COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING 49, 68–84 (1990)

NOTE

Octree Generation from Object Silhouettes in Perspective Views

SANJAY K. SRIVASTAVA AND NARENDRA AHUJA

Coordinated Science Laboratory, University of Illinois Urbana-Champaign, Urbana, Illinois 61801-3082

Received October 26, 1987; accepted March 29, 1989

Octrees are useful for object representation when fast access to coarse spatial occupancy information is necessary. This paper presents an efficient algorithm for generating octrees from multiple perspective views of an object. The algorithm first obtains a polygonal approximation of the object silhouette. This polygon is then decomposed into convex components. For each convex component, a pyramid is formed treating the view point as its apex and the convex components as a cross section. The octree representation of each of these pyramids is obtained by performing intersection detection of the object with the cubes corresponding to octree nodes. The intersection detection step is made efficient by decomposing it into a coarse-to-fine sequence of intersection tests. The octree for one silhouette is obtained by taking the union of octrees obtained for each component. An intersection of octrees corresponding to different viewing directions gives the final octree of the object. An implementation of the algorithm is given. The accuracy of the octree representation of the objects is evaluated. The ratio of the actual volume of the object to the volume of the object reconstructed from the octree representation is used as a performance index of the algorithm.

1. INTRODUCTION

In many robotics applications, such as motion trajectory planning, it suffices to have an approximate representation of the space occupied by objects in the workspace; a precise description of the objects is not necessary. This paper concerns generation of one such representation—the octree.

The octree is a tree of degree eight which represents the space occupied by objects contained in the work space defined by a "universe cube." If this cube is empty or completely filled by objects, then a single (root) node, marked white (empty) or black (full), respectively, constitutes the octree representation. Otherwise, the universe cube is decomposed as shown in Fig. 1.1 into eight octants, which are represented by eight children of the root node. The root node is marked grey to indicate the partial occupancy of the workspace. Each octant is tested to see whether it is completely occupied by objects, is partially occupied by objects, or is completely contained in free space. The octree node corresponding to the octant is given a color black, grey, or white, respectively. This decomposition is carried out recursively for grey nodes. The decomposition halts either when all leaf nodes are found to be black or white, or when a prespecified level of resolution is reached. An example of an object and its corresponding octree are shown in Fig. 1.2.

This paper concerns generation of the octree of an object from its images. Specifically, given the silhouettes of perspective views of an object taken from different directions, we wish to construct the octree representation of the object.

0734-189X/90 $3.00

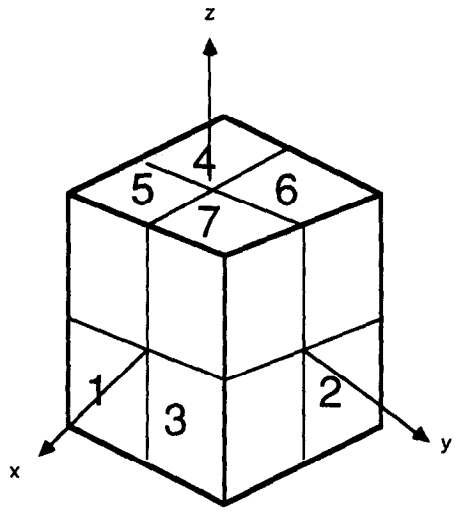

FIG. 1.1. A cube and its decomposition into octants.

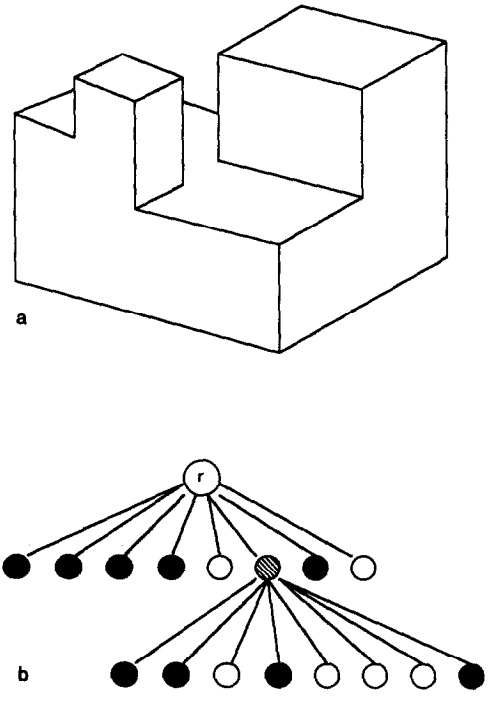

FIG. 1.2. An object (a) and its octree representation (b). r is the root node.

This problem has been addressed by several researchers. Veenstra and Ahuja present an algorithm for octree generation from a fixed number of orthographic views [1, 2]. In this approach, the images must be obtained from some subset of a set of 13 fixed viewing directions corresponding to the 3 "face" views, 6 "edge" views, and 4 "corner" views. These views are chosen because they provide a simple relationship between the pixels in the image and the octant labels in the octree, replacing the computation of detecting intersections between the octree space and the objects by a table lookup operation. Chien and Aggarwal describe another algorithm which constructs an octree from three orthographic projections [4]. The intersections here are also inferred using a table lookup operation; however, their algorithm accepts only up to three (orthogonal) viewing directions. In another approach, Shneier *et al.* construct the octree by projecting each cube onto the image plane and explicitly performing an intersection detection test for each node [3, 8]. Potmesil also uses a similar approach [7]. Both algorithms are similar to ours though the intersection detection is performed in two-dimensions in their case.

We will present an algorithm that accepts images taken from arbitrary viewpoints. While Shneier *et al.* estimate the octree nodes by projecting corresponding cubes in the image and performing intersection test with the silhouette, we perform the intersection test in the three-dimensional space. Given a silhouette, we obtain a representation of the pyramid having apex at the viewpoint and the silhouette as a cross section (Fig. 1.3). An octree is generated for the space occupied by the pyramid, by recursively testing for the intersection of octree octants with the pyramid. The intersection test exploits the simplicity of the three-dimensional structure of cubes. Planar projections of the cubes are in general irregular hexagons, and this makes the intersection detection in two dimensions complex. The algorithms of Shneier *et al.* and Potmesil as well as the one presented here offer the choice, when necessary, of obtaining a higher resolution octree than is possible with silhouettes acquired from a limited set of directions [1, 4]. The higher resolution comes at a higher computational cost [1]. The final octree is generated by performing an intersection of the octrees obtained from the individual silhouettes. The next section presents our octree generation algorithm (Section 2). Section 3 discusses the

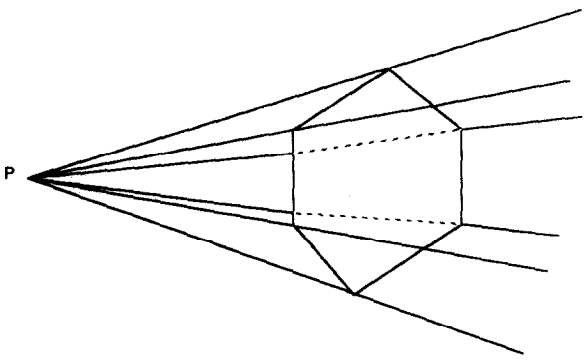

FIG. 1.3. A pyramid formed by treating the view point P as its apex and the polygonal silhouette as one of its cross sections. The pyramid is used as the best estimate of the object from the given view point, for the purpose of constructing the octree representation of the object.

computational complexity of the algorithm and the results of the algorithm for a collection of objects are given in Section 4. Section 5 summarizes the paper.

2. AN ALGORITHM FOR GENERATING OCTREE FROM SILHOUETTES

Our algorithm starts with the object silhouette and constructs a pyramid representation (Fig. 1.3) of the three-dimensional space occupied by the object. The pyramid is treated as the best estimate of the object from the view point under consideration. The nodes in the octree representation of the pyramid are then generated and colored black, grey, or white, recursively, starting with the root node. These two operations comprise the two major steps of the algorithm.

In the first step, we threshold the image to get a binary silhouette. Then, the boundary of the silhouette is approximated by a polygon. We then decompose the polygonal silhouette into convex components. For each convex component, the semi-infinite pyramid illustrated in Fig. 1.3 is constructed and then passed on to the second step.

The second step checks for intersection between a semi-infinite pyramid and an upright cube corresponding to an octree node. We decompose this intersection detection step into a series of tests, each examining a distinct geometric configuration of the cube and the pyramid. These tests exploit the known cubical and pyramidal nature of the shapes to make intersection detection efficient, rather than perform a general three-dimensional polyhedral intersection detection. The tests performed earlier in the sequence are computationally less complex than those that follow; the resulting coarse-to-fine strategy makes intersection detection computationally efficient. The first test approximates the pyramid by the smallest enclosing cone and the cube by the smallest enclosing sphere. It is then easily determined if the sphere and the cone do not intersect, in which case the octree node is white. If they intersect, then the following second test eliminates a few cases of black and grey nodes. The eight vertices of the cube are tested to see whether they lie inside or outside the pyramid. If all the vertices are found to be contained inside the pyramid, it is decided that the node is black; if some vertices are found to be inside and some outside, the node is colored grey. If all vertices are found to be outside, then the cube may or may not intersect with the pyramid. To resolve this problem, a third test examines the location of the cube with respect to each of the faces of the pyramid. If there is any face such that all eight vertices of the cube lie on the outside of the face, then the cube is outside the pyramid. Figure 2.1 illustrates a two-dimensional analog of this test, where a square is tested for being completely on the inside or outside of an edge of polygon. Any intersections of the edges of the pyramid with the extended faces of the cube are found. If any of the points of intersection are contained inside the cube, then the node is colored grey; otherwise the node is colored white.

These steps together determine an octree for each convex component of a given polygonal silhouette. The final octree for the pyramid obtained from a single silhouette is the union of the octrees obtained for all convex components of the silhouette. Such octrees obtained from the different silhouettes are intersected to obtain the final octree of the object which represents the volume of intersection of the different pyramids. We will now describe these steps in detail.

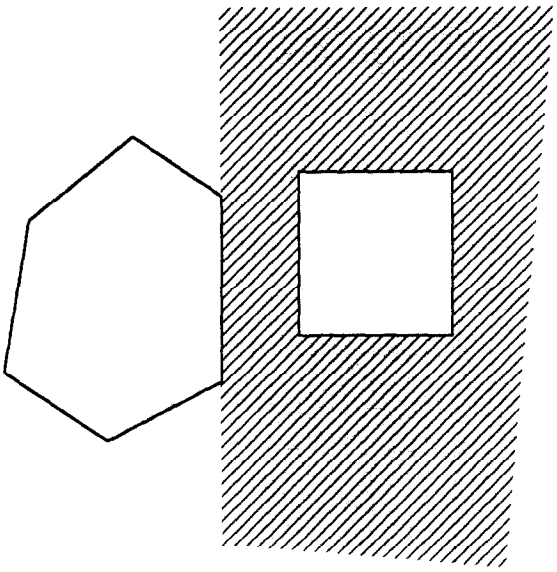

FIG. 2.1. Geometry illustrating two-dimensional analog of a white node detected in Test 3. The square will be detected to be outside the polygon if it lies in the shaded region.

2.1. *Processing the Silhouette*

The first step is to threshold the picture to get a binary image showing the object silhouette. Then the boundary of the silhouette is found. The boundary so obtained is approximated by a polygon using an algorithm given by Pavlidis [5].[1] The silhouette polygon may, in general, be non-convex. If so, it is decomposed into convex polygons using another algorithm described by Pavlidis [5].[2]

Each of the convex polygons obtained above is treated as a separate object, and its octree is constructed as per the following algorithm. The final octree for the given silhouettes is the union of the octrees so obtained. The union can be derived in a straightforward manner from the individual octrees.

A pyramid is constructed making the view point as the apex and the convex polygon as a cross section, as shown in Fig. 1.3. The pyramid is represented as a

[1] The polygon boundary is an ordered set of boundary pixels, denoted $d_0, d_1, \ldots, d_n$. The vertices of the polygon are chosen from d_i's in the following manner. The algorithm starts from an arbitrary point on the boundary (say d_0) and traverses successive boundary points d_i to choose vertices. Let l be the line joining d_0 and d_i. Then, a point j, $0 < j < i$, is chosen as a vertex if distance $(l, d_j) >$ threshold, and $d(l, d_j) \geq d(l, d_k)$ for all k, $0 < k < i$.

[2] The algorithm starts by locating the non-convex corners of the polygon. It is tested whether the polygon is a spiral, i.e., whether its non-convex corners are consecutive. If this is the case, then the bisectrices of the non-convex corners are used for decomposing the polygon. This step is illustrated in Fig. 2.2. If there are non-adjacent non-convex corners, lines joining two such corners are used for decomposing the polygon. Two non-convex vertices are candidates for being joined if the line connecting them lies completely inside the polygon. This is illustrated in Fig. 2.3. From such candidate pairs of vertices, the ones with minimum distance between them are chosen for decomposition.

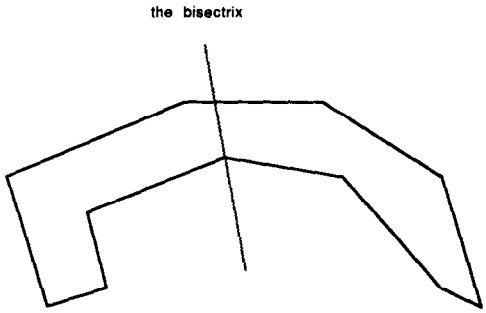

FIG. 2.2. A spiral and a bisectrix used for decomposing the polygon into convex components.

circular list of the bounding constraints of the form

$$g_i(x, y, z) = a_i x + b_i y + c_i z + d_i, \tag{2.1.1}$$

where $a_i x + b_i y + c_i z + d_i = 0$ is the equation of the ith plane passing through the view point and an edge of the polygon. Whether a given point (x_0, y_0, z_0) lies inside, on or outside of the plane represented by $a_i x + b_i y + c_i z + d_i = 0$ can be determined by checking if the value of $g_i(x_0, y_0, z_0)$ is less than, equal to, or greater than zero.

Next, the space occupied by the pyramid is concisely represented by Comba's function [6],

$$f(x, y, z) = \sum_i g_i(x, y, z) + |g_i(x, y, z)|. \tag{2.1.2}$$

We also partition the constraints (i.e., the equations $g_i(x, y, z)$) into eight subsets such that for each subset the coefficients a_i, b_i, and c_i in $(a_i x + b_i y + c_i z + d_i)$ are of a particular sign. That is,

$$G_1 = \{ g_i | a_i, b_i, c_i > 0 \}, \tag{2.1.3a}$$

$$G_2 = \{ g_i | a_i, b_i > 0, c_i < 0 \}, \tag{2.1.3b}$$

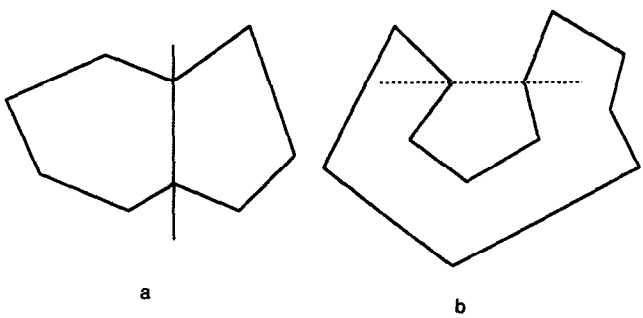

FIG. 2.3. (a) A candidate pair of vertices for decomposing the polygon into convex components. (b) A pair that cannot be used.

and so on for G_j, $1 \leq j \leq 8$. This partitioning of the plane equations is used to improve the efficiency of Test 3 described in Section 2.2.

All the steps discussed so far are performed before the actual construction of the octree begins. They are performed once for each silhouette image and the results are used by the intersection detection test which is discussed next.

2.2. *Intersection Detection Test*

The object represented by the silhouette is contained in the semi-infinite pyramid whose representation is derived in the previous section. To obtain the octree representation of the pyramid, the various nodes of the octree are examined recursively, starting with the root node. The cube corresponding to each node is tested for intersection with the pyramid and assigned a color based on the result of the test. This test determines to a large extent the computational complexity of the entire octree generation. Following is a coarse-to-fine algorithm for detecting intersection between a given cube and pyramid. The algorithm performs a series of tests. Successive tests increase in complexity, but the frequency with which they are used decreases.

Test 1. To determine whether the cube corresponding to a given node is completely outside the given semi-infinite pyramid, we fit a tightest fitting sphere around the cube and a tightest fitting right circular cone around the pyramid. The center of the sphere coincides with the center of the cube, and the diameter of the sphere is the same as a major diagonal of the cube. For finding the cone, we fit a circle around the polygon. The center of this circle is the center of gravity of the polygon and its radius is the maximum distance from the center of gravity to the vertices of the polygon. A cone is formed with the view point as its apex, and the line joining the view point and the center of the circle as its axis. The surface of the cone is constrained to be tangential to the sphere with the same center and diameter as the circle around the polygon. A plane containing the apex of the cone and the center of the sphere is illustrated in Fig. 2.4.

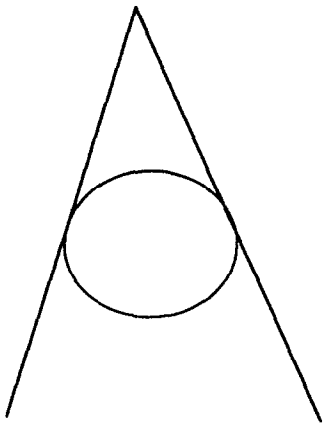

FIG. 2.4. A plane containing the axis of a cone when the surface of the cone is tangential to a sphere. The cross section of the sphere cut by the plane is also shown.

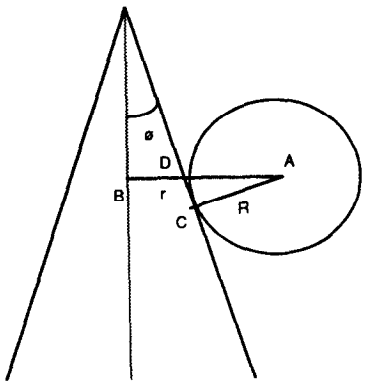

FIG. 2.5. A plane passing through the center of a sphere and the axis of a cone when the sphere is touching the cone from outside. AB is the perpendicular dropped from the center of the sphere on the axis of the cone. AC is the radius of the sphere at the point of contact of the sphere and the cone. ϕ is the half angle of the cone.

Now, we will devise a test to decide whether the sphere and the cone intersect. Let us assume that the sphere is just touching the cone. It is easy to see, using symmetry arguments, that the center of the sphere, the point of contact of the sphere and the cone, and the axis of the cone all lie in the same plane. This plane is illustrated in Fig. 2.5. The cross section of the sphere and the cone cut by the plane are also shown. Let A be the center of the sphere and AB be the perpendicular drawn on the axis of the cone. Let C be the point of contact. Clearly, if *angle* $CAB = \phi$ then $AB = BD + AD = r + R \sec \phi$, where r is the radius of the circular cross section of the cone along the perpendicular mentioned above, R is radius of the sphere, and ϕ is the half angle of the cone. The sphere is outside the cone if and only if the length of the perpendicular dropped from the center of the sphere on the axis of the cone is greater than $r + R \sec \phi$. In such a case, the node is colored white since the cube is totally outside the pyramid. If a decision cannot be made, the algorithm moves on to the next step.

Test 2. Next, it is determined whether the cube is completely inside the pyramid. This case is shown in Fig. 2.6. It follows from the definition of g_i's (2.1.1) that a point (x_0, y_0, z_0) is inside the pyramid if and only if $g_i(x_0, y_0, z_0) < 0$ for all i. We

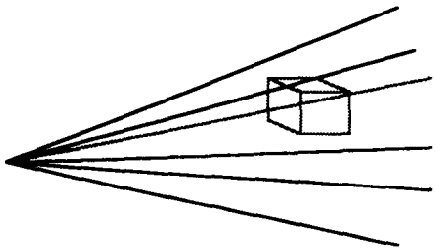

FIG. 2.6. Illustration of a black node. The cube corresponding to the octree node is contained in the pyramid representing the silhouette.

use Comba's function (2.1.2) to devise a simple method to carry out this test. It is obvious that if all constraints $g_i(x_0, y_0, z_0) < 0$ are satisfied then $f(x_0, y_0, z_0) = 0$. Even if one of the constraints is violated then $f > 0$. Hence, computing Comba's function f gives us a simple test for deciding whether a given point is inside the pyramid or not.

To test whether a cube is inside the pyramid, the algorithm performs the test mentioned above on all its vertices. Since the pyramid is a convex polyhedron (as the pyramid is constructed starting with a convex component of the silhouette as its cross section), it will completely contain the cube if all the vertices of the cube are contained inside it. Similarly, if some of the vertices are inside and some outside, we know that the cube partially intersects the pyramid. In this case the node is colored grey. If a decision cannot be made at this stage, the algorithm proceeds to the next step.

Test 3. This step determines whether the cube is completely outside the pyramid under consideration; it does not check for all cases of empty intersection but eliminates a significant number of these cases.

This test determines if the cube is completely contained in one of the half spaces defining the region outside the pyramid. A two-dimensional analog of this case, as discussed in Section 2.1, is shown in Fig. 2.1. Let $g(x, y, z) < 0$ (2.1.1) define a bounding plane of the pyramid, and let (x_i, y_i, z_i), $1 \leq i \leq 8$, be the vertices of the cube. If for any bounding plane g, $g(x_i, y_i, z_i) \geq 0$ for all i, $1 \leq i \leq 8$, then it is decided that the cube is outside the pyramid and the corresponding node is colored white.

If a decision could not be made at this stage, the algorithm moves on to the next test.

Test 4. We are left with the case in which all the vertices are outside the pyramid but no half space defining the region outside of the pyramid contains the cube. A two-dimensional analog of this is shown in Fig. 2.7

To discriminate between the two cases, we perform a simple test. We find the intersection of the edges of the pyramid with the extended faces of the cube. For example, let the cube be defined by $a_1 < x < a_2$, $b_1 < y < b_2$, and $c_1 < z < c_2$. Clearly, one of the faces extended in all directions is defined by $x = a_1$. We find the point of intersection of an edge with the infinite plane $x = a_1$. If this point satisfies

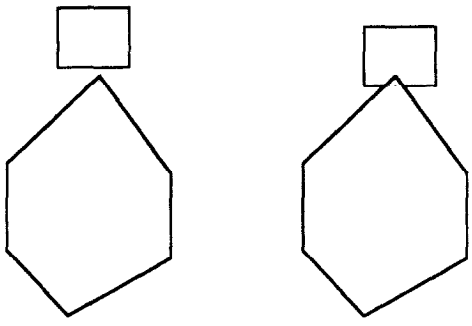

FIG. 2.7. Configurations of a square such that all the vertices of the square are outside the polygon but no half plane defining the region outside the polygon contains it.

the constraints of the cube mentioned above, then we know that the pyramid and the cube intersect. In this case, the node is colored grey. This test is carried out for all the edges with respect to all the faces of the cube. If none of the points of intersection lie inside the cube then the node is colored white, otherwise the node is colored grey.

Algorithm

The above tests are straightforward to implement in an algorithm. However, a direct implementation of Test 3 may be inefficient (Section 3). We use the following formulation of Test 3 in our algorithm in the interest of computational efficiency.

Let one of the bounding constraints be $ax + by + cz + d < 0$, and let the cube be defined by $a_1 \le x \le a_2$, $b_1 \le y \le b_2$, and $c_1 \le z \le c_2$. Suppose we want to know whether the given cube lies completely in the region defined by $ax + by + cz + d \ge 0$. In other words, we want to determine whether the function $g(x, y, z) = ax + by + cz + d$ takes values which are only positive when the coordinates x, y, and z are constrained by $a_1 \le x \le a_2$, $b_1 \le y \le b_2$, and $c_1 \le z \le c_2$. This reduces to the problem of finding the minimum of the function $g(x, y, z)$ under the constraints $a_1 \le x \le a_2$, $b_1 \le y \le b_2$, and $c_1 \le z \le c_2$ and checking whether it is greater than zero. Let the coefficients a, b, and c in $g(x, y, z) = ax + by + cz + d$ be such that $a, b > 0$, and $c < 0$. Clearly, g is minimized by (a_1, b_1, c_2). Hence we can just evaluate g at (a_1, b_1, c_2) and check whether it is greater than zero instead of evaluating g at all the eight corners of the cube. Also, corresponding Comba's function $g + |g|$, evaluated at (a_1, b_1, c_2), will be positive if and only if the cube is on the outside of this plane. Now, consider

$$G = \{ g | a, b > 0, c < 0 \}, \tag{2.2.1a}$$

and

$$f = \sum_{g \in G} g + |g|. \tag{2.2.1b}$$

If f, evaluated at (a_1, b_1, c_2) is greater than zero then it implies that at least one of $g \in G$ is such that corresponding Comba's function is greater than zero. It can then be concluded that the cube is on the outside of at least one of the planes. This observation suggests the following test.

The set of planes is partitioned into eight subsets G_i's (2.1.3) such that each partition has planes with the corresponding coefficients a, b, and c of the same sign. Then, for each subset G_i, Comba's function f_i (2.2.1) is formed. For a given cube, all f_i's are evaluated at one of the corners of the cube, determined by the sign of the coefficients in the elements of the corresponding set G_i. For example, for the cube mentioned above, f_1 is evaluated at (a_1, b_1, c_1), f_2 is evaluated at (a_1, b_1, c_2), and so on. If any of these values turn out to be positive, it implies that the cube is outside the pyramid and is then colored white.

3. COMPLEXITY ANALYSIS

This section discusses the computational complexity of the algorithm presented in Section 2. Complexity of the various steps will be analyzed individually and then their effects on the overall complexity of the algorithm will be discussed.

As discussed in the previous section, the silhouette processing step of the algorithm (Section 2.1) is performed once for each image. The next major step, i.e., the intersection detection step (Section 2.2), is performed once for each node of the octree. Since the number of nodes will be usually very large (of the order of 8^d, where d is the depth of the octree), it is the second step which largely determines the complexity of the algorithm. The complexity of the various tests involved in intersection detection for a specific node is discussed next.

In Test 1 (Section 2.2), to check whether a cone enclosing the semi-infinite pyramid intersects with the sphere around the cube takes constant time. In Test 2 (Section 2.2), Comba's function is evaluated for eight different vertices of the cube under consideration. Hence, once again a constant time is required to perform the computation. In Test 3, we check for empty intersection. As was pointed out, the obvious way of going about it is to check whether $g(x, y, z) \geq 0$, for all the eight vertices of the cube. This test has to be carried out once for each plane. This requires $O(m)$ computations, where m = number of planes bounding the pyramid. But in the efficient implementation of Test 3 in our algorithm (Section 2.2), Test 3 takes constant time. This is because, depending on the sign of coefficients in the equation, only one test is performed. Test 4 requires $O(m)$ computations since an intersection test has to be performed for each of the edges, which are m in number. The overall complexity of the octree generation algorithm then is $O(n - n_1) + O(mn_1)$, where n_1 = number of nodes for which a decision was taken in the last step of the algorithm and n = number of nodes in the octree. Test 4 is performed only when a decision cannot be made in the earlier tests.

The complexity discussed so far is the complexity of constructing the octree for a convex polygonal component of a single silhouette. To obtain the octree corresponding to a given image by taking the union of octrees of the convex components requires $O(\sum_i n_i)$ time, where n_i is the number of nodes in the octree corresponding to the ith component. This is because the union of octrees can be performed in linear time with respect to the sum of the numbers of nodes in the octrees. Similarly, determining the final octree by intersecting the octrees obtained from different silhouettes can be performed in time which is linear in the sum of the numbers of nodes in individual octrees.

4. PERFORMANCE ANALYSIS AND EXPERIMENTAL RESULTS

In this section, we will discuss some issues regarding the performance of our octree generation algorithm. A criterion for evaluating the performance will be discussed. This will be followed by results of our algorithm on some examples.

Our algorithm is expected to derive at best a representation for a bounding volume of the object, since it only works with silhouettes. The algorithm suffers from some limitations which are inherent to the problem. The information loss which occurs during formation of the image is one of them. The 2D quantization error of the image is another. Further, if there are surface concavities in the object they are lost in the image and hence will not be represented in the octree generated by our algorithm. If there are any holes in the object, they can at best be represented only partially. Consider a cube with a hole which goes through it from one face to the face opposite it. This hole will be present, at best, in some of the images wherein one of the faces with the hole is facing the camera. Consequently, in

the cube reconstructed from the octree generated by our algorithm the hole will appear smaller in volume. Also, it is possible that the hole does not appear in any of the images, in which case it will not at all be represented in the octree of the object. Other errors result from the inaccuracies in the octree representation itself, i.e., the 3D quantization that takes place due to limiting the depth of the octree. Beyond these general sources of inaccuracies, our algorithm has shortcomings of its own. A major problem is the information loss which occurs when the silhouette is approximated by a polygon. A tight polygonal fit to the silhouette will reduce this problem but will increase the number of convex components and the number of edges in each polygonal component. As discussed in Section 3 these factors will result in an increase in the time complexity of the algorithm. Hence some compromise between accuracy of the representation and time complexity has to be reached.

To evaluate the performance of the algorithm, we need to evaluate the quality of the generated octrees. Clearly, the octree representation is not well suited for applications where shape details of the object represented are important; surface-based and other representations are more appropriate for such applications. However, in the cases where a coarse occupancy map of the space is desired, the octree representation may suffice, since it would capture most of the object volume, although not surface smoothness. Thus, a measure of error between the reconstructed and true object volumes is appropriate to test the performance of the octree generation algorithm. As a performance measure, we have chosen the ratio of the actual volume of the object to the reconstructed volume of the object.

We conducted simulation experiments with a set of simple geometric shapes. These include: cube, pyramid, sphere, octagonal prism, cone, cylinder, wedge, empty cube (i.e., a cube with a hole), and diamond. To compute the performance for each object we generated perspective views of the object located at a fixed distance, from different viewing directions. Object silhouette was computed in each view. Subsets of silhouettes were then used to construct the octree as described in Section 2. The volume of the object represented by the constructed octree was computed and was used to obtain the ratio of the known object volume to the reconstructed volume. This ratio is recorded in Figs. 4.1–4.6 as the performance measure of the octree generation algorithm. The performance index will, of course, depend on the orientation of the object. However, for cases in which a large number of viewing directions

Performance Results			
View-distance = 10.0			
Number of views = 6			
Object	Reconstructed Volume	Actual Volume	Ratio
cube	9.9086	8.000	0.8080
pyramid	3.2212	2.6667	0.8279
sphere	4.2333	4.1888	0.9895
octagonal-prism	7.7541	6.6274	0.8547
cone	2.7312	2.0944	0.7668
cylinder	6.3024	6.2832	0.9969
wedge	2.1323	1.6000	0.7503
empty_cube	6.6398	6.0000	0.9036
diamond	3.2092	2.6670	0.8310

FIGURE 4.1.

Performance Results			
View-distance = 10.0			
Number of views = 18			
Object	Reconstructed Volume	Actual Volume	Ratio
cube	9.9645	8.0000	0.8110
pyramid	3.2161	2.6667	0.8291
sphere	3.9669	4.1888	1.0559
octagonal-prism	7.7530	6.6274	0.8548
cone	2.4636	2.0944	0.8501
cylinder	5.8951	6.2832	1.0658
wedge	2.1191	1.6000	0.7550
empty_cube	6.6271	6.0000	0.9053
diamond	3.1015	2.6670	0.8599

FIGURE 4.2.

is used, e.g., Figs. 4.2, 4.3, 4.5, 4.6, the viewpoint dependence may be low and the observed performance index may be reasonably reliable.

The viewing directions were restricted to 13 and the viewing distances were limited to two different values. The viewing directions used correspond to head-on views of the: 3 pairs of parallel faces, 6 pairs of parallel edges, and 4 major diagonals, of the universe cube. Each pair may lead to two different silhouettes obtained from two antiparallel directions. The sets of silhouettes used in the algorithm were extracted from 6 face views alone: 6 face views and 12 edge views (a total of 18 views); and 6 face views, 12 edge views, and 8 corner views (a total of 26 views). Figs. 4.1–4.6 give the volume ratios computed experimentally for different cases. Reconstructed objects are displayed in Fig. 4.7.

Clearly, the perfect performance corresponds to a ratio of 1. However, this level of performance will not be achieved on an average with a finite number of views and because of other limitations discussed earlier in this section. In general, the volume of intersection of silhouette pyramids (Fig. 1.3) will be an overestimate of the true volume of the object; this will lead to a ratio of less than 1. Quantization effects due to image resolution, polygonal approximation of the silhouettes, and octree depth limits will cause further deviation in the ratio. The actual performance ratio may

Performance Results			
View-distance = 10.0			
Number of views = 26			
Object	Reconstructed Volume	Actual Volume	Ratio
cube	9.8645	8.0000	0.8110
pyramid	3.21600	2.6667	0.8292
sphere	3.9085	4.1888	1.0717
octagonal-prism	7.7445	6.6274	0.8557
cone	2.4217	2.0944	0.8648
cylinder	5.6807	6.2832	1.1060
wedge	2.1191	1.6000	0.7550
empty_cube	6.6170	6.0000	0.9067
diamond	3.0892	2.6670	0.8633

FIGURE 4.3.

Performance Results			
View-distance = 20.0			
Number of views = 6			
Object	Reconstructed Volume	Actual Volume	Ratio
cube	7.3984	8.0000	1.0813
pyramid	2.4496	2.6667	1.0886
sphere	4.1102	4.1888	1.0191
octagonal-prism	6.2821	6.6274	1.0550
cone	2.6741	2.0944	0.7832
cylinder	5.9776	6.2832	1.0511
wedge	2.1081	1.600	0.7589
empty_cube	6.5763	6.0000	0.9124
diamond	3.4612	2.6670	0.7705

FIGURE 4.4.

Performance Results			
View-distance = 20.0			
Number of views = 18			
Object	Reconstructed Volume	Actual Volume	Ratio
cube	7.3984	8.0000	1.0813
pyramid	2.4458	2.6667	1.0902
sphere	3.8109	4.1888	1.0992
octagonal-prism	6.0283	6.6274	1.0994
cone	2.4217	2.0944	0.8648
cylinder	5.7771	6.2832	1.0876
wedge	2.0976	1.6000	0.7627
empty_cube	6.5764	6.0000	0.9124
diamond	3.4021	2.6670	0.7839

FIGURE 4.5.

Performance Results			
View-distance = 20.0			
Number of views = 26			
Object	Reconstructed Volume	Actual Volume	Ratio
cube	7.3375	8.0000	1.0903
pyramid	2.3926	2.6667	1.1146
sphere	3.8109	4.1888	1.0992
octagonal-prism	5.8913	6.6274	1.1249
cone	2.3976	2.0944	0.8648
cylinder	5.7365	6.2832	1.0952
wedge	2.0976	1.6000	0.7627
empty_cube	6.5154	6.0000	0.9209
diamond	3.3984	2.6670	0.7848

FIGURE 4.6.

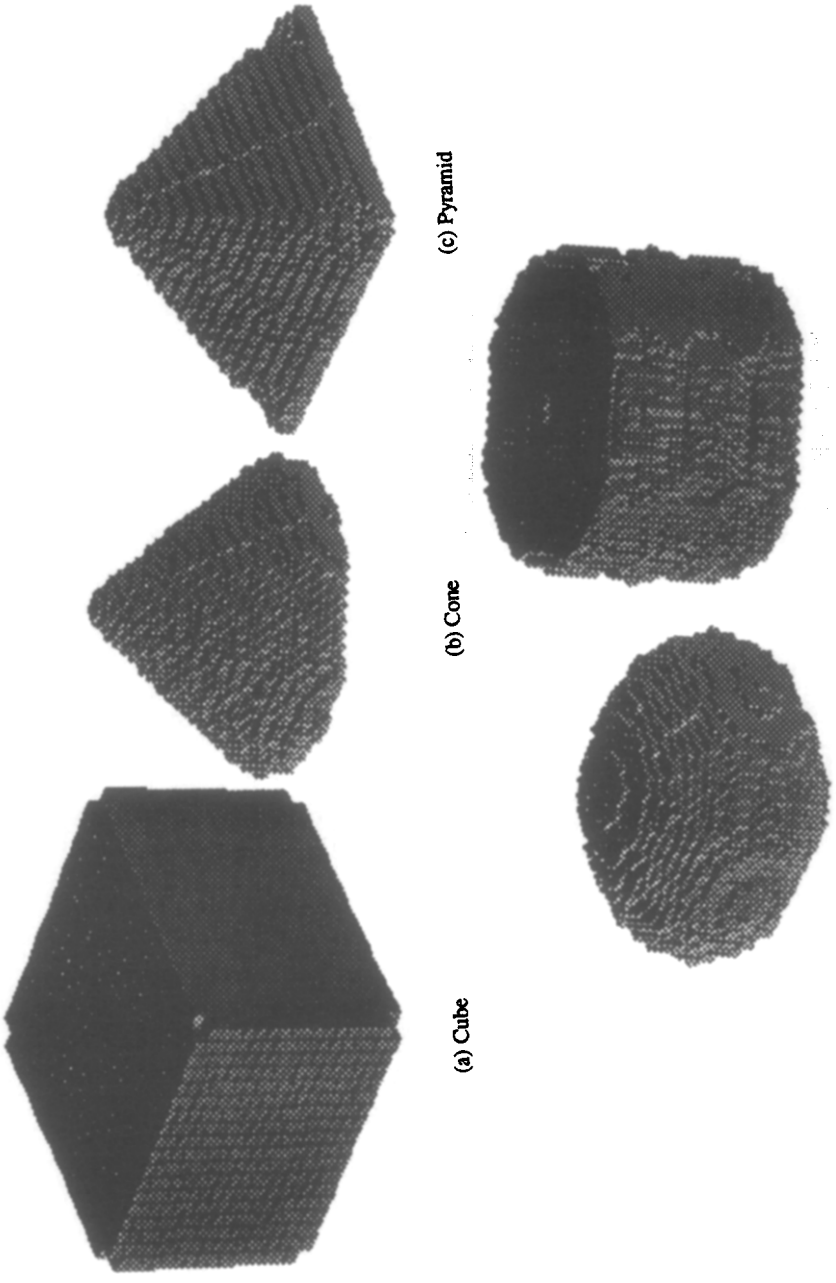

FIG. 4.7. Examples of objects reconstructed from their octree representation using 26 views.

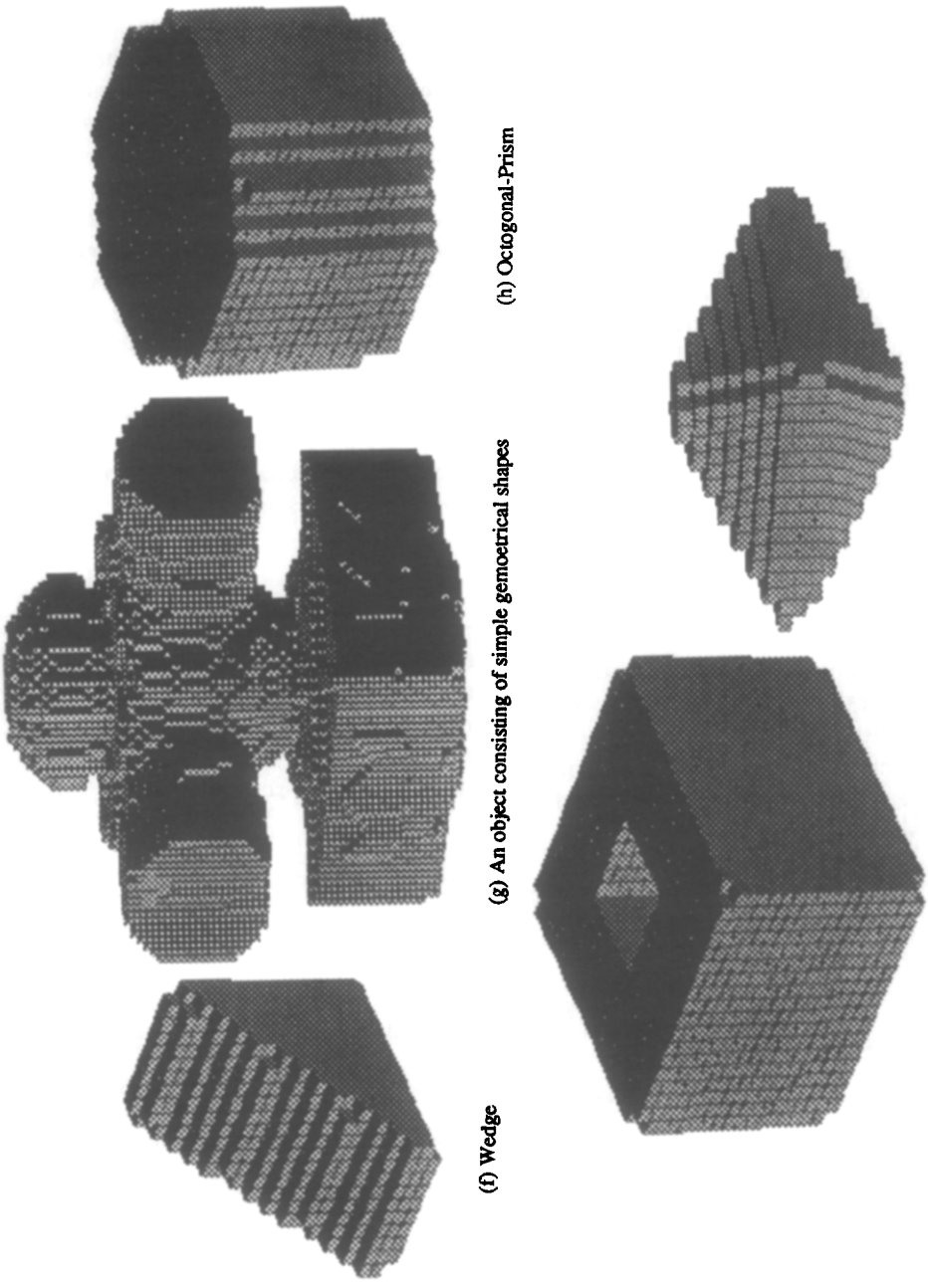

FIG. 4.7—*Continued*

even exceed 1. The former source of error will be more pronounced for less compact objects, since for such objects the convex hull reconstructed by the silhouette pyramids overestimates the actual volumes by a greater amount. This is evident from the experimental results in Figs. 4.1–4.6, where the performance ratios for cube, wedge, pyramid, diamond, etc. are among the worst, whereas the ratios for sphere and cylinder are among the best.

As the viewing distance to the object increases, the perspective projection approaches orthographic projection. Thus, the volume of intersection of the silhouette pyramids decreases with increasing viewing distance and becomes an increasingly accurate estimate of the true object volume. This leads to a general increase in the performance ratio observed going from Figs. 4.1–4.3 to Figs. 4.4–4.6, although there are small exceptions to this increase (e.g., diamond, cylinder), presumably because of the increased effect of image quantization as the object size in the image decreases with increasing viewing distance.

As expected, the volume of the reconstructed object becomes an increasingly better estimate of the actual volume as the number of views used increases (Figs. 4.1–4.3, 4.4–4.6).

5. SUMMARY

We have presented an algorithm to generate the octree representation of an object from its silhouettes. The silhouettes are extracted from perspective projections of the object from different view points. We have presented simulation results of running the algorithm on views of a variety of simple objects. The performance index of the algorithm was chosen to be the ratio of the actual volume of the object to the volume represented by its computed octree. For most compact objects used, this ratio was found to be above 0.85.

ACKNOWLEDGMENTS

The support of the National Science Foundation under Grant ECS 83-52408 and AT&T Information Systems is gratefully acknowledged.

REFERENCES

1. N. Ahuja and J. Veenstra, Generating octrees from object silhouettes, *IEEE Trans. Pattern Anal. Mach. Intell.*, **11**, 1989, 137–149.
2. J. Veenstra and N. Ahuja, Line drawings of octree-represented objects, *ACM Trans. Graphics* **7**, 1988, 61–75.
3. T. H. Hong and M. Shneier, Describing a robot's workspace using a sequence of views from a moving camera, *IEEE Trans. Pattern Anal. Mach. Intell.* **7**, 1985, 721–726.
4. C. H. Chien and J. K. Aggarwal, Volume/surface octrees for the representation of 3D objects, *Comput. Vision Graphics Image Process.* **36**, 1986, 100–113.
5. T. Pavlidis, *Structural Pattern Recognition*, Springer-Verlag, New York/Berlin, 1977.
6. P. G. Comba, A procedure for detecting intersections of three-dimensional objects, *J. Assoc. Comput. Mach.* **15**, No. 3, 1968, 354–366.
7. M. Potmesil, Generating octree models of 3D objects from their silhouettes in a sequence of images, *Comput. Vision Graphics Image Process.* **40**, 1987, 1–29.
8. M. Shneier, E. Kent, and P. Mansbach, Representing workspace and model knowledge for a robot with mobile sensors, in *Proc. Seventh Int. Conf. on Pattern Recognition, Montreal, Canada, July 1984*, pp. 199–202.