013.

This addressing follows a logical numbering scheme that depends on the resolution level of each cell. For this reason, it remains perfectly independent from the subdivision state of a given region and allows dynamic addressing of multiresolution grids. Local addresses can easily be updated following the need to increase or decrease the resolution of some areas of the model or to merge groups of cells exhibiting similar characteristics. This scheme then provides the required flexibility and adaptability for complex environment representations. Moreover, this addressing scheme can directly be extended to 3-D space, where the subdivision of a cell results in eight children instead of four. For instance, the eight children of the first 3-D mother cell would receive, respectively, the addresses 00, 01, 02, 03, 04, 05, 06, and 07, as shown in Fig. 3(b).

It must be noted that the definition of the scanning pattern and the location of the origin are not critical issues. The ordering of addresses can vary, provided that the same scanning pattern is consistently applied at every resolution level and that the same

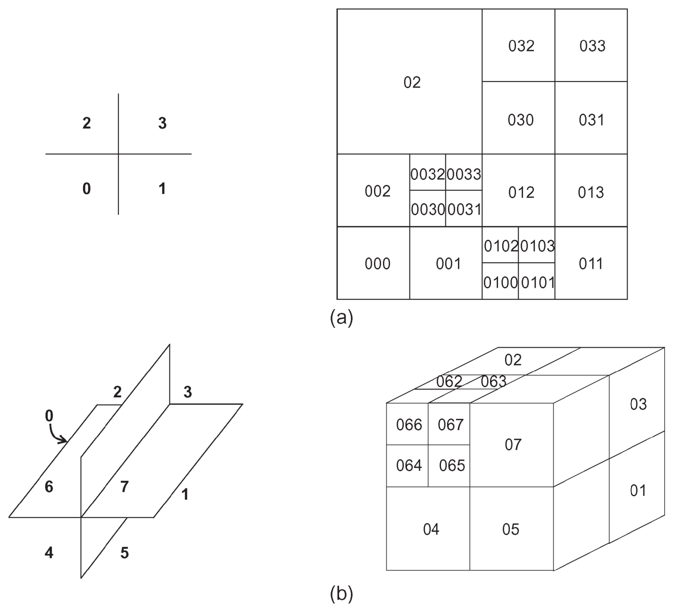

Fig. 3. Incremental addressing scheme in (a) 2-D space and (b) 3-D space.

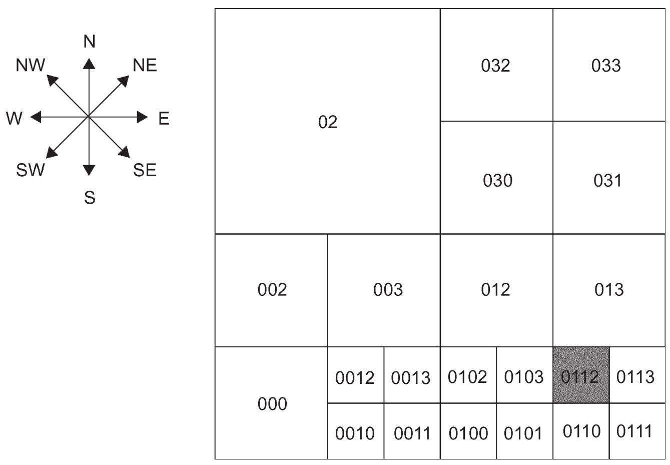

Fig. 4. Two-dimensional Cartesian grid with various levels of resolution.

numbering structure is preserved. The repetitiveness characteristic that results is essential to take advantage of the addressing scheme for neighbor cell identification in 2-D or 3-D space. In a previous work on probabilistic quadtree modeling of mobile robot environments, Tremblay [21] used an addressing pattern similar to that of Major *et al.*, except that the origin is located at the bottom-left corner of the first mother cell, as shown in Fig. 3(a). In this paper, the Major *et al.* addressing scheme is combined with Tremblay's scanning pattern and extended to 3-D space modeling, as shown in Fig. 3(b). The numbering order is defined in accordance with the standard x-, y-, and z-axes. Although this is not critical for the validity of the proposed neighbor-finding approach as demonstrated in the following sections, this choice leads to a simpler implementation.

III. Neighbor Identification in Quadtree Structures

The proposed neighbor-finding technique results from the logical behavior of the selected addressing scheme. Close observation of the addressing structure leads to a generic set of rules that determine the address changes when one moves from a given cell to its neighbor for each possible direction. The advantage is that these rules are similar to elementary arithmetic operations with carry on the next left digit that composes the address. Therefore, they can be efficiently processed on any computer. Moreover, no particular encoding needs to be defined as in Gargantini's and Schrack's approaches [15], [16]. Selection of an incremental and coherent addressing scheme is the only constraint to meet since the rules are defined in accordance with this scheme. However, any coherent pattern can be used as the rules only slightly differ between two different addressing schemes but can be obtained similarly, making the proposed approach general. In this section, a neighboring rule set is developed for the addressing scheme defined in Fig. 3(a) for quadtree models. The extension of these rules to octrees for the corresponding 3-D addressing scheme is presented in Section IV.

To illustrate the rule definition process, let us consider the 2-D Cartesian grid shown in Fig. 4. Possible directions of displacement are identified by means of cardinal directions. In the 2-D problem, these are north (N), south (S), east (E), west (W), northeast (NE), northwest (NW), southeast (SE), and southwest (SW). Therefore, neighbors can be grouped into two categories, namely 1) "edge neighbors," located in the N, S, E, and W directions and 2) "vertex neighbors," located in the NE, NW, SE, and SW directions. In order to define the neighboring rules, we examine how addresses are modified when a displacement occurs along each of these directions. The relative position of the starting cell among the four children also affects the definition of the rules.

"Sister cells" are defined as the four cells (in 2-D space) that result from the subdivision of the same mother cell. "Cousin cells" are defined as cells whose mother cells are sister cells. From there, neighboring rules are built by successively comparing only the digits of the cells' addresses that correspond to the same level of resolution, that is, the digits that occupy the same position in the address expression. The computation is thus performed one level at a time starting at the highest resolution level (the right-most digit) until the lowest resolution level is reached (the left-most digit). Based on these remarks and observing Fig. 4, the following observations may be established.

A. Edge Neighboring Between Sister Cells

Moving from the cell tagged 0112 in the S direction implies that only the right-most digit is affected. The address changes from 0112 to 0110. Hence, moving down to the S direction from child number 2 within a group of four sister cells always conducts to child number 0 at the same level of resolution. This behavior occurs for all levels of resolution and for any area of the grid. In a similar manner, starting from child number 2 and moving in the E direction always conducts to child number 3 at the same level of resolution. Considering cell 0112 as the starting one, the neighboring cell address in the E direction is 0113.

According to these observations, for any group of sister cells, a subset of the neighboring rules can be established: the digit of interest changes from 0 to 1 or from 2 to 3 when moving in the E direction, from 0 to 2 or from 1 to 3 in the N direction, from 1 to 0 or from 3 to 2 in the W direction, and, finally, from 2 to 0 or from 3 to 1 in the S direction. These facts are a direct consequence of the selected addressing scheme. Therefore, neighboring rules need to be slightly modified if another addressing scheme is used. However, the definition process remains identical and needs to be performed only once, i.e., when the scheme is selected.

B. Edge Neighboring Between Cousin Cells

Moving from the cell tagged 0112 in the W direction implies that the right-most digit changes to 3, as seen in Fig. 4, but the next digit starting from the right-most one must also be modified. This illustrates the second important phenomenon defining the neighboring rule set. When a displacement from a given cell to its neighbor implies a change in the parents of these cells, the digit associated with the parent resolution level is also affected. Neighboring rules must then be applied successively to each digit starting from the right-most one, until the affected parent resolution level is reached. For example, starting from 0112 and moving in the W direction first implies that the right-most digit 2 is changed for a 3. Next, the second digit starting from the right (i.e., 1) is changed to a 0 since the movement is toward the W direction (parents are sister cells). The two left-most digits 01 are not affected since the grandparents of the starting cell and of the neighbor cell are the same. The complete address of the neighbor cell is then computed in two steps and results in 0103. In terms of computation, moving from a parent to another is similar to the application of a carry on the immediate left digit in elementary arithmetic. Processing a carry, to which a direction is associated here, follows the same neighboring rules as those applied to the previous digit.

C. Vertex Neighboring Between Sister Cells

For vertex neighbors, the rule definition follows the same logic. For example, if one moves from the cell tagged 0112 in the SE direction, the cell tagged 0111 is reached. Only the right-most digit is affected since this corresponds to a displacement between sister cells. In accordance with the selected addressing scheme, digits are changed from 2 to 1 in the SE direction, from 1 to 2 in the NW direction, from 0 to 3 in the NE direction, and from 3 to 0 in the SW direction. These displacements do not imply carries since they occur between children of the same parent cell.

D. Vertex Neighboring Between Cousin Cells

Since reaching a neighbor cousin cell implies a change in the parent cell, the principle of carries of the left digits is applied as with edge neighboring between cousin cells. For example, starting from the cell tagged 0112 and moving in the SW direction, the right-most digit changes from 2 to 1, and a carry is applied on the second right-most digit that is initially 1. A W-directed displacement is associated with this carry since the neighbor parent 010 is located in the W direction with respect to the starting cell parent 011. The rule defining this movement between sister cells from a child tagged 1 toward the W direction is then applied, and the second right-most digit is changed from 1 to 0. Once again, the grandparents are the same; therefore, the two left-most digits 01 are not affected. Finally, the complete address of the neighbor cell is 0101.

E. Distant Relationship Neighboring

It can occur that carries are transmitted to more than one left digit. This corresponds to displacements that imply geometrically neighboring cells whose parents are not sister cells or cousin cells. The relationship between these cells is located at a lower resolution level, for instance, at the grandparent level or beyond. Such a situation occurs if one moves in the SW direction from the cell tagged 0102 in Fig. 4 to reach cell 0011. Modification to the right-most digit from 2 to 1 brings a W-directed carry of the second right-most digit since parents are changed from 010 to 001. The carry calls the neighboring rule from a 0-tagged cell in the W direction. This results in a 1-tagged cell and in the occurrence of a new W-directed carry on the third right-most digit since a grandparent change is also implied. This new carry calls again the neighboring rule set for a move from a 1-tagged cell in the W direction. In accordance with the selected addressing scheme, this brings a 0-tagged cell. Therefore, the grandparent address is changed from 01 to 00. This last call to the neighboring rule set does not imply any other carry and address processing is stopped, resulting in the final address being 0011.

This example demonstrates the generality of the proposed approach. An elementary rule set defined from the observation of address modification between edge neighbors and vertex neighbors for sister and cousin cells is applied successively to each resolution level until a common parent cell is reached. Because of the recursive structure of the quadtree, this situation must occur. In the worst case, neighboring rule set processing continues until the second left-most digit is reached, but this occurs only when the boundaries between each of the four children of the global mother cell are traversed. In the large majority of situations, the carries on the left digits are limited to higher levels of resolution (few right-most digits).

Although a given number of carries might be required to find the address of a neighbor cell, no backtracking in the tree structure is performed. Only arithmetic manipulations on the address digits are necessary once the rule set is defined for a given addressing scheme. This significantly speeds up processing as the model stored in memory does not need to be accessed before the final address is computed given that no validation of the actual neighboring status between cells needs to be performed. Computation of the destination cell's address with the proposed computational technique ensures that only valid neighbors are reached in the desired direction, provided that such neighbors exist in the model. Otherwise, the closest available ancestor is automatically identified.

The application of the neighboring rule set is driven by two simple principles. First, processing an address always begins with the right-most digit (the highest resolution level) and successively proceeds to the left as carries are generated.

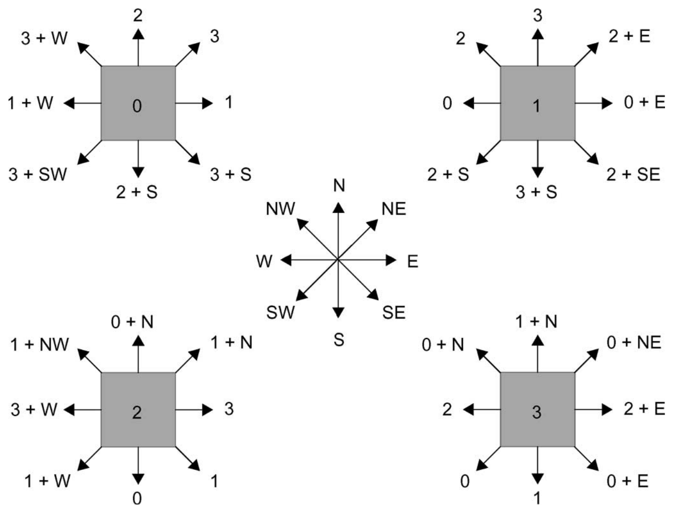

Fig. 5. Generic neighboring rule set for a quadtree structure.

Second, processing is based on the initial value of the digit in the starting cell's address and on the displacement direction. These two parameters (initial value and direction) alone determine the unique rule to apply to identify the appropriate new value for the considered digit. Given its simplicity, the set of rules is advantageously encoded in a lookup table to speed up processing.

Fig. 5 graphically summarizes the entire set of rules for quadtrees associated with the selected addressing scheme described in Section II. This diagram defines the algebra to be used for a 2-D map. The numbers in square boxes correspond to each possible value of the digit in a given position of the initial address, while arrows represent displacement directions in accordance with the compass card. Alphanumeric codes at the extremity of the arrows define the required operation to be performed to compute the new address. When only a number (0, 1, 2, or 3) is indicated, this means that the initial value of the digit must be replaced by this number, and no carry is generated (sister cells neighboring). The addition of a carry mark $(+x)$ indicates that a carry in the x-direction must successively be applied on the digit located immediately to the left of the considered digit in the cell address. For example, starting from cell tagged 031 in Fig. 4 and moving in the S direction first calls the rule defined by the square tagged with a 1 (the right-most digit). Looking in the S direction indicates that the processing rule is $3 + S$. This means that the digit value 1 is replaced by 3 and that a carry in the S direction must be applied to the second right-most digit, i.e., the 3 in the initial address (031). The carry then calls the rule defined by the square tagged with a 3 in the S direction, and the processing rule indicates that only this digit 3 must be replaced by a 1. No more carry is generated. The neighbor cell address of cell 031 in the S direction and for the same level of resolution is then 013. For implementation purposes, this rule set is encoded as a lookup table presented in Table I.

IV. Neighbor Identification in Octree Structures

As shown in Fig. 3(b), the addressing scheme and the definition of the neighboring rule set in 3-D structures (octrees) follow the same idea as for the 2-D case. However, the number of rules increases as a consequence of the additional number of neighbors. The number of potential displacement directions is limited to eight in 2-D space but reaches 26 directions for the 3-D case, as shown in Fig. 6.

For 3-D space, neighboring relationships can be grouped into three categories, depending on the type of intersection that exists between cells in a given direction. "Face neighbors" occur in the N, S, E, W, front (F), and rear (R) directions. "Edge neighbors" occur in the NE, NW, SE, SW, front E (FE), front W (FW), read E (RE), rear W (RW), front N (FN), front S (FS), rear N (RN), and rear S (RS) directions. Finally, "vertex neighbors" occur in the front NE (FNE), front NW (FNW), front SE (FSE), front SW (FSW), rear NE (RNE), rear NW (RNW), rear SE (RSE), and rear SW (RSW) directions.

Neighboring rules are defined based on a close inspection of the address behavior when a displacement in each possible direction occurs and for each of the eight possible children that result from the subdivision of a mother cell in an octree. The principle of a carry applied to the immediate left digit is preserved. The resulting set of rules that defines the algebra for an octree is presented under the form of three lookup tables in Tables II–IV, respectively, for each of the three categories of neighbors found in 3-D space. The method to define these rules is identical as for quadtrees, except that extra directions need to be considered.

Using this augmented set of rules for processing addresses of neighboring cells in an octree follows the same steps as in a quadtree. For example, if one moves in the FNW direction starting from the octree cell tagged 0742 that can be located by pursuing recursive subdivisions in Fig. 3(b), this consists of a vertex-type neighboring relation that makes use of Table IV. This table reveals that for a right-most digit value equal to 2 and a displacement in the FNW direction, the initial digit value must be replaced by a 5 and that a carry in the NW direction is applied on the following left digit, e.g., 4 in the initial address. This carry implies an edge-type neighboring relation. As a result, the line labeled 4 in Table III is consulted for the NW direction and indicates that the digit value of 4 must be replaced by a value of 7 and that a new carry is generated in the W direction and applied to the following left digit, e.g., 7 in the initial address. This carry corresponds to a face neighbor and calls, as a final step, for Table II to be consulted to determine that, for an original digit value of 7 with a displacement in the W direction, the value must be replaced by a 6, and no more carry is generated, thus completing the processing of the address. Therefore, the neighbor cell address from 0742 in the FNW direction is 0675. This can be verified geometrically by close inspection of Fig. 3(b).

V. Multiresolution Models and Border Effects

Some special cases must be considered to ensure the generality of the proposed neighbor-finding algorithm. These cases are concerned with multiresolution models and border effects. This section demonstrates that the previously defined set of neighboring rules for quadtrees and octrees is applicable in all circumstances.

TABLE I
LOOKUP TABLE IMPLEMENTING THE NEIGHBORING RULE SET FOR A QUADTREE STRUCTURE

		Direction							
		N	S	E	W	NE	NW	SE	SW
Initial Digit Value	0	2	2 + S	1	1 + W	3	3 + W	3 + S	3 + SW
	1	3	3 + S	0 + E	0	2 + E	2	2 + SE	2 + S
	2	0 + N	0	3	3 + W	1 + N	1 + NW	1	1 + W
	3	1 + N	1	2 + E	2	0 + NE	0 + N	0 + E	0

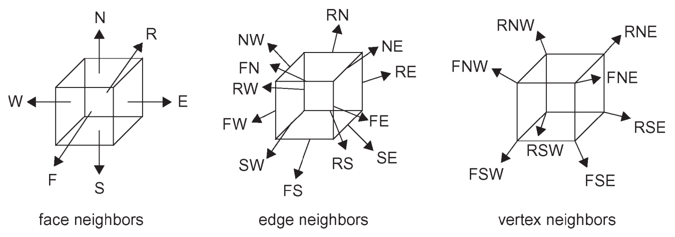

Fig. 6. Twenty six possible directions of displacement in 3-D space.

TABLE II
LOOKUP TABLE FOR FACE NEIGHBORING RULE SUBSET IN AN OCTREE STRUCTURE

		Direction					
		N	S	E	W	F	R
Initial Digit Value	0	2	2 + S	1	1 + W	4	4 + R
	1	3	3 + S	0 + E	0	5	5 + R
	2	0 + N	0	3	3 + W	6	6 + R
	3	1 + N	1	2 + E	2	7	7 + R
	4	6	6 + S	5	5 + W	0 + F	0
	5	7	7 + S	4 + E	4	1 + F	1
	6	4 + N	4	7	7 + W	2 + F	2
	7	5 + N	5	6 + E	6	3 + F	3

A. Multiresolution Models

The neighboring rules for 2-D and 3-D spaces that have been introduced in the previous sections automatically provide the address of a neighbor cell at the same level of resolution in a given direction. This is appropriate for classical Cartesian grids that contain a uniform level of subdivision and do not aim at reducing memory space requirements by combining identical cells. However, such models are not optimal, and Cartesian grids with several levels of resolution are best suited in most applications where model size is significant, and efficient traversal of the model is required.

The rule sets previously defined remain entirely valid for such multiresolution grids encoded as quadtrees or octrees. In the implementation used to develop and test the proposed framework, octree data structures are encoded as pointer-based recursive chained lists containing only the data about the modeled environment, that is, the occupancy probability of a given volume of 3-D space. Compression of the tree structure is not achieved through sequence encoding, as proposed in Gargantini's linear trees [14], [15], or in a similar work, given that the probabilistic models are not suitable for such compression operation because of the nondiscrete nature of the data they contain. Instead, the use of a tree structure with multiple resolutions, which varies locally in accordance with the content of the environment, provides an efficient way to keep storage space requirements within practical limits while minimizing the effort required to access the data, but the existence of leaves corresponding to physical cells of various sizes calls for an adapted neighbor search technique due to complex border effects that occur between nonsister cells that might be of different sizes.

Beyond the ability to handle tree structures containing multiple resolution levels, the proposed approach is independent of the number of resolution levels as the set of rules is generic and remains exactly the same for all resolutions. In fact, resolution level management does not have an impact on the defined neighboring rules but rather influences functions that provide access to the data contained in the tree structure. As a consequence, these access functions can use the addresses provided by neighboring rules in such a way that cells at the proper level of resolution are reached. If the addressing scheme is defined as having one digit per resolution level, it can be used advantageously in the management of access to data encoded into a quadtree or an octree without impeding the proposed neighbor identification procedure. Moreover, the addressing scheme works as a computational artifact that does not need to be explicitly encoded into the model as numerical values associated with each digit of a given address directly map to the corresponding pointer in the data structure. The existence or nonexistence of a cell corresponding to an address at a given level of resolution is determined by accessing the tree structure up to the last available digit, as pointed out by the address.

Given this operational framework, two special cases must, however, be considered to ensure the generality of the proposed approach. The first one is that of a cell whose neighbors are defined at a lower resolution level in a given direction. The neighboring rules then provide an address that corresponds to a cell that is not defined. In other words, the neighbor cell address computed by the proposed approach has too many digits. The access function to data contained in the model is designed such that it stops the descent into the tree when a terminal leaf is reached although the target address corresponds to a branch that is supposed to be further subdivided. An example is shown in Fig. 7, where the E direction neighbor of cell 0013 is only defined with a resolution that is one level lower. The neighbor cell with the highest resolution available in the selected direction is 010, while the target address provided by the neighboring rules is 0102. The access function then automatically reports the data contained in the cell tagged 010

TABLE III
LOOKUP TABLE FOR EDGE NEIGHBORING RULE SUBSET IN AN OCTREE STRUCTURE

	Direction											
Initial Digit Value	NW	NE	SW	SE	FN	RN	FS	RS	FE	FW	RE	RW
0	3+W	3	3+SW	3+S	6	6+R	6+S	6+RS	5	5+W	5+R	5+RW
1	2	2+E	2+S	2+SE	7	7+R	7+S	7+RS	4+E	4	4+RE	4+R
2	1+NW	1+N	1+W	1	4+N	4+RN	4	4+R	7	7+W	7+R	7+RW
3	0+N	0+NE	0	0+E	5+N	5+RN	5	5+R	6+E	6	6+RE	6+R
4	7+W	7	7+SW	7+S	2+F	2	2+FS	2+S	1+F	1+FW	1	1+W
5	6	6+E	6+S	6+SE	3+F	3	3+FS	3+S	0+FE	0+F	0+E	0
6	5+NW	5+N	5+W	5	0+FN	0+N	0+F	0	3+F	3+FW	3	3+W
7	4+N	4+NE	4	4+E	1+FN	1+N	1+F	1	2+FE	2+F	2+E	2

TABLE IV
LOOKUP TABLE FOR VERTEX NEIGHBORING RULE SUBSET IN AN OCTREE STRUCTURE

	Direction							
Initial Digit Value	FNE	FNW	FSE	FSW	RNE	RNW	RSE	RSW
0	7	7+W	7+S	7+SW	7+R	7+RW	7+RS	7+RSW
1	6+E	6	6+SE	6+S	6+RE	6+R	6+RSE	6+RS
2	5+N	5+NW	5	5+W	5+RN	5+RNW	5+R	5+RW
3	4+NE	4+N	4+E	4	4+RNE	4+RN	4+RE	4+R
4	3+F	3+FW	3+FS	3+FSW	3	3+W	3+S	3+SW
5	2+FE	2+F	2+FSE	2+FS	2+E	2	2+SE	2+S
6	1+FN	1+FNW	1+F	1+FW	1+N	1+NW	1	1+W
7	0+FNE	0+FN	0+FE	0+F	0+NE	0+N	0+E	0

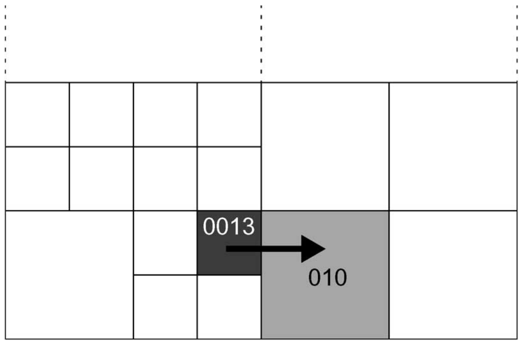

Fig. 7. Searching for a cell whose address corresponds to a higher resolution level than that of the defined neighbor cell.

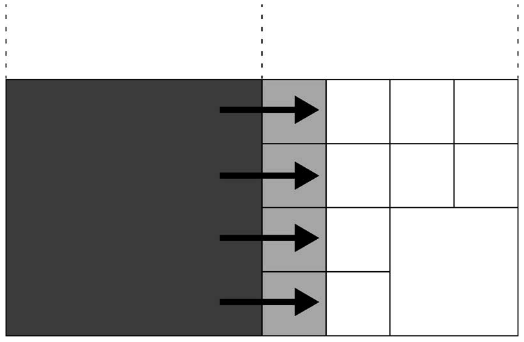
Fig. 8. Example of a cell with several face neighbors in the E direction.

since it represents the closest available parent of cell 0102, which is the address of the former being encoded by default in the address of the target cell.

The problem is more complex if, in the region of interest, the Cartesian grid is subdivided into a higher resolution level than that of the starting cell. This situation occurs when a given cell has several edge neighbors in a given direction, as shown in Fig. 8. In such a case, the address provided by the neighboring rules does not have enough digits to drive the access function up to the terminal leaves. However, determining a neighboring relationship just by a starting point and a direction of displacement does not provide any means to identify which neighbor cell should be selected as the "best" neighbor in a particular application such as, for instance, free path localization. As this task heavily depends on the requirements of the application, it is better defined in the application's algorithm, which also provides the access function to the model with a neighbor selection criterion that is suitable for the application. For instance, in a collision-free path planning application based on occupancy grids, the selection would be made based on the lowest risk of finding an obstacle among the four neighbor candidates identified in the example of Fig. 8.

However, a restricted number of addresses are valid candidates as neighbors of a given starting cell. Fig. 9 illustrates this phenomenon for a 2-D Cartesian grid. In this example, we observe that only cells 0300 and 0301 can be the neighbors of cell 012 in the N direction. In other words, only the children 0 and 1 of the following resolution levels are valid neighbors when one moves toward the N direction. Similarly, the NE neighbor of cell 012 can only be the cell 03100, that is, the

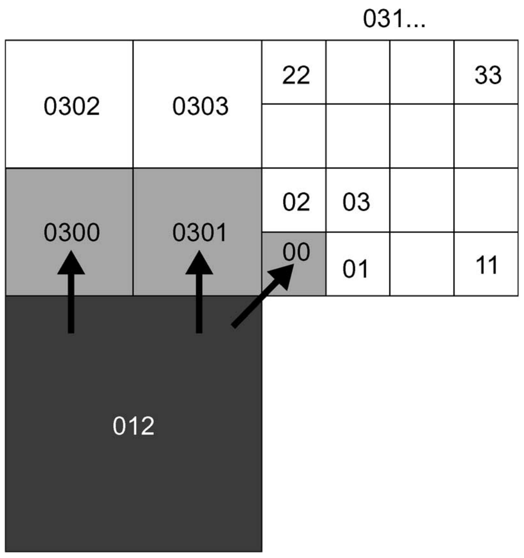

Fig. 9. Particularity of neighboring with multiple neighbors in the N and NE directions.

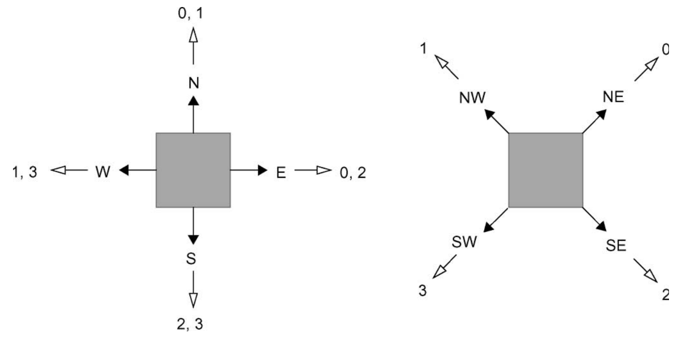

Fig. 10. Validating rules for multiple neighbors in 2-D space determining valid candidate children at a higher resolution level for each direction.

successive children 0 and 0 of the following resolution levels when one moves in the NE direction.

These observations depend on the selected addressing scheme, but only the numerical values are affected, while the general principle remains the same. Validating rules are, therefore, added to the elementary set of neighboring rules to help in dealing with such situations. Fig. 10 presents this limited set of extra rules for multiple neighbors in 2-D space (quadtree) for the addressing scheme defined in Section II. For each direction of displacement, the valid addresses of neighbor cells are indicated. The numbers correspond to the children of the next resolution level that can be considered as neighbors in a given direction. These rules only depend on the direction of displacement and are related neither to the initial resolution level nor to the initial digit values as for the neighboring rules defined previously. Fig. 11 presents similar validating rules for multiple neighbors in 3-D space (octree).

In relation with the example in Fig. 9, where the N direction neighbors from cell 012 are to be identified, the three left-most digits (030) of the neighbor's address are computed by the standard arithmetic procedure, while the set of extra rules provides the values for the fourth digits that are valid along that direction, that is, 0 and 1. Therefore, the resulting neighbor candidates addresses are, respectively, 0300 and 0301. Starting

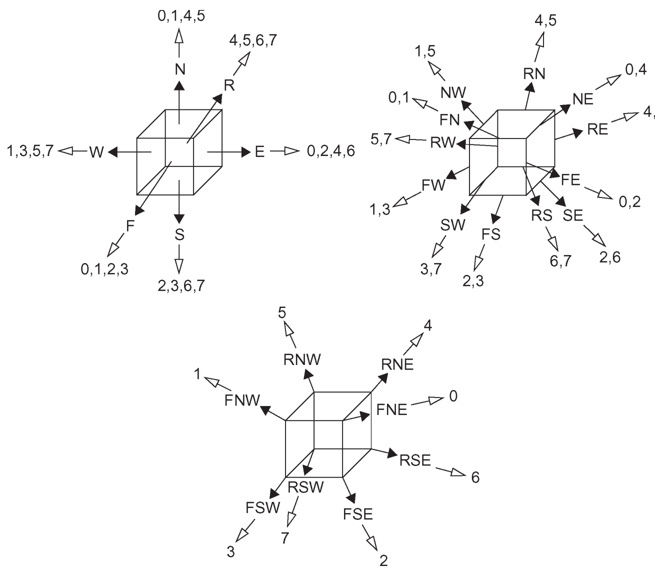

Fig. 11. Validating rules for multiple neighbors in 3-D space determining valid candidate children at a higher resolution level for each direction.

from the root node, a descent in the tree structure allows us to verify their existence and whether they are further subdivided, in which case, the extra rules apply recursively to add an extra digit to both addresses in order to determine their valid children cells 03000, 03001, 03010, and 03011 as immediate neighbors to 012 in the N direction. This recursive procedure continues until the highest level of resolution is reached in the related branch of the tree structure.

Similarly, the immediate NE direction neighbor address from cell 012 is computed to be 031 through the arithmetic procedure. This seed address is then padded twice to the right by the extra validation rules with 0 and 0, respectively, as the descent in the tree structure indicates that subsequent divisions exist and requests the address for the valid children cells that are encoded as pointers. The descent finishes when leaf 03100 is reached, which is the highest resolution level leaf encountered in that branch of the tree structure.

B. *Neighbors Along Model Boundaries*

Special care must also be taken when neighboring rules are applied along the model boundaries. Cells outside these boundaries do not exist because they are not contained in the bounded space that is modeled. To illustrate such a situation, we might consider the cell tagged 002 in the grid of Fig. 4. Searching for a neighbor in the W direction leads outside of the grid. In a similar way, for 3-D grids, no neighbor cells exist in the R and E directions from the cell tagged 01, as shown in Fig. 3.

Such problems are easy to detect in a geometric model. However, as mentioned previously, a quadtree or an octree does not contain explicit information about the geometric location of a given cell in Cartesian space. These data are rather encoded into the tree structure. Nevertheless, attempting to reach a cell that is located outside of the model corresponds in all cases to try to access a cell that is a neighbor of the global mother cell that delimits the entire model. Such an operation would consist of replacing the value 0 (main mother cell) of the left-most digit in an address. The current implementation preempts such an operation on the left-most digit and rather generates an error flag, indicating that there is no neighbor in the specified direction.

Another alternative would consist of adding supplementary rules to connect opposite sides of the model to each other, which is similar to the wraparound model resulting from the normalization operation required by the definition of dilated integers proposed by Schrack [16]. With such rules, searching for a cell that is located outside of the model would lead to the address of a neighbor cell located on the opposite face of the global grid. This could be an interesting strategy and would be easily implemented following the proposed logic. However, for the kind of applications that are of interest here, such a mapping would make the representation incoherent with the geometric reality of the world that is not defined in a circularly symmetrical map but, rather, in a bounded Cartesian world. Therefore, it has not been included in the current implementation. Nevertheless, such an approach could be very useful in applications where the geometric coordinates are angular variables, for instance, path planning in the joint world reference frame for a revolute manipulator robot. This possibility expands the flexibility of the proposed neighbor identification strategy.

VI. Algorithm Complexity

The complexity of the proposed approach only depends on the number of digits to be processed for finding a neighbor cell address in a given direction. In other words, it depends on the number of accesses that are made to the lookup tables containing the rule set. The geometric location of neighboring cells in the grid thus influences the complexity. As shown above, sister neighbors are rapidly identified since no carry is generated. Only one access to lookup tables is sufficient to compute the neighbor cell address completely. In the case of cousin cells, two digits must be changed, and for more distant neighboring relationships, the number of accesses to the lookup tables grows proportionally with the number of levels between terminal leaves and their common ancestor in the tree structure.

Therefore, the maximum complexity of the proposed neighbor-finding algorithm is $O(p-1)$, where p represents the number of resolution levels in the octree that is equal to the number of digits in the addresses for the highest resolution. The left-most digit of the address is never processed as the model is bounded. The maximum number of digits to estimate is therefore limited to $p-1$. As a result, this algorithm is simpler in comparison with classical backtracking approaches that search for a common parent cell before validating the neighborhood relationship for all candidates by repetitively accessing the model [8], [9]. Such classical strategies reach a maximum complexity of $O((2^2)^{p-1})$ for a 2-D model and $O((2^3)^{p-1})$ for a 3-D model. On the other hand, the proposed approach compares well with techniques using compact encoding schemes, such as linear quadtrees or dilated integers, and various algebraic rules, as introduced by Gargantini [15] or Schrack [16], to achieve proportional $O(p)$ or even constant-time operation, respectively. However, the latter methods mainly address neighbor search over grids of a single and uniform resolution representing only discrete-space parameters, such as black and white pixels. This imposes limitations on the content and distribution of the model that are not immediately compatible with probabilistic mapping. The propose scheme addresses these issues and provides a robust framework for multiresolution continuous-space representations that are more adequate for mapping environments with various levels of complexity while offering similar computational performance to state-of-the-art approaches.

The complexity of the proposed framework remains independent of the dimensionality of the model (2-D, 3-D, or more) since neighbor search is entirely computed on virtual addresses that do not have extra digit positions for additional spatial dimensions. One advantage of the arithmetic solution comes from the fact that adding one level of resolution to the model only adds one digit to all addresses, while extending the dimensionality of the model only changes the range of possible digit values but not their number. Taking advantage of this fact and manipulating addresses by only accessing lookup tables drastically limit the computational explosion typically associated with the manipulation of 3-D representation. Although the complexity of the lookup tables grows with the number of dimensions, the implementation of the proposed arithmetics is straightforward, being limited to the encoding of one static lookup table (Table I) for quadtrees or three static lookup tables (Tables II–IV) for octrees.

VII. Applications

Such an algorithm finds numerous applications in computer vision and robotics. The context in which it has been designed is concerned with collision avoidance for an autonomous robot. Quadtrees and octrees are used to model the occupancy probability of regions of space [22], [23]. Using the proposed set of rules for computing the addresses of neighboring cells offers an efficient way to directly identify paths of empty space connected with a given starting point (a given cell), no matter the complexity of the scene encoded in the 2-D or 3-D virtual representation. Starting from the actual configuration, the path planning algorithm successively selects, among the candidates identified by neighbor search, cells containing the lowest occupancy probability and progressively builds a collision-free trajectory for the robot [24]. Such a process must be repeated a very large number of times in a path planning operation with collision avoidance to successfully connect volumes of space through which all components of the structure of a robot can safely circulate. Fig. 12 presents two examples of the path planning operation based on free space identification in 2-D space for a mobile robot and for a planar 4 degrees-of-freedom manipulator. Two-dimensional space is considered here for the sake of clarity; similar results were obtained in 3-D space. We observe that an efficient neighbor technique directly influences the performance of the path planning phase through the determination of a proper sequence of movements.

The application context of path planning was used to validate the correctness of the approach and of the implementation. The critical point here is to ensure that the definition of rules and their implementation in lookup tables are accurate and in accordance with the selected addressing scheme, but since the

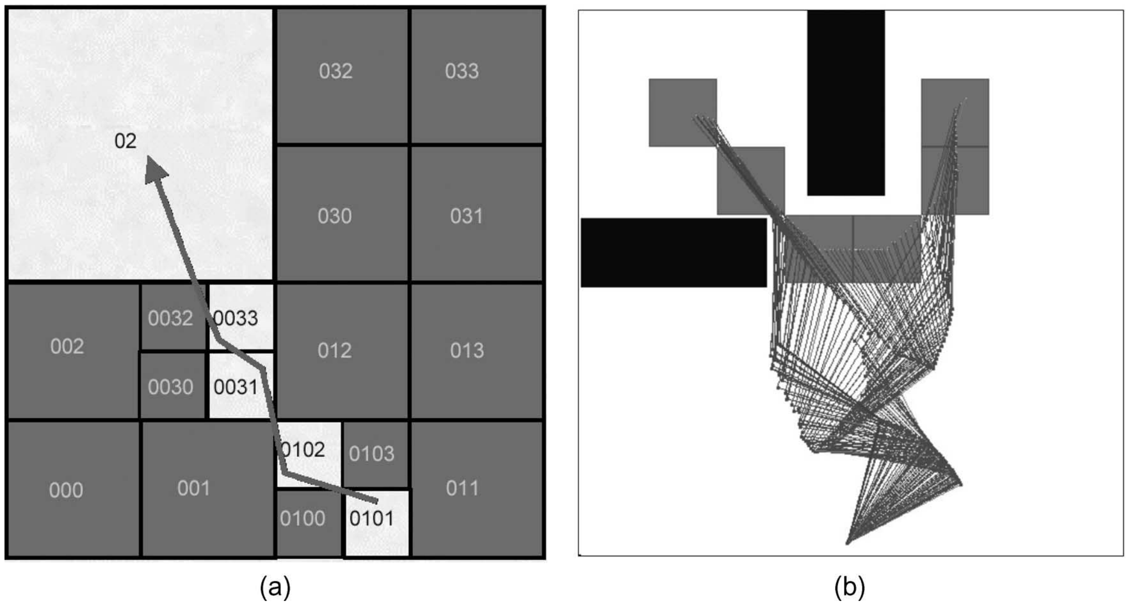

Fig. 12. Path planning with collision avoidance for (a) a mobile robot and (b) a 4-degree-of-freedom planar manipulator.

TABLE V
COMPUTATION TIME COMPARISON BETWEEN THREE NEIGHBOR-FINDING SCHEMES

Environment configuration	Computation time for complete neighbor search (ms)		
	Backtracking	Sequence encoding	Address-based arithmetic
a)	139 060	16 010	16 043
b)	141 490	17 680	18 620
c)	38 550	4 570	3 920
d)	51 040	4 680	4 340
e)	17 690	2 608	2 640

principle is somewhat repetitive, with only variations in the labels following the direction of the search and the level of neighboring between adjacent cells, this procedure is simplified. Our implementation was validated by simulating several progressions in 2-D and 3-D space and visually inspecting the correctness and connectivity between cells in the sequences that were generated.

Moreover, in order to properly monitor the performance of the proposed scheme, experimental evaluation was conducted on several 2-D and 3-D environment configurations over which a collision-free path was computed using 1) an implementation of the classical backtracking neighbor search proposed by Samet [8], [9], 2) an implementation of a sequence-based occupancy model performing bits manipulation inspired from the linear quadtrees scheme [15], and 3) the proposed address-based arithmetic approach in combination with a pointer-based tree structure. While sequence-based encoding allowed to operate on more compact models than the others, the focus of the present evaluation was put on computational efficiency. Table V summarizes the measured computation time needed to generate a complete neighbor sequence of empty space cells as obtained with each of the neighbor search techniques for a subset of 2-D environment configurations that are presented in Fig. 13. Black areas represent obstacles to avoid, while white areas are safe regions to navigate. Apparent blurred regions around obstacles correspond to zones with uncertainty on the occupancy state.

From these results, similar variations in computation time are observed for all neighbor search approaches according to the length of the path and complexity of the configuration in the environment. However, general trends clearly show that the proposed framework offers comparable computation performance with the sequence-based encoding technique that also uses heuristic schemes for adjacency determination. The proposed arithmetic performs considerably faster than the traditional backtracking strategy. On average, over our entire set of tests, a reduction of 85% in computation time was observed when the proposed scheme was compared to backtracking operation, while a variation of only 2.5% was noticed when compared with the sequence-based neighbor search [23]. These experimental observations validate our theoretical complexity analysis and follow our expectations.

Beyond immediate neighbor cell identification as considered in our experiments, the computation of distance maps, which are widely used in robotics and pattern recognition, can also be processed with the neighboring rules presented here. As the distance is usually computed as a propagation wave successively transmitted from one cell to its neighbors [25] to lead to a diagram of safe free space [26], direct identification of

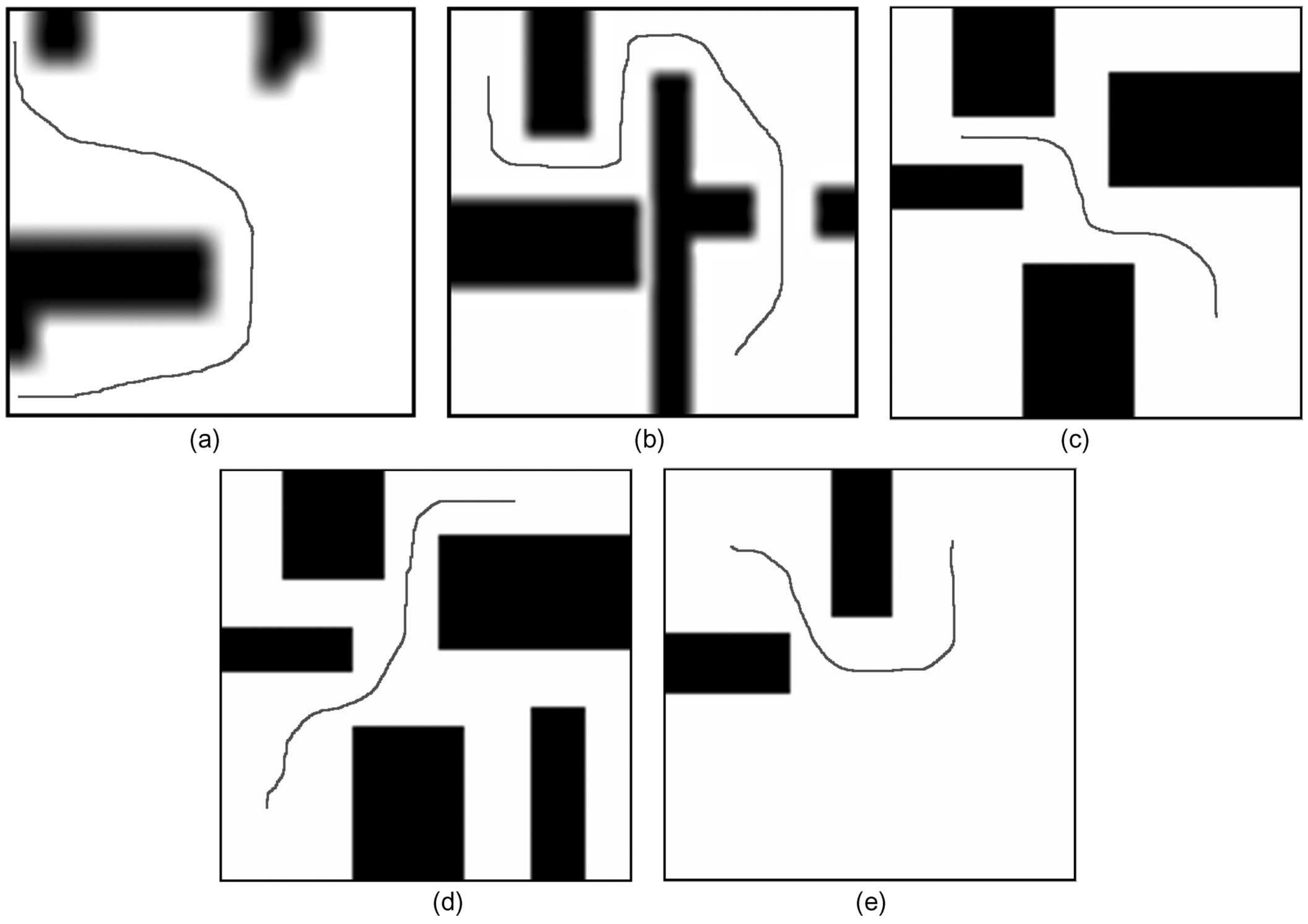

Fig. 13. Environment configurations for neighbor search performance evaluation.

neighbor cells is extensively required. The approach can also find important applications in medical imaging as octrees are becoming more popular for 3-D representations of complex organs [27]. In this case, neighboring rules can help in speeding up the estimation of various parameters that characterize the organs under examination and eventually lead to more accurate diagnoses.

VIII. Conclusion

In this paper, a framework that provides fast neighbor cell identification in multiresolution quadtree or octree models has been presented. Encoding a spatial model as a compact quadtree or octree results in the loss of the explicit geometric relationship between the cells. This leads to difficulties in spatial displacement management that can be overcome using the proposed approach. Relying on a hierarchical addressing scheme of cells and a set of algebraic rules encoded as lookup tables to optimize performance, the technique provides a reliable and efficient displacement strategy within multiresolution quadtrees or octrees for a straightforward identification of free space in robotic guidance tasks.

The algorithm is independent of the dimensionality of the model, while its average performance evolves with the number of resolution levels present in the model, which depends on the desired accuracy of the representation and on the geometrical distribution of objects in 2-D or 3-D space. However, as the proposed framework is able to take full advantage of multiresolution encoding by dealing with multiple neighbors and boundary effects, it provides a valuable solution to preserve model compactness and low processing overhead without requiring any coding conversion steps from occupancy grids that are readily available from typical range-sensing systems.

Accurate neighbor identification is also achieved without geometrical validation of the actual neighboring status of identified candidate cells. The information contained in the model can be directly accessed by relying only on the addresses computed, provided that a corresponding pointer-based tree structure is used. Finally, the approach does not require any supplementary information to be explicitly added in the model and is not limited to encoding discrete data; thus it is compatible with probabilistic maps.

The complexity is similar to other adjacency determination approaches that also rely on variations of an addressing scheme but often operate through an intermediate compact encoding of the model, which is not directly suitable in a robotic context given the dynamic component. Testing with 2-D and 3-D multiresolution probabilistic representations for the manipulator's path planning with collision avoidance demonstrated that the computational explosion usually related with the addition of the third dimension in the model of the environment can be avoided while providing reliable sequences of physically connected cells for the robot to traverse. Although the algorithm has been developed in the context of collision-free path planning for autonomous robots, numerous other applications can be envisioned, especially in the fields of computer vision, visualization, and modeling.

ACKNOWLEDGMENT

The author would like to thank M. Soucy for his collaboration in testing the proposed framework and the anonymous reviewers for providing insightful comments.

REFERENCES

[1] A. Klinger, "Patterns and search statistics," in *Optimizing Methods in Statistics*. New York: Academic, 1971, pp. 303–307.
[2] H. H. Chen and T. S. Huang, "A survey of construction and manipulation of octrees," *Comput. Vis., Graph. Image Process.*, vol. 43, no. 3, pp. 409–431, Sep. 1988.
[3] R. Shu and M. S. Kankanhalli, "Efficient linear octree generation from voxels," *Image Vis. Comput.*, vol. 12, no. 5, pp. 297–303, Jun. 1994.
[4] M. C. Martin and H. P. Moravec, "Robot Evidence Grids," Robotics Inst., Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-RI-TR-96-06, 1996.
[5] H. Samet, "Region representation: Quadtrees from boundary codes," *Commun. ACM*, vol. 23, no. 3, pp. 163–170, Mar. 1980.
[6] ——, *The Design and Analysis of Spatial Data Structures*. Reading, MA: Addison-Wesley, 1990.
[7] ——, *Applications of Spatial Data Structures*. Reading, MA: Addison-Wesley, 1990.
[8] ——, "Neighbor finding techniques for images represented by quadtrees," *Comput. Vis. Graph. Image Process.*, vol. 18, no. 1, pp. 37–57, 1982.
[9] ——, "Neighbor finding in images represented by octrees," *Comput. Vis. Graph. Image Process.*, vol. 46, no. 3, pp. 367–386, Jun. 1989.
[10] J. E. Besançon, *Vision par Ordinateur en Deux et Trois Dimensions*. Paris, France: Eyrolles, 1988, pp. 341–353.
[11] D. H. Ballard and C. M. Brown, *Computer Vision*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
[12] A. Klinger and M. L. Rhodes, "Organization and access of image data by areas," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-1, no. 1, pp. 50–60, Jan. 1979.
[13] G. M. Hunter and K. Steiglitz, "Operations on images using quad trees," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-1, no. 2, pp. 145–163, Apr. 1979.
[14] I. Gargantini, "Linear octrees for fast processing of three-dimensional objects," *Comput. Graph. Image Process.*, vol. 20, no. 4, pp. 365–374, Dec. 1982.
[15] ——, "An effective way to represent quadtrees," *ACM Commun.*, vol. 25, no. 12, pp. 905–910, Dec. 1982.
[16] G. Schrack, "Finding neighbors of equal size in linear quadtrees and octrees in constant time," *CVGIP, Image Underst.*, vol. 55, no. 3, pp. 231–239, May 1992.
[17] C. Y. Huang and K. L. Chung, "Fast operations on binary images using interpolation-based bintrees," *Pattern Recognit.*, vol. 28, no. 3, pp. 409–420, Mar. 1995.
[18] ——, "Faster neighbor finding on images represented by bincodes," *Pattern Recognit.*, vol. 29, no. 9, pp. 1507–1518, Sep. 1996.
[19] P. Payeur, "An optimized computational technique for free space localization in 3-D virtual representations of complex environments," in *Proc. IEEE Int. Conf. Virtual Environments, Human-Comput. Interfaces, and Meas. Syst.*, Boston, MA, Jul. 2004, pp. 13–18.
[20] F. Major, J. Malenfant, and N. F. Stewart, "Distance between objects represented by octrees defined in different coordinate systems," *Comput. Graph.*, vol. 13, no. 4, pp. 497–503, 1989.
[21] J. Tremblay, "Modélisation de l'Environnement d'un Robot Mobile et Application à une Architecture de Robot Mobile," M.S. thesis, Laval Univ., Quebec City, QC, Canada, 1996.
[22] P. Payeur, D. Laurendeau, and C. M. Gosselin, "Range data merging for probabilistic octree modeling of 3-D perturbed workspaces," in *Proc. IEEE Int. Conf. Robot. and Autom.*, Leuven, Belgium, May 1998, vol. 4, pp. 3071–3078.
[23] M. Soucy, "Manipulator path planning with multi-resolution potential fields and fuzzy logic control," M.S. thesis, Univ. Ottawa, Ottawa, ON, Canada, 2005.
[24] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *IEEE Computer*, vol. 22, no. 6, pp. 46–57, Jun. 1989.
[25] D. Jung and K. K. Gupta, "Octree-based hierarchical distance maps for collision detection," in *Proc. IEEE Int. Conf. Robot. and Autom.*, Nagoya, Japan, Apr. 1995, pp. 454–481.
[26] O. Takahashi and R. J. Schilling, "Motion planning in a plane using generalized Voronoi diagrams," *IEEE Trans. Robot. Autom.*, vol. 5, no. 2, pp. 143–150, Apr. 1989.
[27] P. Kochunov, J. Lancaster, P. Thompson, A. Boyer, J. Hardies, and P. Fox, "Evaluation of octree regional spatial normalization method for regional anatomical matching," *Hum. Brain Mapp.*, vol. 11, no. 3, pp. 193–206, Nov. 2000.

Pierre Payeur (S'90–M'98) received the Ph.D. degree in electrical engineering from the Université Laval, Quebec City, Canada, in 1999.

In 1998, he joined the University of Ottawa, Ottawa, ON, Canada, as an Assistant Professor in the School of Information Technology and Engineering (SITE) and co-founded the Vision, Imaging, Video and Autonomous Systems Research Laboratory. His current research interests are volumetric 3-D modeling, range data processing, robot guidance, teleoperation, and integration of computer vision in autonomous systems control.

Dr. Payeur is a member of the IEEE Robotics and Automation Society, the IEEE Instrumentation and Measurement Society, and of the Ordre des Ingénieurs du Québec.