COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING 39, 167–185 (1987)

Octrees of Objects in Arbitrary Motion: Representation and Efficiency

JUYANG WENG AND NARENDRA AHUJA

Coordinated Science Laboratory, University of Illinois, Urbana, Illinois 61801

Received November 8, 1985; revised May 23, 1986

An algorithm is presented that updates the octree of a three-dimensional object after the object has undergone arbitrary rotation and translation. The algorithm moves each of the black leaves and then adds the subtree corresponding to the displaced black leaf. The algorithm uses a computationally efficient method to detect the intersections of moved blocks with octree tessellation. On the average, the space and time complexities of the algorithm are both bounded by $O(Kn)$, where K is the number of nodes in the source tree and n is the logarithm of the side-length of the universe space. The quantization errors are prevented from accumulating by maintaining a compact source octree. Each incremental displacement is performed with respect to the source tree. Implementation results are given for the motion of synthetic three-dimensional objects. Upper bounds are derived for the numbers of nodes in an octree representing a cubic block of arbitrary orientation and position. These bounds are proportional to the face area of the cube plus the logarithm of the side length of the universe space.

1. INTRODUCTION

Octrees represent the space occupied by objects using a cubic decomposition of the universe space. The space is recursively partitioned into octants until each octant is completely inside or outside an object, or the limit of resolution is reached. The final partition is represented by an 8-ary tree, or octree, whose leaf nodes identify the occupied and unoccupied space [10, 16, 20]. Octree is a useful representation of three-dimensional objects for space planning, computer animation, and navigation [2, 7, 10, 13].

Major characteristic of octrees is the use of volumetric primitives (cubes) of fixed sizes, locations and orientations to tessellate the universe space. The universe-centered definition of the cubes and the use of a fixed shape, nonoverlaping primitives help in reducing the complexity of many common spatial operations on objects, e.g., intersection detection and volume computation [4, 9, 11, 14, 17, 18]. However, this efficiency comes at the cost of the representation becoming very sensitive to object location and orientation. For example, if an object moves, it occupies different cells of the cubic tessellation and as a result its octree may change drastically.

In object centered representations, the placement of the primitives is determined by the placement of the objects to be represented. Each object has an associated cluster of primitives whose locations are specified with respect to the object location. The union of the primitives in the cluster, of course, is the volume of the object. Such representations include the medial axis transformation [16] and piecewise approximation [1], and they can be easily updated to represent moving objects. However, they are not well suited for computations such as intersection detection and volume computations. Octree may also be viewed as object centered representa-

0734-189X/87 $3.00

tion if a universe space is defined with respect to each object in the scene. Thus, representing many objects may require multiple octrees, each with respect to a different universe space. This, in general, results in a larger fragmentation of the object because of the rigidity of the relative locations and the sizes of the primitives compared to the other object-centered representations. The compactness characteristic of the object centered representation is not achieved but the computational efficiency of the universe centered octree is lost. Therefore, an object centered octree is not a useful representation here.

In robotic manipulation of environment, representations of dynamic scenes are necessary to design collision-free and efficient trajectories for object movement. Trees must be continuously obtained to reflect varying positions of objects with respect to fixed axes. Thus, starting with the octree obtained for a given configuration [5, 12, 19, 21], updating the tree is necessary to represent moving objects. For the case of pure translation of the object [3, 13, 15], the problem of octree updating is relatively manageable, especially if axial translation magnitudes are integral multiples of the dimension of the smallest blocks used. However, because of the anisotropic nature of the primitives, arbitrary rotation of objects usually results in a large fragmentation or compaction of the octree. Some difficulties encountered in arbitrary rotation are discussed in Ahuja and Nash [3]. A brief reference to the problem can also be found in Meagher [13]. Jackins and Tanimoto [10] discuss the simpler problem of rotation by integral multiples of 90°.

In this paper we present an efficient algorithm to update the octree for arbitrary translation and rotation of the represented objects. The source octree is traversed and each leaf node is subject to the desired motion. Octree nodes corresponding to the block at its new position are identified by a computationally efficient method. As different leaves are moved, the evolving tree may require compaction which is performed at the earliest time. The average space and time requirements of the algorithm are bounded by $O(Kn)$, where K is the number of nodes in the source tree and n is the logarithm of the side-length of the universe space. Implementation results show very few superfluous nodes are generated in the intermediate steps. Upper bounds are derived for the numbers of different nodes in an octree representing a cubic block of arbitrary orientation and position. These bounds are proportional to the face area of the cube plus the logarithm of the side length of the universe space.

Section 2 reviews the octree representation with some definitions. Section 3 discusses arbitrary motion and general problems associated with updating an octree for arbitrary motion of the objects. Section 4 presents the algorithm. Section 5 analyzes the performance and complexity of the algorithm and derives some upper bounds on numbers of nodes in octree. Section 6 discusses an implementation of the algorithm. A summary follows in Section 7.

2. OCTREE REPRESENTATION

We will first define the octree representation of three-dimensional objects. Consider a $2^n \times 2^n \times 2^n$ array of unit cubic volume elements (voxels or unit cubes) called the voxel array, where n corresponds to the size of space or resolution if space is regarded as fixed. Only objects within the universe space are represented by the octree. To represent a three-dimensional object each voxel is assigned a color, black

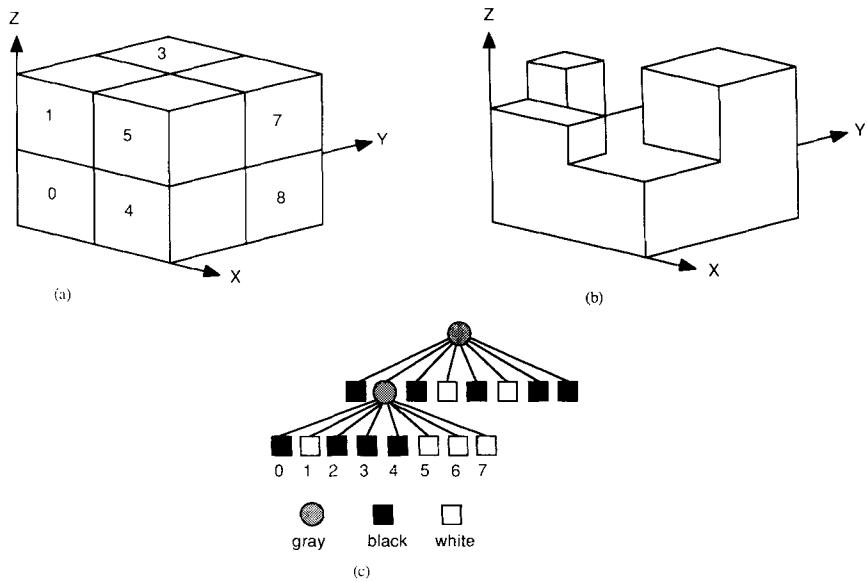

FIG. 1. An object and corresponding octree. (a) Labels of octants. (b) Object. (c) Octree.

or white, according to whether that voxel is inside or outside the object. The object space is represented by the space occupied by all black voxels. Since there may be partially filled voxels, the object can be only approximately represented by the voxel array at a given resolution. In the octree, the root node corresponds to the universe space. The space is divided into eight octants each of side-length 2^{n-1} labeled $0.1, \ldots, 7$, as shown in Fig. 1a. These octants correspond to eight children nodes of the root. Each octant is recursively subdivided further into octants, until each octant contains only voxels of a single color. Their corresponding nodes have no children and are called leaf nodes. All leaf nodes are assigned the color of their corresponding voxels. All nonleaf nodes are called internal nodes and are colored gray. A cube of side-length 2^k is called k-cube. Figure 1b shows a three-dimensional voxel array and Fig. 1c shows the corresponding octree.

Let us define the level of the root as n. If the parent is at level k its children are at level $k - 1$. The lowest possible level is zero corresponding to voxels. A cube is said to be upright if its edges are parallel to coordinate axes, otherwise it is said to be tilted.

3. GENERAL MOTION

Updating an octree for general, arbitrary motion of the represented object involves updating the tree for all black leaf nodes or cubes. Let each cube be characterized by its coordinates. Then, any arbitrary motion of a cube can be characterized by a rotation component and a translation component. Let X and X' be the coordinate (column) vectors of a cube at time instances t_1 and t_2, respectively, T be the translation vector, and R be a three-dimensional rotation matrix for

rotation by angle ϕ about the unit vector $n^0 = (n_x, n_y, n_z)$ through the origin. Then,

$$X' = RX + T$$

where

$$R = \begin{bmatrix} (n_x^2 - 1)(1 - \cos\phi) + 1 & n_x n_y (1 - \cos\phi) - n_z \sin\phi & n_x n_z (1 - \cos\phi) + n_y \sin\phi \\ n_y n_x (1 - \cos\phi) + n_z \sin\phi & (n_y^2 - 1)(1 - \cos\phi) + 1 & n_y n_z (1 - \cos\phi) - n_x \sin\phi \\ n_z n_x (1 - \cos\phi) - n_y \sin\phi & n_z n_y (1 - \cos\phi) + n_x \sin\phi & (n_z^2 - 1)(1 - \cos\phi) + 1 \end{bmatrix}.$$

Since octrees are defined in terms of a fixed tessellation of space, the resulting representation is very sensitive to object motion. This is because of the rigid anisotropic shape of (cubic) primitives and a finite resolution. When an object is translated, its updated octree must represent arbitrary intersections of upright cubes. This, in general, results in fragmentation or compaction of tree nodes, and quantization error from partially filled voxels. However, when objects are also allowed to rotate arbitrarily, updating the octree requires representation of intersections of arbitrarily oriented cubes. The nonrectalinear shape of the regions of intersection makes the above quantization effect much more severe. While for translation one can avoid the problem by performing translations with integral axial components [3], no such solution is possible for the case of rotation, except for restricting rotation to multiples of 90° [10]. Worse still, if an object is incrementally rotated through a sequence of angles, and the corresponding octrees are progressively updated, the quantization effects accumulate since the quantization losses and gains in volume would not preserve object shape. For example, a rotation by an angle ϕ followed by an angle $-\phi$ may not result in the same tree. Thus the octree may undergo cumulative distortion as an increase of rotations is performed. Our solution to this consists in keeping a compact octree representation of the object called source tree as a reference. Each consecutive motion is transformed into a displacement with respect to the source tree. For example, if we want to perform another motion from X' to X'', characterized by rotation matrix R' and translation vector T', i.e.,

$$X' = RX + T \quad \text{and} \quad X'' = R'X' + T',$$

then

$$X'' = (R'R)X + (R'T + T').$$

Thus, the displacement by R and T followed by another displacement by R' and T' is performed as a single displacement of the source tree by the rotation matrix $R'R$ and translation vector $R'T + T'$.

After a motion, a cube may occupy a position where it partially overlaps with voxels. To obtain an octree representation for the cube after motion, some rounding

must be considered. If all the partially occupied voxels were colored white, it would reduce the volume of the cubes. Furthermore, the result of moving an object's cubes corresponding to the leaf nodes, accompanied by such rounding of partially occupied voxels, might create holes in the moved object. (For example, consider rotating a unit cube by 45° about the x axis. After rotation it may partially occupy six voxels, each of which would be colored white!) These holes reduce node compaction and result in an octree with many nodes. If all the partially occupied voxels are colored black, the volume would tend to increase. In the algorithm presented in this paper, a partially occupied voxel is colored black if and only if its center is on or inside some displaced black cube. If a voxel is completely inside the displaced object, its center must be on or inside some displaced cube therefore it is colored black. Thus, no nonexistent inside holes would be created after motion. This rounding scheme also tends to keep the volume unchanged compared to the other two schemes mentioned.

4. ALGORITHM

Given rotation matrix R and translation vector T, the algorithm constructs a target octree T_2 from source octree T_1. To define T_2, first the occupancy of the discrete three-dimensional space, i.e., the three-dimensional voxel array is defined after motion. Each voxel of the array is treated black if and only if its center is on or inside some displaced black cubes from source tree. The algorithm to construct T_2 traverses T_1 in post order. Once a black leaf node is found in T_1 the new position of the corresponding cube after displacement is calculated. The displaced cube is tested for intersections with the upright cubes to determine the color of the corresponding nodes in T_2. Each partially intersecting upright cube is recursively subdivided until all the voxels included in this upright cube are of single color, possibly after rounding. Node condensation is performed at the earliest opportunity after a group of eight identically colored children is generated. After condensation the eight children are deleted and their parent is given their color.

The main steps of the algorithm are as follows:

1. Traverse the source tree. For each black leaf node encountered, perform step 2.

2. Obtain the new position and orientation of the cube corresponding to the leaf node under consideration in T_1, which is generally no longer upright because of motion. If this new position is outside of the universe space, report error and stop. Otherwise, perform step 3.

3. Starting from the root of the target tree, generate the nodes of the target tree by testing their corresponding upright cubes for intersection with the moved cube. For each test there are three cases to consider, each leading to a different decision about the nature of the target tree node:

(1) no intersection: the upright cube is unoccupied and is left alone.

(2) the upright cube is inside the displaced cube: the upright cube is occupied. Color corresponding node black.

(3) otherwise: generate all eight children of the upright cube (node). Perform step 3 recursively for each child. If all the eight children are of a single color (white or black), delete the children and make the parent the same color.

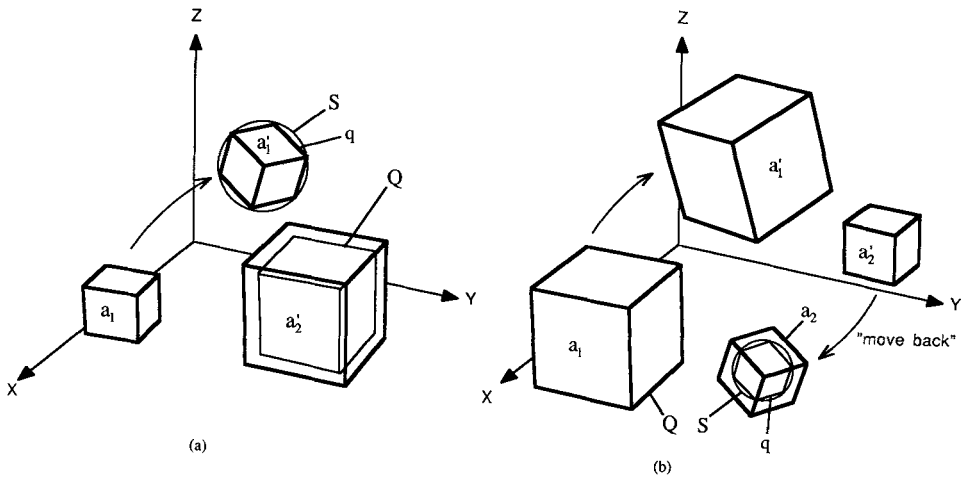

FIG. 2. Relations between object and octree cubes.

Intersection Detection

The main step in the above algorithm is that of intersection detection. Consider a black leaf node or the corresponding cube a_1 of source tree T_1 at level l_1. It is moved to a new position where it occupies the space corresponding to a tilted cube a_1' as shown in Fig. 2. To construct the nodes in target tree T_2 corresponding to a_1', we must test the intersection of the tilted cube a_1' with the upright cubes. The relationship between upright cube a_2' in the target tree and a_1' has three cases. We denote these three cases by relation A.

Relation A (a_2' vs a_1').

1. Inside: All the centers of the voxels in a_2' are inside a_1'.
2. Outside: No centers of any voxels in a_2' are inside a_1'.
3. Partial: Otherwise.

In case 1, a_2' should be colored black. In case 2, a_1' does not affect the color of a_2', however, a_2' might become black when other black leaves of T_1 are considered. In case 3, a_2' generates eight children and each child is recursively tested for intersection with a_1'.

Case 2 of relation A is computationally expensive. It involves the computation of the three cases of partial intersection shown in Fig. 3. A modification can be made to make intersection testing efficient. Let q and Q be the smaller cube and the bigger cube, respectively (see Fig. 4). To test the intersection of q with Q, we consider the smallest sphere S that contains the smaller cube q. If we use S to represent q and regard the intersection of sphere S with Q as that of q with Q, some nonintersections of q with Q would be regarded as partial intersections as

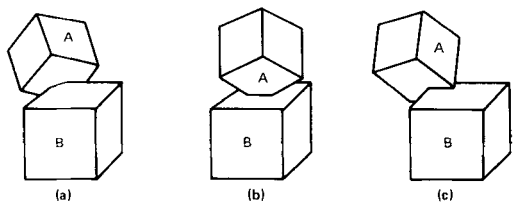

FIG. 3. Examples of intersection.

shown in Fig. 4. This means that some of the case 2 situations in relation A are incorrectly viewed as case 3. Though this results in some unnecessary children generation followed by condensation, the computation is still much more efficient than direct testing of relation A. Assume that a_1 is at level l_1 and a_2' is at l_2. To always represent the smaller cube by a sphere we consider two cases with respect to the sizes (or levels) of a_1' and a_2':

(a) $l_1 < l_2$. The smaller cube q is a_1'. The sphere S circumscribing q has diameter $\sqrt{3}\,2^{l_1}$. To test for intersection between S and a_2', it is sufficient to test if any of the centers of the unit cubes in a_2' are contained in S. This is according to the rounding scheme chosen. We can therefore test for the discrete space intersection between S and a_2', by a true Euclidean domain intersection detection between S and a smaller version Q of a_2'. The side length of Q is a unit smaller than that of a_2', i.e., $2^{l_2} - 1$ as shown in Fig. 2a. Since Q is upright, to test the intersection of S and Q, we just need to check the absolute values of the coordinate differences of their centers. Let *disa* be the value of the sum of the radius of S and half side-length of Q, i.e.,

$$disa = \frac{\sqrt{3}\,2^{l_1} + 2^{l_2} - 1}{2}.$$

Let a_1' be centered at $X_1' = (fcx_1', fcy_1', fcz_1')$, a_2' be centered at $X_2' = (cx_2', cy_2', cz_2')$. Here f and c in the notation refer to *f*orward motion (to distinguish it from backward motion considered later) and *c*enter, respectively. Thus fcx_1'

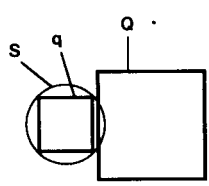

FIG. 4. Intersection of S with Q and non-intersection of q with Q.

denotes the x coordinate of the center after forward displacement. If

$$|fcx_1' - cx_2'| > disa \quad \text{or} \quad |fcy_1' - cy_2'| > disa \quad \text{or} \quad |fcz_1' - cz_2'| > disa, \tag{1}$$

then no centers of any voxels in a_2' can be on or inside the smallest sphere S that includes a_1'. So none of them are on or inside a_1'.

(b) $l_1 \geq l_2$. Here a_2' is smaller than or as big as a_1'. The sphere S here should still substitute the smaller cube instead of the bigger one to reduce the likelihood of false intersection detection. However, the bigger cube a_1' here is tilted. So the inequalities similar to (1) cannot be applied to S and a_1' directly. Fortunately, a_1 is upright and the geometric relationship of a_2' and a_1' is the same as that of a_2 and a_1, where a_2 is the tilted cube "moved backward" from a_2' as shown in Fig. 2b. Thus, if a_1 and a_2 were to move together as a rigid body, then a_1 would move to a_1', and a_2 would move to a_2'. So we consider a_1 and a_2 instead of a_1' and a_2' since this make possible the use of the inequalities similar to (1). The bigger cube Q here is a_1. We must test if Q and a_2 intersect each other. To take into account rounding, as before, we need only check if Q contains any centers of unit cubes contained in a_2 (see Fig. 2b). To do this, we first replace a_2 with the smallest sphere S that contains all the centers of unit cubes in a_2. S has a diameter $\sqrt{3}(2^{l_2} - 1)$. To test the intersection of S with Q we check the absolute values of the coordinate differences of their centers (we have made Q upright!). Let *disb* be the value of the sum of the radius of S and the half side-length of Q, i.e.,

$$disb = \frac{2^{l_1} + \sqrt{3}(2^{l_2} - 1)}{2}.$$

Let $X_2 = (bcx_2, bcy_2, bcz_2)$ be the center of a_2; b stands for "$b$ackward" displacement. We have $X_2' = RX_2 + T$ or $X_2 = R^t(X_2' - T)$. Let (cx_1, cy_1, cz_1) be the coordinates of the center of a_1 (see Fig. 2b). If

$$|cx_1 - bcx_2| > disb \quad \text{or} \quad |cy_1 - bcy_2| > disb \quad \text{or} \quad |cz_1 - bcz_2| > disb, \tag{2}$$

then S and Q are nonoverlapping. Also, we can further test for complete inclusion of a_2 in a_1 by checking if each vertex of q is contained in a_1, where q is a smaller version of a_2, i.e., the smallest cube that contains all the centers of the unit cubes in a_2. The steps (a) and (b) then constitute the modification we set out to obtain a more efficient intersection test than that defined by relation A. We summarize the modified test as relation B.

Relation B (a_2' vs a_1').

(a) $l_1 < l_2$.

1. Inside (a_2' inside a_1'): impossible.
2. Outside (Q outside S): if

$$|fcx_1' - cx_2'| > disa \quad \text{or} \quad |fcy_1' - cy_2'| > disa \quad \text{or} \quad |fcz_1' - cz_2'| > disa.$$

3. Partial: otherwise.

(b) $l_1 \geq l_2$.

1. Inside (a_2' inside a_1'): if

$$|x_2 - cx_1| \leq 2^{l_1 - 1} \quad \text{and} \quad |y_2 - cy_1| \leq 2^{l_1 - 1} \quad \text{and} \quad |z_2 - cz_1| \leq 2^{l_1 - 1}$$

hold for coordinates (x_2, y_2, z_2) of all eight corners of q (a smaller version of a_2).

2. Outside (S outside Q): if

$$|cx_1 - bcx_2| > disb \quad \text{or} \quad |cy_1 - bcy_2| > disb \quad \text{or} \quad |cz_1 - bcz_2| > disb.$$

3. Partial: otherwise.

In summary, the intersection detection for an upright cube and a tilted cube is replaced by the intersection detection for a sphere and an upright cube, thus making the computation more efficient.

For each leaf node in T_1, the algorithm checks the inequalities in relation B. If the relation "inside" holds, a_2' is colored black. If the relation "outside" holds. The color of a_2' determined by the traversal and rotation of the nodes in T_1 that are traversed previously is not changed by the current node a_1 under consideration. If the relation "partial" holds, a_2' is recursively subdivided further and relation B is tested for each of its children. When $l_2 = 0$, i.e., level zero of T_2 is reached, then a_2' is colored black if and only if the center of a_2' is on or inside a_1'.

Algorithm Description

The algorithm has five subroutines **fdleaf, intersect, condense, black** and **gray**. The procedure **fdleaf** traverses T_1, the source tree, in post order. For each black leaf node found, it checks whether this leaf node goes out of universe space after displacement. If so, error is reported. Otherwise **fdleaf** calls **intersect** for this leaf node, to add appropriate nodes to T_2.

The procedure **intersect** adds nodes corresponding to the leaf node passed from **fdleaf** by testing relation B starting from root of T_2. If relation B yields "partial," eight children nodes are generated. The procedure **intersect** is called recursively for each child immediately followed by a call to procedure **condense**. The procedure **condense** checks if all eight siblings have the same color. If so, all children are deleted and the parent node of the deleted siblings is assigned the common color of the deleted siblings. The function **black** return true if the node is black. Similarly **gray** return true if the node is gray.

The following is an outline of the algorithm. The notation is straightforward. Inside(node1, node2) and partial(node1, node2) denote the corresponding condi-

tions defined in relation B.

```
PROCEDURE fdleaf(node);
BEGIN
 IF black(node) THEN
   IF out-of-space THEN error-stop
   ELSE intersect(node)
 ELSE IF gray(node) THEN
     FOR each child of node DO
     fdleaf(child)
END;

PROCEDURE intersect(sourcenode, targetnode);
BEGIN
 IF l1 < l2 THEN
   BEGIN
   IF partial(targetnode, sourcenode) THEN
     FOR each child of targetnode DO
     intersect(sourcenode, child) END
 ELSE {l1 ≥ l2}
   IF l2 = 0 THEN
     IF center of targetnode is inside sourcenode
     THEN color targetnode black
     ELSE color targetnode white
   ELSE
     IF inside(targetnode, sourcenode) THEN
        color targetnode black
     ELSE IF partial(targetnode, sourcenode) THEN
        BEGIN FOR each child of targetnode DO
             intersect(sourcenode, child);
             condense(targetnode)
        END
END;
```

5. ANALYSIS

One way to reduce the severe space requirements of an octree is to avoid explicit leaf nodes. Information about the leaf children of a parent can be compactly stored as a bit pattern in the color field of the parent. The following lemma says this scheme stores only about one eighth of the total nodes. This is a considerable reduction of space requirement.

LEMMA. *Let T and I be the total number of nodes and the total number of the internal nodes, respectively, then $I = (T - 1)/8$.*

Proof. See Appendix A.

Since octree representation uses upright cubes and fixed positions as primitives, different orientations and positions of an object result in different degrees of object fragmentations, and hence in different numbers of nodes within octree. To analyze the space requirements of our algorithm, (actually of any octree algorithm) we need

an upper bound on the number of nodes in an octree representing a cube of any size, orientation and position. The following theorem tells how many black leaf are sufficient to represent a cube of arbitrary orientation and position.

THEOREM 1. *Let B_m be the number of black leaves in an octree representing a cube of side-length 2^m at arbitrary orientation and position, then*

$$B_m \leq 24.25 \cdot 4^m - 200 \cdot 2^m + 1454.$$

Proof. See Appendix B.

Theorem 1 not only provides a bound for black nodes but also is useful for deriving Theorem 2, which gives the bound on the number of internal nodes.

THEOREM 2. *Let I_m be the number of internal nodes in an octree representing a cube of side-length 2^m at any orientation and position in $2^n \times 2^n \times 2^n$ space. Then, for $m \geq 3$,*

$$I_m \leq 5.76 \cdot 4^m + 17 \cdot 2^m + 8n + 76m - 277,$$
$$I_0 \leq 8n - 7, \quad I_1 \leq 8n + 12, \quad I_2 \leq 8n + 101.$$

Proof. See Appendix C.

The bound on total number of nodes in the octree representing a cube can be derived from Theorem 2.

COROLLARY. *Let T_m be the number of total nodes in an octree representing a cube of side-length 2^m at any orientation and position in $2^n \times 2^n \times 2^n$ space. Then, for $m \geq 3$,*

$$T_m \leq 47 \cdot 4^m + 136 \cdot 2^m + 64n + 608m - 2215.$$

Proof. From Lemma 1, $T_m = 8I_m + 1$. Corollary follows from Theorem 2. Q.E.D.

The above corollary shows that the number of nodes in an octree representing an m-cube (size: $2^m \times 2^m \times 2^m$) is bounded by $O(4^m + n)$, that is, the number is at most proportional to the surface area of the cube plus the logarithm of the side-length of the space. (By an analogous analysis, it can be shown that number of nodes in a quadtree representing a square at any orientation and position is at most proportional to the sum of the perimeter of the square plus the logarithm of the image side-length, which is a generalized result compared to [6], which considers only upright squares.)

Theorem 1 and Theorem 2 as well as its corollary show how octree can greatly compact the voxel array representation of a cube. Let us define

$$\rho_B(m) = \frac{\text{number of black leaves}}{\text{volume of an } m\text{-cube}}$$
$$\rho_I(m) = \frac{\text{number of internal nodes}}{\text{volume of an } m\text{-cube}}$$
$$\rho_T(m) = \frac{\text{total number of nodes}}{\text{volume of an } m\text{-cube}}.$$

TABLE 1
Bounds on the Ratios of the Number of Octree Nodes and the Volume of an Arbitrarily Oriented and Positioned Cube

m	$\rho_B(m) \leq$	$\rho_I(m) \leq$	$\rho_T(m) \leq$
8	0.092	0.0228	0.186
10	0.024	0.0057	0.047
12	0.0060	0.0015	0.012
14	0.0015	0.00036	0.0029

Note. See text for definition; cube size: $2^m \times 2^m \times 2^m$, $n = 5m$.

From Theorem 1 we have

$$\rho_B(m) \leq 24.25\left(\tfrac{1}{2}\right)^m - 200\left(\tfrac{1}{4}\right)^m + 1454\left(\tfrac{1}{8}\right)^m.$$

From Theorem 2 and its corollary we get

$$\rho_I(m) \leq 5.76\left(\tfrac{1}{2}\right)^m + 17\left(\tfrac{1}{4}\right)^m + (8n + 76m - 227)\left(\tfrac{1}{8}\right)^m$$
$$\rho_T(m) \leq 47\left(\tfrac{1}{2}\right)^m + 136\left(\tfrac{1}{4}\right)^m + (64n + 608m - 2215)\left(\tfrac{1}{8}\right)^m$$

As can be seen from these expressions these ratios approaches zero very fast as m get large. Table 1 shows the upper bounds of these ratios for some value of m. The actual ratios are even smaller.

The results in Table 1 show that the number of nodes in an octree of an arbitrary oriented cube is much smaller than the number of voxels in the cubes. Each octree node needs a constant times more space than a voxel. However, the saving in the storage space required is dominated by the *order* of the ratio of the number of nodes and the number of voxels for large cubes. The ratios derived above show the *order* of the savings incurred in using the octree representation. These bounds approach zero very fast with increasing m, i.e., the larger the represented cube the larger are the savings over voxel array representation.

For a cube, the above bounds show how the octree can compact the corresponding voxel array. Since cubes are the primitives of octree, these bounds can be used to find a worst case bound and an average bound on the number of nodes for arbitrary objects.

From Theorem 2, in the worst case the space and time have an upper bound of $O(K4^M)$, where K is the number of nodes in the source tree and M is the highest level at which black nodes occur in the source tree.

For an object with surfaces that are all perpendicular to the coordinate axes, the nodes tend to be at higher levels and thus the source tree tends to consist of few nodes. In such a case, the worst case bound is a good estimate. For an object consisting of curved surfaces, the number of nodes in the octree does not change drastically with change in object orientation and position. Thus the source tree has many nodes. In this case, the worst case bound does not give a good estimate of the number of nodes in the target tree. We give below an average bound based on the following assumption for the source tree: the probability of the number of black

nodes at level k is proportional to the number of possible positions at level k, i.e., the number of nodes at level k in a complete tree. For the objects consisting of curved surfaces and nonparallel planar surfaces this assumption could be expected to be approximately true.

THEOREM 3. *Assuming the number of black nodes at level k is proportional to the number of positions at level k, the average time and space of the algorithm is bounded by $O(Kn)$, where K is the number of nodes in the source octree, and n is the logarithm of the side-length of universe space.*

Proof. At level k there are 8^{n-k} positions in a complete octree. The total number of positions for leaves is $\sum_{i=0}^{n-1} 8^{n-i} = (8^{n+1} - 8)/7$. From Theorem 2, the average number of internal nodes traversed is bounded by ($n \geq 3$)

$$\begin{aligned}
&\left(\frac{8^{n+1}-8}{7}\right)^{-1} \sum_{m=0}^{n-1} 8^{n-m} I_m \\
&\quad \leq \frac{7}{8^{n+1}-8}\Bigg[8^n(8n-7) + 8^{n-1}(8n+12) + 8^{n-2}(8n+101) \\
&\qquad + \sum_{m=3}^{n-1} 8^{n-m}(5.76 \cdot 4^m + 17 \cdot 2^m + 8n + 76m - 277)\Bigg] \\
&\quad = \frac{7}{8-8^{1-n}}\Bigg[(8n-7) + 8^{-1}(8n+12) + 8^{-2}(8n+101) \\
&\qquad + 5.76\frac{\frac{1}{2^3} - \frac{1}{2^n}}{\frac{1}{2}} + 17\frac{\frac{1}{4^3} - \frac{1}{4^n}}{\frac{3}{4}} + (8n-277)\frac{\frac{1}{8^3} - \frac{1}{8^n}}{1-\frac{1}{8}} \\
&\qquad + 76\left(\frac{\frac{n-1}{8^{n+1}} - \frac{n}{8^n} + \frac{1}{8}}{\left(1-\frac{1}{8}\right)^2} - \frac{1}{8} - \frac{2}{8^2}\right)\Bigg] \\
&\quad \leq 10n + 1.
\end{aligned}$$

For $1 \leq n \leq 3$ the result still holds. The function **fdleaf** traverses all the internal nodes. Its time and space requirements are bounded by $O(K)$. For each black leaf node in the source tree, **intersect** would visit at most $10n + 1$ ($= O(n)$) nodes. Thus the average time to run **fdleaf** is bounded by $O(K) \cdot O(n) = O(Kn)$. The total average time, and hence, the average space, are bounded by $O(K) + O(Kn) = O(Kn)$. Q.E.D.

6. IMPLEMENTATION

The algorithm was implemented in *C* on a VAX 11/780. Some synthetic three-dimensional objects were simulated. Various rotations and translations are per-

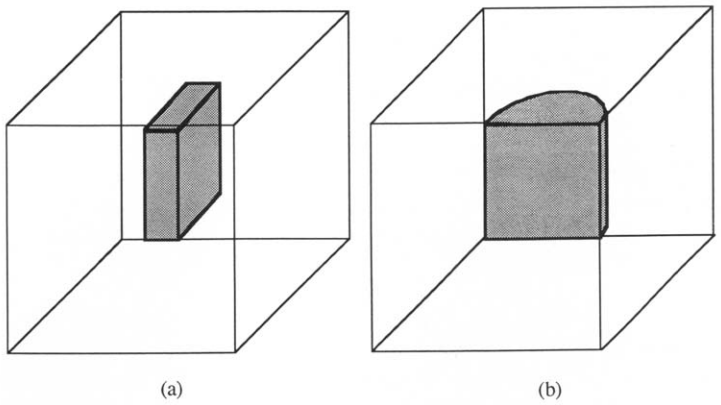

FIG. 5. Two synthetic objects for simulations.

formed. The results for two objects are presented here. Their shape, original orientations and positions are shown in Fig. 5. In Table 2a, b, T_1 is the source tree and T_2 is the target tree. Table 2a, b gives the number of black nodes, B, the number of internal nodes (also tree size since only internal nodes are created), I, maximum memory used for the construction of this target tree, M, in terms of nodes. From Table 2a, b it can be seen that the number of nodes is larger if the object contains curved surfaces. In Table 2a, b the number of black and internal nodes each increases almost by a factor of 4 as n increases by 1, where n corresponds to the size of the universe space. This is consistent with the results of Theorems 1 and 2. It can be seen from Table 2a, b that given the position and the orientation of the object to be represented by the target tree, the maximum memory (M) used to construct this target tree is always close to the completed target tree size (I), which means the intermediate evolving (incomplete) tree size is tightly bounded. Table 2a, b shows also the ratio I/V, where V is the volume of the object. These ratios are small. As pointed out earlier, an internal node may use more space than required by a voxel, the exact space requirement of the octree depends on the actual data structure used. For example, the linear octrees use a one dimensional array to store the nodes, this requires less storage space for the given tree [8].

7. SUMMARY

An algorithm for arbitrary rotation and translation of objects represented by an octree is presented and implemented. The algorithm generates very few superfluous nodes in the intermediate stages so the memory used is very close to the tree size. The one-displacement approach avoids the error accumulation with a compact source tree.

The analysis reveals the compactness properties of the octree representation (Theorems 1 and 2). A general result is derived that the number of nodes in octree representing a cube is at most proportional to the surface area of the cube plus the side-length of the universe space. The bound on the ratio of the number of nodes in an octree representing a cube of side-length 2^m, and the volume of the represented cube approaches zero very fast when m gets large (bounded by $O(1/2^m)$, if n is

TABLE 2a

Simulation Data for Object (a) in Fig. 5

n		4			5				6				7			
V		96			768				6144				49152			
	T_1	T_2			T_1	T_2			T_1	T_2			T_1	T_2		
n^0	—	0,1,0	0,1,0	1,1,1	—	0,1,0	0,1,0	1,1,1	—	0,1,0	0,1,0	1,1,1	—	0,1,0	0,1,0	1,1,1
ϕ	—	−90°	−30°	30°	—	−90°	−30°	30°	—	−90°	−30°	30°	—	−90°	−30°	30°
T	—	16,0,0	8,0,0	2,2,0	—	32,0,0	16,0,0	4,4,0	—	64,0,0	32,0,0	8,8,0	—	128,0,0	64,0,0	16,16,16
B	12	12	47	85	12	12	183	441	112	12	739	2042	12	12	211	8851
I	7	7	21	39	7	7	64	170	7	7	236	673	7	7	922	2713
M	7	11	25	42	7	2	69	174	7	13	242	676	7	14	929	2718
I/V	0.073	0.073	0.22	0.41	0.0091	0.0091	0.083	0.22	0.0011	0.0011	0.038	0.11	0.00014	0.00014	0.019	0.055

TABLE 2b

Simulation Data for Object (b) in Fig. 5

n		4			5				6				7			
V		150.8			1206				9651				77210			
	T_1	T_2			T_1	T_2			T_1	T_2			T_1	T_2		
n^0	—	0,1,0	0,1,0	1,1,1	—	0,1,0	0,1,0	1,1,1	—	0,1,0	0,1,0	1,1,1	—	0,1,0	0,1,0	1,1,1
ϕ	—	−90°	−30°	−30°	—	−90°	−30°	−30°	—	−90°	−30°	−30°	—	−90°	−30°	−30°
T	—	16,0,0	4,0,0	1,−1,0	—	32,0,0	8,0,0	2,−2,0	—	64,0,0	16,0,0	4,−4,0	—	128,0,0	32,0,0	8,−8,16
B	40	40	152	129	376	376	684	721	1288	1288	2928	3114	4904	4904	12622	13277
I	17	17	55	64	97	97	231	250	337	337	907	1007	1457	1457	3667	4034
M	17	21	58	67	97	101	235	252	337	343	913	1010	1457	1464	3673	4041
I/V	0.11	0.11	0.36	0.42	0.0080	0.080	0.19	0.21	0.035	0.035	0.094	0.10	0.019	0.019	0.047	0.052

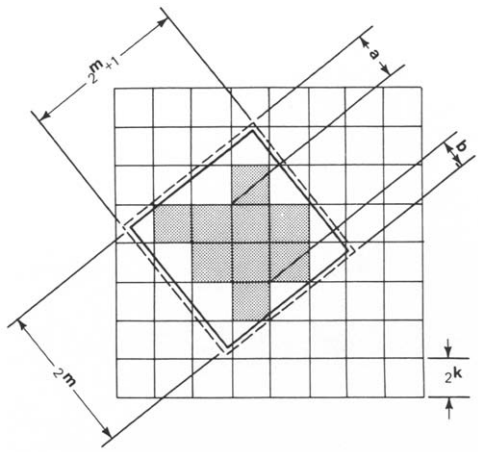

FIG. 6. Two-dimensional illustration for Theorem 1.

constant). The average space and time complexity is bounded by $O(Kn)$, where K is the number of nodes in source tree and the universe space has side-length 2^n.

APPENDIX A (LEMMA)

LEMMA. *Let T and I be the number of total nodes and the internal nodes, respectively, then $I = (T - 1)/8$.*

Proof. Let L be the number of the leaves. Consider each internal node corresponds to an eight-person game. Originally there are L players to take part in the games. Seven players lose after each game. Only the winner participates in the rest of the games. Finally only one player survives after I games. So we have $L - 7I = 1$ or $I = (L + I - 1)/8 = (T - 1)/8$. Q.E.D.

APPENDIX B (THEOREM 1)

THEOREM 1. *Let B_m be the number of black leaves in an octree representing a cube of side-length 2^m at arbitrary orientation and position, then*

$$B_m \leq 24.25 \cdot 4^m - 200 \cdot 2^m + 1454.$$

Proof. Let C be an m-cube and C' be the cube by growing C in all direction by 0.5 as shown in Fig. 6. Any unit cubes completely inside of C' must be black since their centers are on or inside C. a and b in Fig. 6 are the biggest depth unfilled by black blocks on two sides if only k-cubes or bigger ones are considered. a or b can not be greater than $\sqrt{3} \cdot 2^k$ otherwise another k-cube should have been black. So $a + b \leq 2\sqrt{3} \cdot 2^k$. Let v_k be the number of k-cubes in C' with the bigger cubes are considered as the corresponding number of k-cubes. Let w_k be the number of exact k-cubes in C'. Then, $w_k = v_k - 8v_{k+1}$. The volume filled by black k-cubes or bigger ones is no less than $[2^m + 1 - (a + b)]^3 \geq (2^m + 1 - 2\sqrt{3} \cdot 2^k)^3$. So $v_k \geq (2^m + 1 - 2\sqrt{3} \cdot 2^k)^3/(2^k)^3 = (2^{m-k} - 2\sqrt{3} + 2^{-k})^3$. $v_{m-2} \geq (2^2 - 2\sqrt{3} + 2^{-m+2})^3 > 0$, v_i is integer. So $v_{m-2} \geq 1$. Black unit cubes is no more than $(2^m + 1)^3 - 8(2^{m-1} - 2\sqrt{3} + 2^{-1})^3$, the whole volume subtracted by the low bound of v_1. Summing up all the low bounds of k-cubes of exact size, the low bound of B_m

follows for $m \geq 3$:

$$\begin{aligned}
B_m &\leq (2^m + 1)^3 - 8(2^{m-1} - 2\sqrt{3} + 2^{-1})^3 \\
&\quad + \sum_{i=4}^{m-1} \left[(2^i - 2\sqrt{3} + 2^{-(m-i)})^3 - 8(2^{i-1} - 2\sqrt{3} + 2^{-(m-i+1)})^3\right] \\
&\quad + (2^3 - 2\sqrt{3} - 2^{-(m-3)})^3 - 8 \cdot 1 + 1 \\
&= (2^m + 1)^3 - 7 \sum_{i=3}^{m-1} (2^i - 2\sqrt{3} + 2^{-(m-i+1)})^3 - 8 \cdot 1 + 1 \\
&\leq (3 + 14\sqrt{3})4^m - 249 \cdot 2^m + 2522 - 728\sqrt{3} \\
&\quad - 3\left(4^m - \frac{8^3}{2^m}\right) + 28\sqrt{3}\left(2^m - \frac{4^3}{2^m}\right) - 252\left(1 - \frac{2^3}{2^m}\right)
\end{aligned}$$

When $m \geq 3$, $1/2^m \leq 1/2^3$. We have

$$\begin{aligned}
B_m &\leq (3 + 14\sqrt{3})4^m - 249 \cdot 2^m + 2522 \\
&\quad - 728\sqrt{3} - 3 \cdot 4^m + 28\sqrt{3}\,2^m + 192 \\
&= \sqrt{3}\,4^m - (248 - 28\sqrt{3})2^m + 2714 - 728\sqrt{3} \\
&\leq 24.25 \cdot 4^m - 200 \cdot 2^m + 1454.
\end{aligned}$$

Since $B_m \leq (2^m + 1)^3$, the theorem also holds for $m \geq 0$. Q.E.D.

APPENDIX C (THEOREM 2)

THEOREM 2. *Let I_m be the number of internal nodes in an octree representing a cube of side-length 2^m at any orientation and position in $2^n \times 2^n \times 2^n$ space, then for $m \geq 3$*

$$I_m \leq 5.76 \cdot 4^m + 17 \cdot 2^m + 8n + 76m - 277$$
$$I_0 \leq 8n - 7, \qquad I_1 \leq 8n + 12, \qquad I_2 \leq 8n + 101.$$

Proof. Consider that an upright m-cube C is displaced to a new position. To construct the octree for it the subdivision will continue down to at least level m. A cube can overlap at most eight octants of higher level and overlap at most 27 octants of the same level. So at most

$$27 + \sum_{i=m+1}^{n-1} 8 + 1 = 8(n - m) + 20 \tag{3}$$

gray nodes of level m or higher would be generated. If subdivisions are proceeded further below level m, those cubes that have relation "partial" with C would probably become gray nodes and those have relation "inside" will become a black leaf node and be condensed to their parent later as shown in Fig. 7. Such k-cubes that have "partial" or "inside" relations with C are no more than

$$\frac{(2^m + \sqrt{3}(2^k - 1) + \sqrt{3}\,2^k)^3}{(2^k)^3} = (2^{m-k} + 2\sqrt{3} - \sqrt{3}\,2^{-k})^3 \tag{4}$$

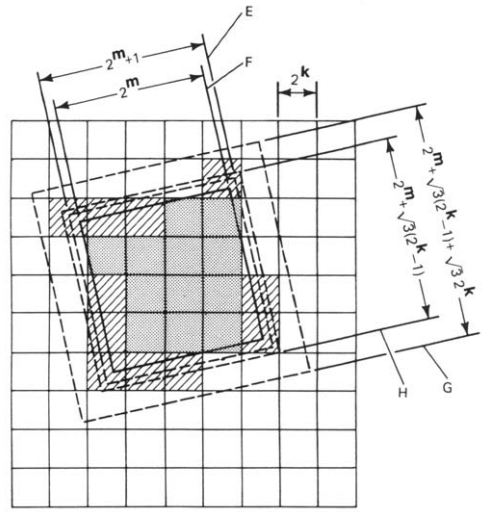

FIG. 7. Two-dimensional illustration for Theorem 2. F: Displaced cube to be represented. E: All the k-cubes completely inside E are black. H: All the k-cubes with centers on or inside H have "partial" relations. G: All the k-cubes which have "inside" or "partial" relations are included in G.

because all such nodes are included in a cube C' of side-length $2^m + \sqrt{3}(2^k - 1) + \sqrt{3}2^k$ as shown in Fig. 7. Among them at least

$$\frac{(2^m + 1 - 2\sqrt{3}2^k)^3}{(2^k)^3} = (2^{m-k} - 2\sqrt{3} + 2^{-k})^3 \tag{5}$$

k-cubes are black as discussed in the proof of Theorem 1. From (4) and (5), at most

$$(2^{m-k} + 2\sqrt{3} - \sqrt{3}2^{-k})^3 - (2^{m-k} - 2\sqrt{3} + 2^{-k})^3 \tag{6}$$

gray nodes are at level k. With (3) and (6) we have, for $m \geq 3$

$$\begin{aligned} I_m &\leq 8(n-m) + 20 \\ &\quad + \sum_{k=1}^{m-1} \left[(2^{m-k} + 2\sqrt{3} - \sqrt{3}2^{-k})^3 - (2^{m-k} - 2\sqrt{3} + 2^{-k})^3\right] \\ &= \frac{25\sqrt{3} - 3}{7}4^m + \left(16 + \frac{6}{7}\right)2^m + 8n + (48\sqrt{3} - 8)m \\ &\quad - \frac{449 + 983\sqrt{3}}{7} + \frac{752 + 528\sqrt{3}}{7}\frac{1}{2^m} \\ &\quad - \frac{48 + 224\sqrt{3}}{7}\frac{1}{4^m} + \frac{8 + 24\sqrt{3}}{7}\frac{1}{8^m} \\ &\leq 5.76 \cdot 4^m + 17 \cdot 2^m + 8n + 76m - 277. \end{aligned} \tag{7}$$

When $m < 3$ from the above arguments, we have

$$I_0 \leq 1 + \sum_{i=1}^{n-1} 8 = 8n - 7.$$

From (3), $I_1 \leq 8(n-1) + 20 = 8n + 12$. From (3) and (4),

$$I_2 \leq 8(n-2) + 20 + \left[(2^{2-1} + 2\sqrt{3} - \sqrt{3}\,2^{-1})^3\right] = 8n + 101. \quad \text{Q.E.D.}$$

ACKNOWLEDGMENT

This research was supported in part by the National Bureau of Standards under Grant COMM 60NANB4D0004 and the National Science Foundation under Grant ECS 83-52408.

REFERENCES

1. N. Ahuja, L. S. Davis, D. L. Milgram, and A. Rosenfeld, Piecewise approximation of pictures using maximal neighborhoods, *IEEE Trans. Comput.* **C-27**, 1978, 375–379.
2. N. Ahuja, R. T. Chien, R. Yen, and N. Bridwell, Interference detection and collision avoidance among three dimensional objects, in *Proceedings, 1st National Conf. on Artificial Intelligence, August 19–21, 1980*, pp. 44–48.
3. N. Ahuja and C. Nash, Octree representation of moving objects, *Comput. Vision. Graphics Image Process.*, **26**, 1984, 207–216.
4. D. Ballard and C. Brown, *Computer Vision*, Prentice–Hall, Englewood Cliffs, NJ, 1982.
5. C. H. Chien and J. K. Aggarwal, A volume/surface octree representation, in *Proceedings, 7th Int. Conf. on Pattern Recognition, 1984*, pp. 817–820.
6. C. R. Dyer, The space efficiency of quadtrees, *Comput. Graphics Image Process.*, **19**, 1982, 335–348.
7. C. M. Eastman, Representations for space planning, *Comm. ACM*, **13**, 1970, 242–270.
8. I. Gargantini, Linear octrees for fast processing of three-dimensional objects, *Compt. Graphics Image Process.*, **20**, 1982, 365–374.
9. G. M. Hunter and K. Steiglitz, Operations on images using quad trees, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-1**, No. 2, 1979, 145–153.
10. C. L. Jackins and S. L. Tanimoto, Oct-trees and their use in representing three-dimensional objects, *Comput. Graphics Image Process.*, **14**, 1980, 249–270.
11. K. Maruyama, *A Procedure for Detecting Intersections and Its Application*, University of Illinois, Computer Science Technical Report No. 449, May 1971.
12. W. N. Martin and J. K. Aggarwal, Volumetric descriptions of objects from multiple views. *IEEE Trans. Pattern Anal. Machine Intell.*, **PAMI-5**, No. 2, 1983, 150–158.
13. D. Meagher, Geometric modeling using octree encoding, *Comput. Graphics Image Process.*, **19**, 1982.
14. D. Meagher, Efficient synthetic image generation of arbitrary 3D objects, in *Proceedings, IEEE Computer Soc. Conf. on Pattern Recog. and Image Proc., 1982*, pp. 473–478.
15. W. Osse and N. Ahuja, Efficient octree representation of moving objects in *Proceedings, 7th Int. Conf. on Pattern Recognition, 1984*, pp. 821–823.
16. A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, Academic Press, New York, 1982.
17. H. Samet, Computing perimeters of regions in images represented by quadtrees, *IEEE Trans. Pattern Anal. Machine Intell.*, **PAMI-3**, No. 6, 1981, 683–687.
18. M. Shneier, Calculations of geometric properties using quadtrees, *Comput. Graphics Image Process.*, **16**, 1981, 296–302.
19. M. Shneier, E. Kent, and P. Mansbach, Representing workspace and model knowledge for a robot with mobile sensors, in *Proceedings, 7th Int. Conf. on Pattern Recognition, 1984*, pp. 199–202.
20. S. N. Srihari, Hierarchical representations for serial section images, in *Proceedings, 5th Int. Conf. on Pattern Recognition, 1980*, pp. 1075–1080.
21. J. Veenstra and N. Ahuja, Octree generation of an object from silhouette views, in *Proceedings, IEEE Int. Conf. on Robotics and Automation, 1985*, pp. 843–848.