Generating Octrees from Object Silhouettes in Orthographic Views

NARENDRA AHUJA, SENIOR MEMBER, IEEE, AND JACK VEENSTRA

Abstract—**Octrees are used in many 3-D representation problems because they provide a compact data structure, allow rapid access to occupancy information, and implement many geometric manipulation algorithms efficiently. The initial acquisition of the 3-D information, however, is a common problem. This paper describes an algorithm to construct the octree representation of a 3-D object from silhouette images of the object. The images must be obtained from any subset of thirteen viewing directions corresponding to the three "face" views, six "edge" views, and four "corner" views of an upright cube. These views were chosen because they provide a simple relationship between pixels in the image and the octant labels in the octree, thus replacing the computation of detecting intersections between the octree space and the objects by a table lookup operation. The average ratio of the object volume to the octree volume is found to be greater than 90 percent. The sequential use made of the chosen viewing directions results in a coarse-to-fine acquisition of occupancy information. The number and order of the viewpoints used provides a mechanism for trading accuracy of the representation against the computational effort needed to obtain the representation.**

Index Terms—**Computational efficiency, occupancy map, octree, performance analysis, robotics, silhouette images, three-dimensional representation.**

I. Introduction

THREE-dimensional object representation is of crucial importance to robot vision. Part of the task lies in the generation and maintenance of a spatial occupancy map of the environment. The occupancy map describes the space occupied by objects. Some of the uses of such a representation include robot navigation and manipulation of objects on an assembly line. This paper is concerned with the construction of one such representation, namely, the octree representation, of an object from its silhouette images.

An octree [1], [7], [12] is a tree data structure. Starting with an upright cubical region of space that contains the object, one recursively decomposes the space into eight smaller cubes called octants which are labeled 0–7 (see Fig. 1). If an octant is completely inside the object, the corresponding node in the octree is marked black; if completely outside the object, the node is marked white. If

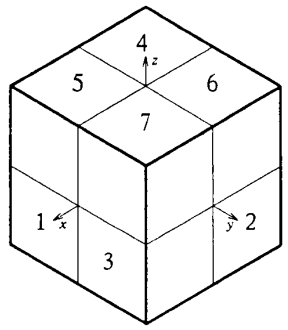


Fig. 1. A cube and its decomposition into octants.

the octant is partially contained in the object, the octant is decomposed into eight suboctants each of which is again tested to determine if it is completely inside or completely outside the object. The decomposition continues until all octants are either inside or outside the object or until a desired level of resolution is reached. Those octants at the finest level of resolution that are only partially contained in the object are approximated as occupied or unoccupied by some criteria.

We call the starting cubical region the "universe cube." The recursive subdivision of the universe cube in the manner described above allows a tree description of the occupancy of the space (see Fig. 2). Each octant corresponds to a node in the octree and the node is assigned the label of the octant. Fig. 2(a) shows a simple object. Fig. 2(b) shows the same object enclosed in the universe cube and Fig. 2(c) shows the corresponding octree. The children nodes are arranged in increasing order of label values from left to right. The black nodes are shown as dark ovals and the white and gray nodes are shown as empty ovals. In practice, of course, the white nodes need not be stored.

This paper addresses the following problem: given a sequence of sihouette views of an object, construct the octree representing the object which gave rise to those views. A given silhouette constrains the object to lie in a cone (for perspective projection) or a cylinder (for orthographic projection) whose cross section is defined by the shape of the silhouette. In this paper, we will consider orthographic projection of an object onto a plane perpendicular to a viewing direction. We will call "extended silhouette" the solid region of space defined by sweeping the silhouette along a line parallel to the viewing direction

Manuscript received June 12, 1986; revised October 5, 1987. This work was supported by the National Science Foundation under Grant ECS 83-52408 and by AT&T Information Systems.

N. Ahuja is with the Coordinated Science Laboratory, University of Illinois, 1101 W. Springfield Avenue, Urbana, IL 61801.

J. Veenstra was with the Coordinated Science Laboratory, University of Illinois, 1101 W. Springfield Avenue, Urbana, IL 61801. He is now with AT&T Communications and Information Systems, Naperville, IL 60566.

IEEE Log Number 8824653.

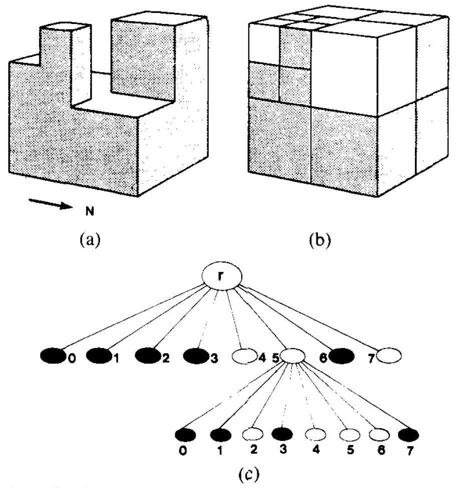


Fig. 2. (a) A simple object. (b) Recursive subdivision of space superimposed on the object in (a) to obtain its representation. (c) Octree for the object in (a); *r* denotes the root node. Black (white) leaves are indicated by darkened (empty) circles.

used in obtaining the silhouette. The object is constrained to lie in the intersection of all extended silhouettes. As the number of silhouettes processed increases, the fit of volume of intersection of the cylinders to the object volume becomes tighter. In our algorithm, we do not perform the intersection explicitly, but infer the octree nodes from silhouette images according to a predetermined table that pairs image regions with their corresponding octree nodes.

An alternate approach, due to Shneier *et al.* [5], [12], to the problem of constructing the octree from silhouettes explicitly tests for the intersection between an octree node and the extended silhouette by projecting the nodes of the tree onto the silhouette image. Consequently, their algorithm can process silhouettes obtained from any viewpoint. Our algorithm restricts viewing direction in order to make the intersection detection process more efficient. Further, their algorithm allows perspective views, whereas our algorithm requires orthographic views. The two algorithms are compared in Section III. Chien and Aggarwal [4] describe an efficient method for constructing an octree for an object from silhouettes of its three orthogonal views. Our method is similar to theirs except that their algorithm provides coarser results since they only use three axial views. The accuracy of the octree describing the object is improved if, in addition to the three orthogonal views, information from other views of the object is also used. Of course, the challenge lies in containing the amount of computation while improving accuracy.

During the development of our image-to-octree algorithm, we felt the need for an octree-to-image display algorithm to visually monitor the accuracy of the octree as it evolves by assimilating object information present in successive silhouette views. We developed an algorithm for this octree-to-object transformation, which is the reverse transformation of constructing the octree from the object discussed in the first part of the paper. The octree-to-image algorithm displays an object represented by a given octree as a line drawing in perspective with hidden lines removed. The line drawing can be displayed corresponding to any arbitrary viewpoint. We do not give details of the display algorithm in this paper, but we use this algorithm to display line drawings of the octrees derived by the octree generation algorithm. A visual comparison of the original objects with those depicted by the line drawing algorithm then serves as a useful test of the correctness of octree generation algorithm.

We also present results on more precise evaluation of the performance of our octree generation algorithm. The performance measure used is the ratio of the true volume of the object to the volume represented by the octree generated. The variation of the accuracy of representation with changes in object orientation, object complexity, and allowed computation time are studied.

The algorithm reported in this paper is part of a three-dimensional representation, manipulation, and navigation system that we are developing. The common theme through the various components of the system is the use of octree repesentation. The problem addressed in this paper is that of initial acquisition of the occupancy information and construction of octree representation from orthographic views. Elsewhere, we have examined the problems of generating octrees from perspective views [13], updating octrees as objects in the scene move [1], [11], [14], and using octress for path planning [6].

In Section II we describe our octree generation algorithm. Section III discusses the performance of the algorithm. Section IV presents experimental results as line drawings of the objects represented by the generated octrees. Section V presents concluding remarks.

II. Octree Generation Algorithm

The algorithm presented in this section provides a method of octree generation which avoids computations of projections and intersections. The viewing directions are defined with reference to the universe cube and are restricted to be those providing one of the six "face" views, one of the twelve "edge" views, or one of the eight "corner" views of the universe cube. Although this allows only thirteen useful views (since views from any two opposite directions provide the same silhouette), the viewpoints are distributed widely in space and together provide significant information to construct a good approximation of the object. All thirteen views are not essential for the algorithm to work. Any subset of the thirteen viewing directions can be used resulting in a cost and accuracy tradeoff discussed in Section III.

Restricting the viewpoints in this manner allows us to find correspondences between the pixels in the two-dimensional silhouette image and the octants in the three-dimensional space that define the octree. The relationship between pixels and octants for an orthographic face view is easily derived so it is described first. Then the relationships between pixels and octants for orthographic edge and corner views are presented. Similar relationships would be difficult to obtain for an arbitrary viewpoint.

A. Face View

A "face view" is the view obtained when the line of sight is perpendicular to one of the faces of the universe cube and passes through the center of the cube. Thus a face x view is the orthographic projection of the object onto the yz plane. A digitized silhouette image would be represented in the computer as a square array of pixels. Pixels having a value of 1 denote the region onto which the object projects. Pixels having a value of 0 represent the projection of free space.

The projection of the cube in Fig. 1 along the x direction results in pairs of octants projecting onto the same region in the image. For example, octants 5 and 4 project onto the upper left quadrant, octants 7 and 6 project onto the upper right quadrant, and so on. (See Figs. 1, 3(a).) This simple relationship between octants and their projections allows the construction of the octree directly from the pixels in a digitized silhouette image.

Given a square array of pixels representing a face x silhouette image, its contribution to the octree can be obtained using the decomposition scheme shown in Fig. 3(a). The quadrants of the silhouette image are processed as if a quadtree were being constructed. A quadrant is recursively decomposed until it is either all ones or all zeros. But instead of adding to the tree only one node per quadrant during recursive decomposition, as is the case with quadtrees, two nodes are added, as in Fig. 3(a). Thus, when a quadrant of the silhouette is further decomposed, each subquadrant could add up to four nodes to the octree instead of one. Fig. 3(b) shows the nodes assigned to the subquadrants.

A similar procedure is used for the other two face views, the only difference being in the labeling scheme for the image quadrants. For example, the labels for the upper left quadrant for the face y view are 7, 5, and for the face z view are 4, 0 (see Fig. 4).

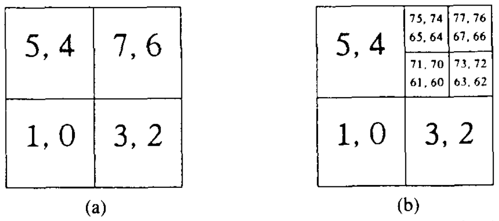


Fig. 3. The labeling scheme for quadrants for the face x view. Each quadrant is assigned two labels (a) instead of one. Each time a quadrant is subdivided, the subquadrants have twice as many labels (b).

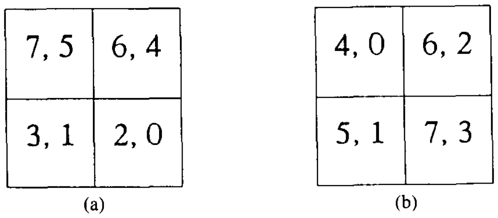


Fig. 4. The labeling scheme for quadrants for the face y view (a), and face z view (b).

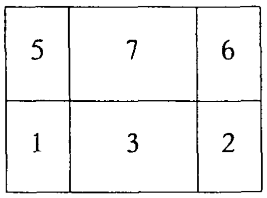


Fig. 5. The cube in Fig. 1 view from edge 3-7.

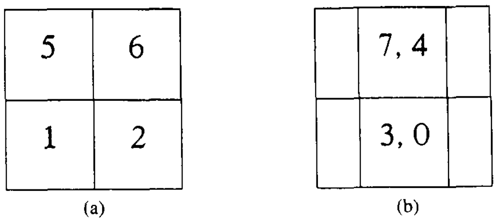


Fig. 6. The decomposition of the image array for the edge 3-7 view into 4 quadrants (a) and 2 center squares (b).

B. Edge View

An "edge view" of a cube is the view obtained when the line of sight bisects an edge of the universe cube and passes through the center of the cube. An edge view is labeled with the two adjacent octants of the universe cube each of which contains one-half of the bisected edge. The octants of the cube in Fig. 1 viewed from edge 3-7 would appear as shown in Fig. 5. Since the octree generation algorithm requires a square array, the elongated image from an edge view must be compressed into a square array. This is accomplished by resampling the digitized image with smaller density along the horizontal direction.

The recursive procedure for constructing an octree from a square array of pixels representing an edge view is similar to procedures for constructing a quadtree. If the square array is all ones or all zeros, then it is marked black or white, respectively. Otherwise it contains some ones and some zeros and it is decomposed recursively in two different ways.

1) It is decomposed into the usual four quadrants, each with one label. The labels depend on which edge is being viewed. For example, the labels for the four quadrants for the edge 3-7 view are given in Fig. 6(a). If a quadrant contains both zeros and ones then it is recursively decomposed.

2) It is decomposed into two center squares and two margins [see Fig. 6(b)]. The center squares are the same size as the quadrants in the first decomposition step. The margins are half the width of the squares and are not used. Each center square has two labels. These are treated in a manner similar to the way the quadrants with two labels for the face view are treated. Whenever a node with one of the two labels is added to the octree, another node with the other label is also added. If a center square contains both zeros and ones, then it is recursively decomposed.

Each time a quadrant or a center square is decomposed, both methods of decomposition described above are used, unless it is a 2×2 square in which case only the first method is used. At each recursive decomposition step the dimension of the quadrants examined is half that at the previous step. When the dimension of a quadrant is 2, it can only be decomposed according to the first method above (otherwise an image pixel would have to be split in half). To prevent the introduction of nonexistent "holes" in the octree during this process, a center square of a 2 × 2 array is marked black if either one of the two quadrants intersecting with it is marked black. For example, if a silhouette image taken from edge 3-7 is decomposed down to a 2 × 2 square and the upper right quadrant (whose label is 6) is black, then, in addition to octant 6, octants 7 and 4, corresponding to the upper central square, will also be marked black.

Fig. 7(a) shows the center squares decomposed into four quadrants (each quadrant inherits two labels from the parent square) and Fig. 7(b) shows quadrant 6 decomposed into two center squares.

C. Corner View

A "corner view" is the view obtained when the line of sight intersects a corner of the universe cube and passes through the center of the cube. Each corner view is labeled according to the corner octant which is closest to the viewer. The silhouette of a cube viewed from one of its corners is a regular hexagon [see Figure 8(a)]. Because the geometry of the corner view silhouette does not correspond naturally with the rectangular image plane, processing a corner view is somewhat more complicated than processing face or edge views where the silhouette of the universe cube is a rectangle.

The silhouettes of the different octants of a cube are all regular hexagons whose regions of intersection are composed of equilateral triangles (see Fig. 9). The silhouette of any octant is a union of subset of these triangles. Thus, the occupancy of an octant can be inferred from the occupancy of an appropriate set of triangles. The occupancy of the universe cube can be inferred from the occupancy of the six major triangular cells [Figure 8(a)]. To generate nodes at lower levels in the octree, the triangles of interest are the result of recursive triangular decomposition [see Fig. 8(b) and 8(c)] of the six major triangles. Therefore, the processing of a corner view is done in two phases. First, six quadtrees are generated from the digitized image in such a way that each quadtree represents one of the six triangular sections of the regular hexagon silhouette of the universe cube. Second, the octree is constructed from the six quadtrees. The quadtrees and octree are constructed recursively.

To construct the quadtrees, the universe hexagon is divided into six triangles labeled 0–5 as shown in Fig. 8(a). Each triangle is recursively subdivided and its nodes are labeled 0–3 according to one of the two schemes shown in Fig. 8(b) and (c). The orientation of the triangle determines which labeling scheme is used. In both labeling schemes, quadtree node 0 is in the center and node 2 is in the lower corner. The recursive subdivision of a triangle into four subtriangles continues until the distance between the centers of adjacent triangles is less than or equal to the distance between pixels in the digitized image. The image pixel nearest the center of the triangle at the lowest level in the quadtree determines whether the corresponding quadtree node is black or white. The color of a quadtree node above the lowest level is determined by the colors of its children. If the children of a quadtree node are all white or all black, then the parent node is assigned the same color as its children, and the children are removed from the tree. Otherwise, some children are white and some are black so the parent is assigned the color gray. Each quadtree is given the label (0–5) of the major triangle it represents. Once the six quadtrees are constructed, the raw silhouette image data are no longer needed. The octree is generated directly from the quadtrees; quadtrees are used only as an intermediate step in the octree construction.

The octree construction is best explained by showing how the color of a particular octant is determined. Fig. 9

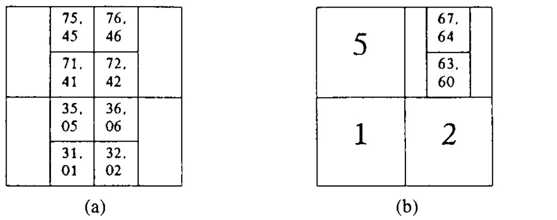


Fig. 7. Further decomposition of center squares into quadrants (a), and of the upper right quadrant into center squares (b).

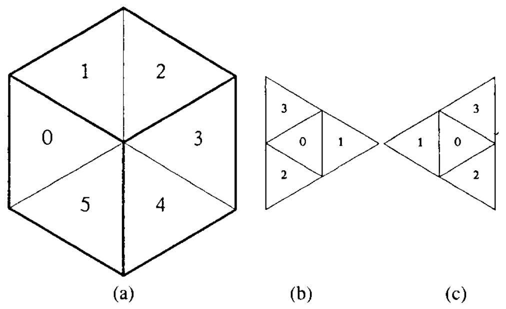


Fig. 8. The division of the universe hexagon into six triangular sections (a). Each section is represented by a triangular quadtree. The labels of the quadtree nodes for each of the two orientations of a triangle (b), (c).

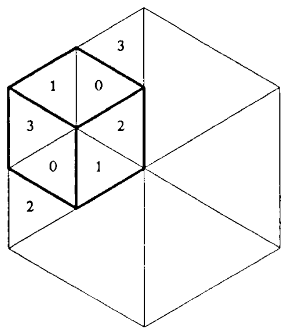


Fig. 9. The projection of octant 5 (shown in bold lines) which overlaps triangular quadtrees 0 and 1.

shows the projection of octant 5 in relation to the projection of the universe cube viewed from octant 7. Octant 5 also projects as a hexagon and overlaps triangular quadtrees 0 and 1. Octant 5 is labeled white if the nodes of quadtrees 0 and 1 covered by the projection of octant 5 are all white. In this case the relevant nodes that need to be examined are: nodes 0, 1, 3 of quadtree 0 immediately below the root level, and nodes 0, 1, 2 of quadtree 1 immediately below the root level. If these nodes are not all white nor all black and the predetermined maximum depth for the octree has not been exceeded, then octant 5 is recursively subdivided and the appropriate children of these nodes in the quadtree at the next lower level are examined. This process continues until all nodes are labeled black or white. The same procedure is used for the other octants, the only difference being in the set of quadtree children which needs to be examined. The details for the actual procedures used can be found in [2].

D. Octree Intersection

One octree is generated from the silhouette obtained from each viewpoint. The resulting octrees are intersected a pair at a time to find the global intersection octree. We experimented with other algorithms where the already generated octree is used to guide the search of the next silhouette image, to reduce the number of generated nodes in constructing the updated octree. However, we found that the savings obtained were marginal on an average, and generally independent generation of octrees leads to a simpler method with comparable performance.

III. Performance of the Octree Generation Algorithm

In this section, we will discuss some issues regarding the performance of our octree generation algorithm described in the previous section. First and foremost is the issue of the accuracy of the object shape captured by the octree generated. Next is the question whether any tradeoff is possible between the accuracy of the representation obtained and the complexity of the computation performed to obtain it. Another aspect of the algorithm concerns the execution time, i.e., the efficiency with which the algorithm implements the necessary computations.

A. Defining Accuracy

An algorithm of the kind described in this paper that attempts to reconstruct an object shape from its silhouette views suffers some inherent limitations. First, the algorithm cannot *detect* those three-dimensional features of the object that are lost during the projection process. For example, no surface concavities are registered. Thus, at its very best the algorithm suffers from such *detection errors* that occur during the data acquisition phase; the algorithm can provide a representation of only a *bounding* volume of the object. Then, there are the inaccuracies arising from the limitations of the representation itself. In our case, the octree of a given depth will have an associated error due to the 3-D spatial quantization. An increase in the allowed depth (resolution) of the octree will reduce the error, possible to zero. Finally, there is the usual 2-D digitization error in obtaining the image. This error can also be reduced by using higher resolution images.

Beyond these general sources of inaccuracy is the restriction of the viewing directions used specifically by our algorithm. This limits the amount of available information about the object, leading to differences between the shape of the object captured by the octree and the original object. This error in shape representation may serve to measure the accuracy with which an object is approximated using silhouettes obtained from the given viewing directions. The approximation, of course, depends on the shape and orientation of the object viewed.

Thus, one possible measure of accuracy for a set of viewing directions is the ratio of the volume of the smallest object which could give rise to a given set of silhouettes to the volume of intersection of the extended silhouettes. This measure is a fraction since the volume of the intersection of extended silhouettes of an object contains that object. Even if the object is convex, the volume of the object is probably smaller than the volume of the intersection. This worst-case definition means that if a given set of silhouettes has an accuracy measure of 90 percent then the volume of the actual object can be no less than 90 percent of the computed volume. Some restrictions must be placed on the object shape (such as requiring it to be convex) to prevent the smallest object from having an arbitrarily small volume. Even with restrictions, however, the accuracy measure can be very low if only a few views are used. For example, there exist convex objects smaller than a unit cube which have unit squares as silhouettes when viewed along three orthogonal directions. The projection of a tetrahedron oriented so that its four vertices coincide with four vertices of the unit cube is a unit square when viewed along any direction perpendicular to the face of the unit cube. The tetrahedron would be represented as a cube by the algorithm since the intersection of extended silhouettes is a cube. The volume of the tetrahedron, however, is only one-third the volume of the cube. Since a tetrahedron inscribed in a unit cube is the smallest convex object whose three orthogonal silhouettes are unit squares, the accuracy measure for that set of three silhouettes is 33.3 percent.

The above definition of accuracy may be of only theoretical interest; the difficulty of finding the smallest object for each set of silhouettes makes this definition impractical. An alternate approach is to empirically measure the performance of a chosen set of viewing directions on a suitable set of objects. The measure of accuracy for a given object is the observed ratio of the volume of the object to the volume of the intersection of the extended silhouettes of the object. For example, a sphere would have an accuracy measure equal to the ratio of its volume to the volume of the intersection of circular cylinders containing it, where the axes of the cylinders coincide with the viewing directions. Using this measure of accuracy, three orthogonal views of a sphere would yield an accuracy of 88.9 percent. The accuracy of nine face and edge views is approximately 98.7 percent.

Except for the sphere, the accuracy of a set of viewing directions for a given object is dependent on the object's orientation. In one orientation, the tetrahedron yields an accuracy of 33.3 percent for three orthogonal views; in another orientation, the accuracy is 100 percent. In fact, only two orthogonal views of the tetrahedron are necessary to represent it exactly. To obtain the average performance of all orientations, a Monte Carlo simulation experiment can be performed to measure the desired ratio of volumes over a large number of randomly chosen orientations. Then, for a given set of objects, the measurement of accuracy for a set of viewing directions is the estimated expected value of the ratio of the object volume to the constructed volume for a randomly selected object at a randomly selected orientation.

B. Measuring Accuracy

How should the objects constituting the test set be chosen? One way to resolve this question is to use objects having shapes used as primitives for three-dimensional representations, e.g., generalized cones. A generalized cone is defined by a space curve spine and a planar cross section which is swept along the spine according to a sweeping rule. The sweeping rule determines how the cross section changes as it is translated along the spine. Fig. 10 shows a sample of generalized cones used by Brooks [3] as primitive volume elements.

The measure of accuracy can then be computed as the average of the results of a large number of executions of the following three step procedure. First, an arbitrary object from the chosen set and a random orientation are selected. Second, the object is projected along each viewing direction to provide a set of silhouette images. Finally, the octree is constructed and the corresponding object volume computed. The ratio of the actual to the computed volume is the desired result for the chosen object and orientation.

In our experiments, we used the set of geometric objects shown in Fig. 10 with the following modifications. In place of 10(f) we used a circular cone; in place of 10(g) we used a regular cube; and in place of 10 (i) we used a regular pyramid. We further added the sphere and a small cube to the set of objects giving us eleven objects on which to observe the performance of the octree generation algorithm. These objects were viewed at random orientations to determine an average accuracy resulting from the thirteen viewing directions described in this paper. Fig. 11 lists the objects used along with their mathematical definitions and volumes. The cube is the largest object in the list and all other objects fit inside it. This was done so that the silhouettes of all the objects would be guaranteed to fit on the simulated image screen.

For each object in the test set, one hundred random orientations were selected. For each orientation, the thirteen digitized silhouette images in the form of binary-valued

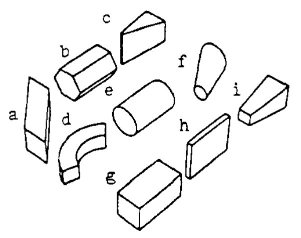


Fig. 10. A selection of generalized cones taken from [3]. A rectangular prism (a), an octagonal prism (b), a wedge (c), an arc (d), a cylinder (e), a truncated cone (f), a rectangular solid (g), a slice (h), and a truncated pyramid (i).

Objects Used in Performance Analysis

Name	Definition	Volume
Cube	x, y, z *in* $[-1, 1]$	8.0
Cylinder	$x^2 + y^2 \leqslant 1$, and z *in* $[-1, 1]$	6.2832
Cone	$x^2 + y^2 \leqslant \frac{(z-1)^2}{4}$, and z *in* $[-1, 1]$	2.0944
Sphere	$x^2 + y^2 + z^2 \leqslant 1$	4.1888
Slice	x, z *in* $[-1, 1]$, and y *in* $[-0.25, 0.25]$	2.0
Pyramid	$x^2 \leqslant \frac{(z-1)^2}{4}$, and $y^2 \leqslant \frac{(z-1)^2}{4}$, and z *in* $[-1, 1]$	2.6667
Wedge	$x^2 \leqslant \frac{(z-1)^2}{4}$, and $y^2 \leqslant \frac{(z-1)^2}{4}$, and z *in* $[0, 0.8]$	0.6293
Octagonal Prism	x, y, z *in* $[-1, 1]$, and $\left\| \begin{array}{lll} \text{if } \sqrt{2}-1 \leqslant x \leqslant 1 & \textit{then} & x-\sqrt{2} \leqslant y \leqslant -x+\sqrt{2} \\ \text{if } -\sqrt{2}+1 \leqslant x < \sqrt{2}-1 & \textit{then} & -1 \leqslant y \leqslant 1, \\ \text{if } -1 \leqslant x < -\sqrt{2}+1 & \textit{then} & -x-\sqrt{2} \leqslant y \leqslant x+\sqrt{2} \end{array} \right.$	6.6274
Rectangular Prism	x *in* $[0, 1]$, and y, z *in* $[-1, 1]$, and $2y - 1 \leqslant z \leqslant 2y + 1$	2.0
Arc	$x, y \geqslant 0$, and $0.25 \leqslant x^2 + y^2 \leqslant 1$, and z *in* $[-0.25, 0.25]$	0.2945
Small Cube	x, y, z *in* $[-0.125, 0.125]$	0.0156

Fig. 11. List of primitive objects used to test the accuracy of the octree generation algorithm.

square arrays were computed assuming that the object is placed with its center at the origin. The octree was then constructed from the thirteen silhouettes and the ratio of the object volume to the octree volume was computed.

The silhouettes were generated on the computer from mathematical definitions of the test objects and a simulated 128 × 128 digitized image was created. To determine the value of an image pixel, a line was constructed perpendicular to the image plane and passing through the center of the image pixel. If this line intersected the test object the pixel value was set to 1, else it was set to 0. This process was repeated for each of the 16,384 pixels to generate a digitized silhouette image.

Since the octree constructed from extended silhouettes represents an object larger than or equal to the actual object, the ratio of the object volume to the octree volume should always have a value less than or equal to 1. Due to digitization error, sometimes a pixel on the border of a silhouette is marked empty (set to zero) when it is actually partially covered by the silhouette. This can lead to lost

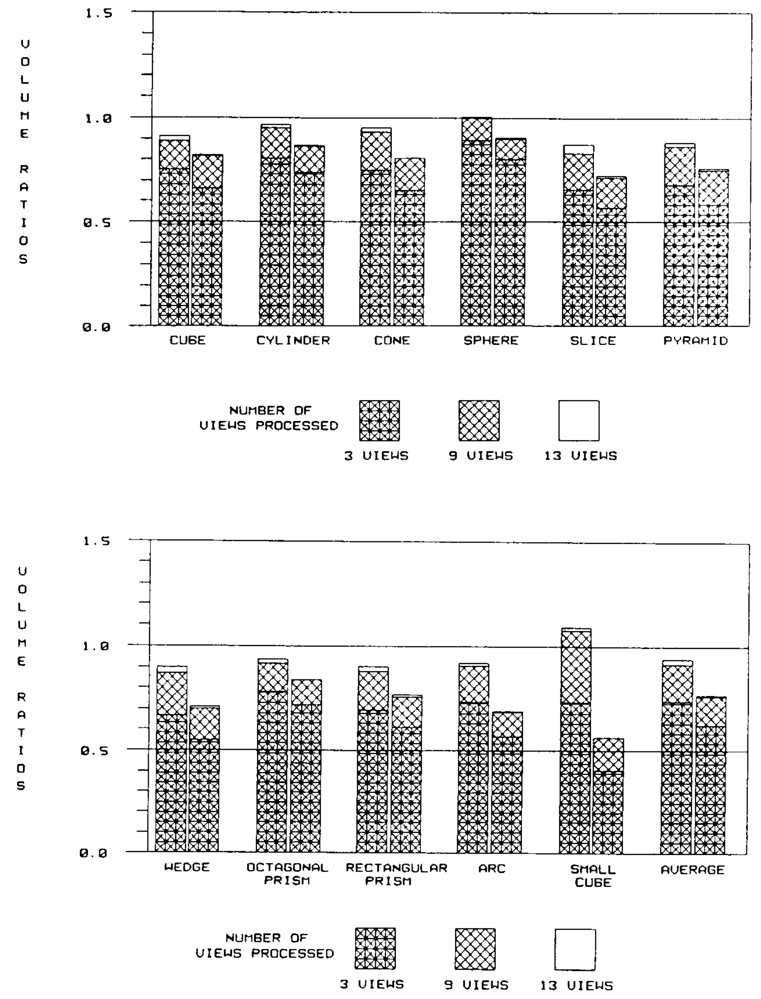


Fig. 12. Ratios of the volume of a randomly oriented object to the volume represented by the octree generated from the object's silhouettes. For each object the left bar corresponds to the case of no preprocessing of silhouettes and the right bar corresponds to preprocessing.

volume in the object represented by the octree since the method of taking intersections of extended silhouettes always yields a smaller (or possibly the same size) octree as each new silhouette is processed. Once an octant is removed from the octree, it is never returned. Similarly, when a pixel partially covered by the silhouette is marked full, it may add extra volume to the final octree. Specific applications may assume complete occupancy or no occupancy for partially occupied pixels to minimize the effect of digitization error. Thus, when visibility of scene features is to be predicted, and missing visible features has a high cost, it is safer to mark the partially occupied pixels as empty. On the other hand, in collision avoidance applications, it is safer to assume that the partially full pixels represent completely occupied space. Alternately, some measure of the partial occupancy may be stored at the leaf nodes of the octree, which can then be interpreted in a way chosen by the application algorithm.

This impact of digitization error is more pronounced for smaller objects since the removal of a fixed chunk of the object represents a larger portion of its volume. To illustrate the effect of digitization error, we consider here both the original silhouette as well as expanded silhouette, i.e., we "grow" the silhouette by locating every pixel with value 1 and setting its neighbors (in all eight directions) to 1. This guarantees that the octree will not represent an object smaller than the actual object as desired by path planning algorithms. We have included a small cube, 1/512 of the volume of the universe cube, in the set of test objects to assess the impact of silhouette expansion on accuracy.

Fig. 12 shows the average accuracies over a sample of one hundred random orientations for each object. The results for both the original silhouettes and the expanded silhouettes, after preprocessing, are shown for comparison. Above each object's name are shown two bars. The left bar depicts the accuracy using silhouettes with no preprocessing, and the right bar depicts the accuracy using expanded silhouettes. The average accuracy for the entire set of test objects with no silhouette preprocessing 93.7 percent and with silhouette expansion is 76.5 percent.

C. Accuracy-Computation Tradeoff

In many cases it may suffice to have occupancy information which is even less detailed than that provided by the 13 directions. Of greater importance may be the speed at which the octree is generated. Under such conditions it is of interest to know how the accuracy degrades with a decrease in the number of directions. In Fig. 12 we have indicated the ratios of actual volumes to the octree volumes corresponding to 1) only face views (3 directions), 2) face and edge views (9 directions), and 3) face, edge, and corner views (13 directions). The successive increments in bar heights represent the additional accuracy contributed by the additional silhouettes. The graphs show that for all the objects tested, the additional six edge views contribute a significant amount of new information. The addition of the four corner views to the nine face and edge views, however, increases the accuracy only slightly.

1) Parallel Versus Serial Computation: For any algorithm that constructs a volumetric description from object silhouettes, the silhouette information from multiple viewpoints may be acquired either using one camera and changing viewpoints with time, or by using multiple cameras located at fixed viewpoints and acquiring the multiple silhouette data in parallel. The former leads to necessarily serial computation since the input data arrives as the camera moves. The frequency with which successive silhouette views may be acquired is determined by the relative magnitudes of the processing time T for a single view, and the camera velocity. If T is small then closely separated viewpoints may be used to eventually obtain a relatively accurate octree representation, and the overall octree generation time is determined by the time taken by the camera to traverse viewpoints. If T is large, then the overall computation time is proportional to the number of silhouettes acquired by one camera.

An algorithm that allows arbitrary viewpoints is useful when accuracy of the representation is important and the time delay due to camera motion is insignificant. In such cases, one can use partial knowledge of the object shape to make a judicious choice of next viewpoint so as to best reveal the hitherto unknown part of the object shape. An algorithm such as that of Shneier *et al.* would be very useful under such conditions. On the other hand, if the goal is to generate the octree representation fast, albeit less accurately, then this can be achieved by using multiple cameras at different viewpoints. This eliminates any delays due to camera movement, and opens up the possibility of parallel processing of silhouette data. Our algorithm uses a fixed set of viewpoints, suitably chosen so as to make the construction of octrees from images efficient. Thus, our algorithm is more appropriate for this second scenario.

2) Coarse-to-Fine Computation: When the computation time per silhouette view is larger than the time taken by the camera to move to the next viewpoint, or when the time taken for the intersection of multiple silhouette data in a multiple camera setup can be reduced by reducing the number of silhouettes, it may be desirable to have a mechanism of tradeoff between the accuracy of the constructed octree and the time taken to construct it. Both our algorithm, and an algorithm that accepts general viewpoints [5] would benefit from availability of such a mechanism.

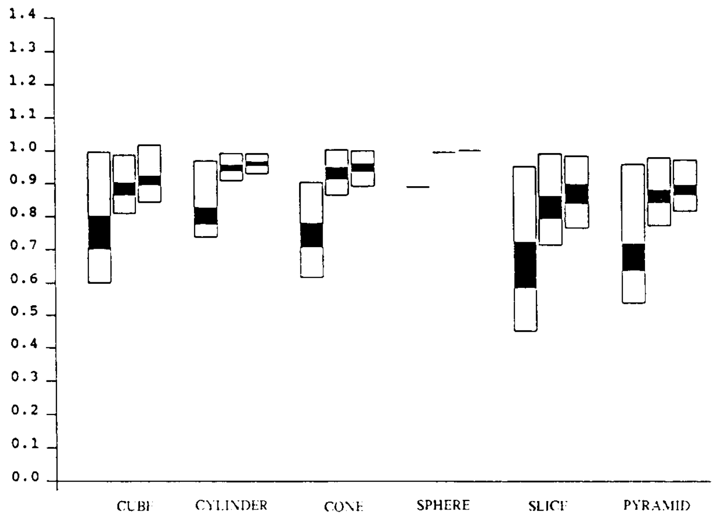
The choice and order of the viewing directions used in our algorithm constitute a coarse-to-fine mechanism of acquisition of occupancy information. The face views act as coarse sensors of occupancy. Their spatial orthogonality contributes to independence of the information they provide. The additional edge views are well separated and reveal occupancy of the space halfway between the face viewing directions. Thus, they provide finer grain occupancy information. The corner views further increase the density of viewing directions fairly isotropically since the corner viewing directions are located far away from the face and edge viewing directions. Using the three sets of viewing directions in different orders results in different, near optimal ways of acquiring the silhouette information in a coarse-to-fine manner. For example, if the order (face, edge, corner) is used, then one can stop after using 3, $3 + 6 = 9$, or $3 + 6 + 4 = 13$ directions: if the order (edge, face, corner) is used, then one can stop after 6, 9, or 13 directions. For certain numbers of allowed viewing directions, e.g., 8, not all directions in a given subset may be used, thus, making information acquisition less isotropic.

D. Stability

The measure used in the performance analysis of the derived octree representation is an average computed over a large number of random orientations of the objects. Thus, the results correspond to the expected fraction of the volume represented by the octree that is actually occupied by the object. *In practice*, we may derive the octree for a given *single* orientation of the object, which may be random. The question then arises as to how reliable the resulting representation is. In other words, how stable is the representation from orientation to orientation even though we know how the representation performs on an average over many orientations. Fig. 13 shows the observed maximum and minimum values of the measure over all 100 observations for each object and for the case of no preprocessing. Also shown are the average values and the standard deviations of the values. Clearly, the smaller the standard deviation, and the smaller the differences among the maximum, minimum, and the average values, the better the stability of the derived octree representation. The improvement in the stability of the representation with an increasing number of views can be seen in Fig. 13. Figs. 16(a)–(h) show a graphic display of the objects represented by the constructed octrees.

E. Complex Objects

An inadequacy of the above performance analysis is the simplicity of the objects analyzed. Originally, these ob-



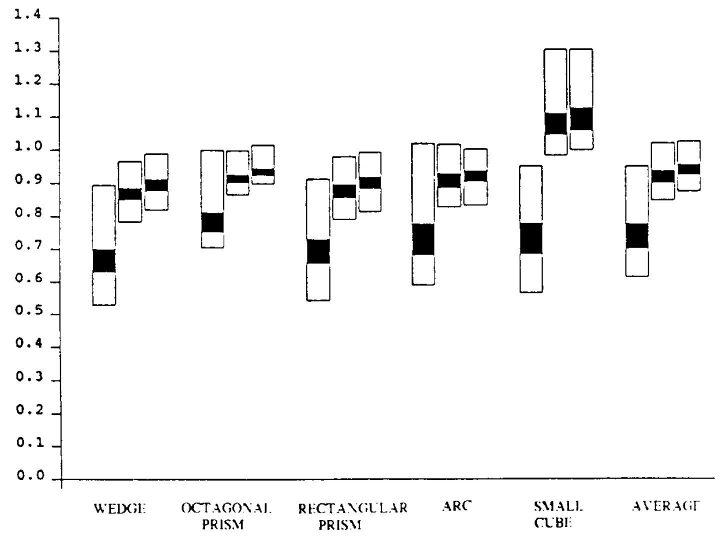


Fig. 13. Stability characteristics of the algorithm. For each object the three bars correspond, respectively, to 3, 9, and 13 views, with no silhouette preprocessing. Each bar extends from the minimum to the maximum observed measure value over the 100 observations made. The band in the bar is centered at the observed average value and has a thickness of the observed standard deviation.

jects were chosen because they are diverse and fairly powerful to serve as geometrical primitives to construct arbitrary objects. However, the construction entails *simultaneous* presence of these objects in the octree space, as components of the larger object whose octree representation is to be derived. Their relative configuration is determined by the complexity of the object to be constructed. For example, consider the object shown in Fig. 14 consisting of the volumetric primitives of Fig. 10. The spatial configuration of the components leads to their mutual occlusion when the object is viewed from an arbitrary direction. Thus, the already incomplete information in the silhouettes about the object is further confounded by self-occlusion. Of course, unlike surface concavities, the information about occupancy of regions self-occluded from a viewpoint may be recovered if other, allowed directions are more revealing. The availability of a large number of viewing directions assumes increased importance in this context.

To test the performance of our algorithm over more complex objects than shown in Fig. 10, we conducted experiments with the object shown in Fig. 14. Fig. 15 shows the results analogous to Fig. 13. Fig. 16(j) shows a graphic display of the self-occluding object in Fig. 14 as represented by the constructed octree. The upright rectangular solids have reproduced well without any staircase effect because these are oriented with their faces parallel to the faces of the universe cube. Parts of the curved sur-

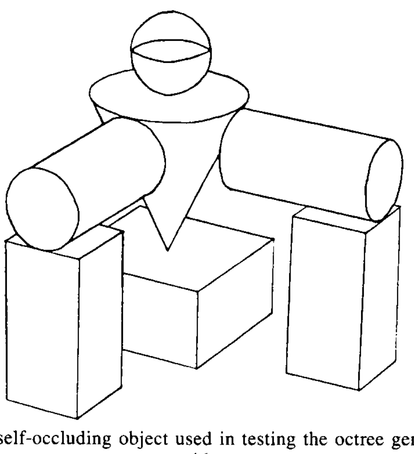


Fig. 14. A self-occluding object used in testing the octree generation algorithm.

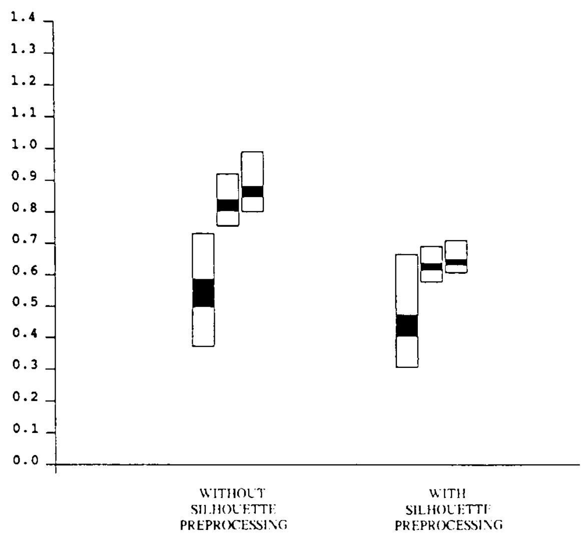


Fig. 15. Graphical depiction of the stability characteristics of the self-occluding object shown in Fig. 14. The three bars correspond to the 3, 9, and 13 views. Each bar extends from the minimum to the maximum observed value over the 100 observations made. The band in the bar is centered at the observed average value and has a thickness of the observed standard deviation. The three bars on the left show the results with no silhouette preprocessing and the three bars on the right show the results with silhouette preprocessing.

faces of the circular cylinders are lost since these parts were occluded by the rectangular solids in all the silhouette views used.

F. Experimental Details

The algorithms were implemented in C on a VAX 11/780 computer (see [2] for details). Some simple modifications were made to the edge view algorithm to improve its efficiency. The C programs were timed using test data in the form of 64 × 64 arrays representing binary images of varying complexity. The average CPU time spent in the octree generation procedures was recorded for face and edge views and was found to increase linearly with the number of nodes in the octree for a fixed image size.

The accuracy of the octree increases as its depth increases. Whenever the image resolution is finer than the maximum allowed tree depth, then octants at the lowest level in the tree represent square regions of image pixels instead of individual pixels. As discussed in Section III-B, when a square region is not of uniform value, its degree of nonuniformity may be used to determine the color of the representative octree node. In applications such as path planning among obstacles, it is safer to overestimate the object sizes rather than underestimate them. In our case, we chose to label the node black if at least 1/4 of the pixels were black. This resulted in a good approximation without being too conservative. The depth of the generated octree in our experiments is determined by a parameter under user control. We set the tree depth so that the deepest node corresponds to a pixel in the face views. Before closing, we would like to mention that the current implementation suffers from the problem that nodes in a newly created set of siblings may have identical color in which case they may have to be deleted. This results in wasted space and time.

IV. Visual Evaluation of Performance

During our work on the octree generation algorithm, it was necessary to monitor the accuracy of the octree representation constructed at different stages of development. First, we did this by printing each node in the octree with its associated "black" or "gray" label ("white" or empty nodes were not stored), and then verifying by hand that the octree was correct. As the octrees became larger, however, it became necessary to be able to view directly the object which the octree represented. This section describes an algorithm we developed for this purpose. The algorithm produces a line drawing of an object represented by on octree. The object is drawn with hidden lines removed. An alternative method of displaying the object represented by an octree is described by Meagher [9], [10]. His algorithm produces a surface display from octree after hidden surface removal. However, surface displays depend upon light source positions. In addition, many output devices cannot draw shaded surfaces. A line drawing representation, on the other hand, captures the essential details of the object structure in the form of edges since the objects are polyhedral. We therefore chose to display the objects represented by the octree as line drawings which can be easily drawn. We will not give the details of the display algorithm here; they can be found in [2], [8].

The octree generation algorithm followed by the line drawing generation algorithm should provide a display of the original object. Any differences between an object and its line drawing represent the approximations and errors involved in octree generation, and thus serve as a quick method of evaluating the performance of the octree generation algorithm.

Figs. 16(a)-(h) show the line drawings generated by our algorithm for the octrees of the test objects in Fig. 10. Fig. 16(i) is a line drawing generated from an octree that was obtained from gray level, silhouette images of a cof-

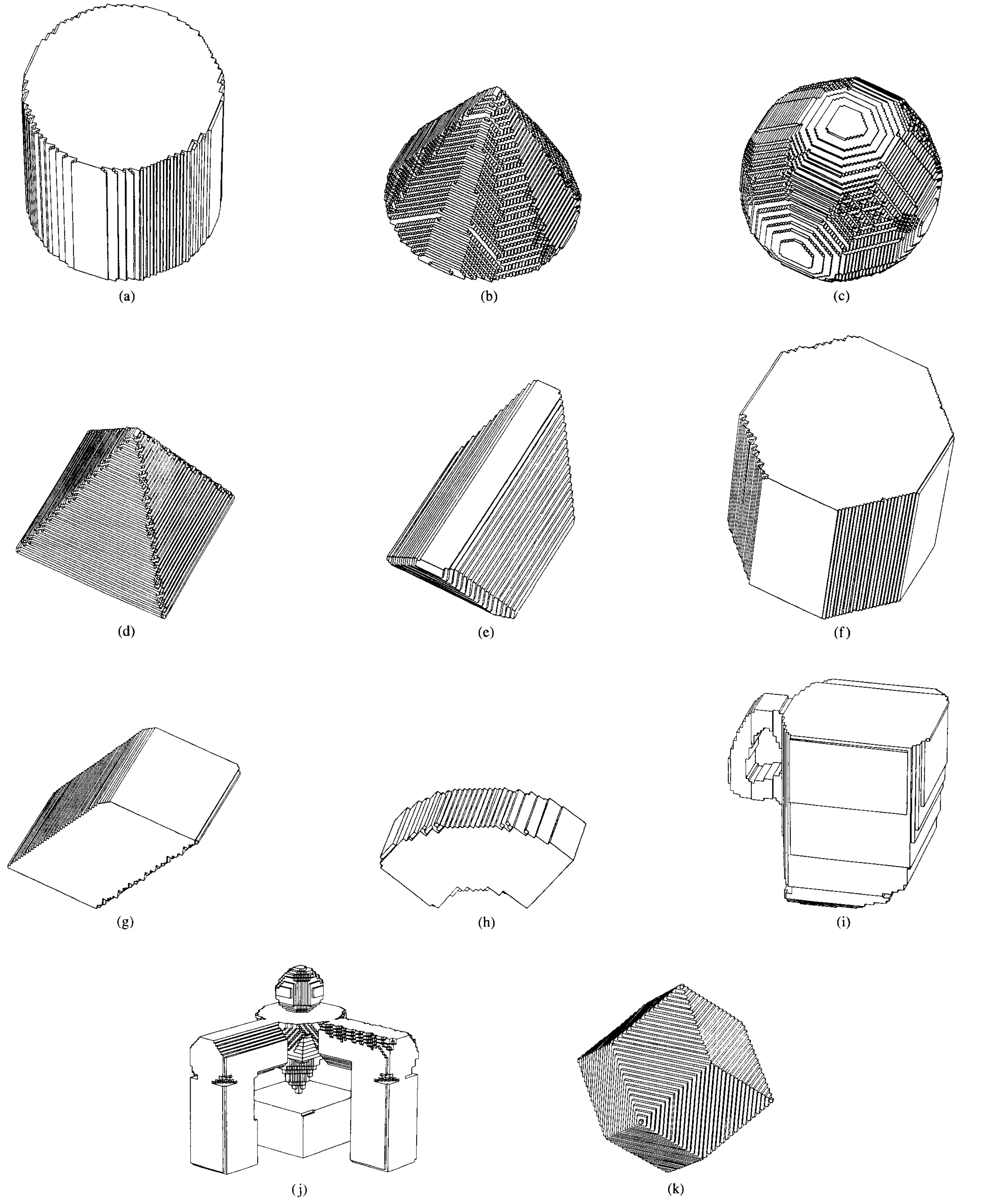


Fig. 16. Line drawing depictions of octree representations of several objects derived by the octree generation algorithm. (a) Cylinder. (b) Cone. (c) Sphere. (d) Pyramid. (e) Wedge. (f) Octagonal prism. (g) Rectangular prism. (h) Arc. (i) Coffee cup. (j) Self-occluding object. (k) A diamond shaped object.

fee cup. Fig. 16(j) shows the drawing for the octree of the self-occluding object in Fig. 14. Fig. 16(k) shows an object whose silhouette is a diamond when viewed from any face of the universe cube which contains it. The line drawing algorithm was executed on a VAX and the output sent to a QMS laser printer.

V. Summary

We have presented an algorithm to generate the octree representation of an object from silhouette images taken from a set of thirteen viewing directions. These viewing directions are parallel to three orthogonal faces, six face-diagonals, and four long-diagonals of an upright cube. Each silhouette of an object is first extended into a cylinder parallel to the viewing direction, and the corresponding octree is constructed. An intersection is performed on the octrees generated from the silhouettes to obtain an octree representing the space occupied by the object. The octree for each silhouette image is computed efficiently by a recursive quadtree decomposition of the image, and identification of the occupied octree nodes from a table listing corresponding pairs of image windows and octree nodes. In actual applications, the requirements of the thirteen images may be met very simply by placing cameras in fixed positions in a cubical room, namely, at centers of walls, edges and corners, all pointing at the room center and taking orthographic images. We have also run performance tests on the accuracy of the octree and concluded that thirteen silhouette views can provide enough information for a good approximation of the object. The three sets of viewing directions (face, edge, and corner) act as coarse-to-fine, information acquisition probes. Fewer than thirteen directions may be used to reduce computation time in exchange for reduced accuracy of the representation generated.

Although a general view algorithm allows an arbitrary viewpoint, this generality requires an explicit computation of the volume of intersection for determining the octree nodes corresponding to extended silhouettes. The corresponding intersection tests are more complex and may require greater computation time than the direct construction of octree nodes from image pixels used in our approach. Moreover, since silhouette images taken from viewing directions which are widely spaced yield more information, in general, than do silhouette images which are close together, and since the thirteen viewing directions used in our algorithm are distributed widely about the entire octree space, it is unlikely that a large amount of additional information will be obtained using a silhouette taken from a viewpoint which falls at an intermediate position. The results of our experiments bear out this expectation, where the accuracy for the thirteen views used in our algorithm is over 90 percent (with no silhouette preprocessing). Thus, there may be only a marginal gain in accuracy by using a general view algorithm, especially considering the inherent limitations discussed earlier of any shape-from-silhouette reconstruction algorithm. Given that the octree representation is useful only as a coarse occupancy map (for applications such as rough path planning), and is not intended as a representation of fine shape details, the accuracy provided by the thirteen viewing directions may suffice. This may be particularly important if the general-view algorithm turns out to be more expensive than our alogrithm in processing silhouette views. Our algorithm requires orthographic views in order to make use of the *a priori* relationships between image space and octree nodes. These relationships do not directly extend to perspective views.

We have used a display algorithm to produce a line drawing of an object from its octree. The object is drawn with cracks and hidden lines removed. The line drawing produced may be used to check the performance of the octree generation algorithm by comparing the original and the represented object.

One of the inherent weaknesses of using silhouette images to obtain 3-D information is that surface concavities cannot be identified. A subject worthy of study is the use of range information, in addition to the silhouettes, to help identify these concavities. The range information could be obtained from such sources as sonar devices or laser range finders.

Several applications of this research suggest themselves. One possibility is the construction of a working system of thirteen cameras to monitor the objects in a room. The resulting octree representation of the occupied space in the room might be used to guide a robot. Another possibility is to point cameras at a spot above a conveyor belt and as objects pass through the octree space defined by the camera viewing directions, perform some task based on the volume or shape of the octree approximated objects. If cameras are a scarce or precious resource, one could experiment with using multiple or movable mirrors (in conjunction with a single camera) to obtain the silhouette images. Finally, one could experiment with mounting a camera on a moving vehicle to obtain sequential images of an object or an environment for the purpose of maintaining a representation of the workspace, planning paths, and locating objects.

Acknowledgment

Thanks are due to anonymous reviewers who made very helpful comments on the manuscript.

References

[1] N. Ahuja and C. Nash, "Octree representations of moving objects," *Comput. Vision, Graphics, Image Processing*, vol. 26, pp. 207-216, 1984.

[2] N. Ahuja and J. Veenstra, "Octree generation and display," Coordinated Sci. Lab., Univ. Illinois, Tech. Rep. UILU-ENG-86-2215, May 1986.

[3] R. Brooks, "Symbolic reasoning among 3-D models and 2-D images," *Artificial Intell.*, vol. 17, pp. 285-348, 1981.

[4] C. H. Chien and J. K. Aggarwal, "Volume surface octrees for the representation of 3-D objects," *Comput. Vision, Graphics, Image Processing*, vol. 36, pp. 100-113, 1986.

[5] T. H. Hong and M. Shneier, "Describing a robot's workspace using a sequence of views from a moving camera," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-7, pp. 721-726, Nov. 1985.

[6] Y. Hwang and N. Ahuja, "Path planning using a potential field representation," Univ. Illinois Coordinated Sci. Lab., Tech. Rep. UILU-ENG-88-2251, Oct. 1988.

[7] C. L. Jackins and S. L. Tanimoto, ''Oct-trees and their use in representing three-dimensional objects,'' *Comput. Graphics Image Processing*, vol. 14, pp. 249-270, 1980.

[8] J. Veenstra and N. Ahuja, ''Line drawings of octree-represented objects,'' *ACM Trans. Graphics*, to be published.

[9] D. Meagher, ''Efficient synthetic image generation of arbitrary 3-D objects,'' in *Proc. IEEE Conf. Pattern Recognition and Image Processing*, Las Vegas, NV, June 14-17, 1982, p. 473.

[10] —, ''Geometric Modeling Using Octree Encoding.'' *Comput. Graphics Image Processing*, vol. 19, p. 129, 1982.

[11] W. Osse and N. Ahuja, ''Efficient octree representation of moving objects,'' in *Proc. 7th Int. Conf. Pattern Recognition*, Montreal, P.Q., Canada, July 30-Aug. 2, 1984, pp. 821-823.

[12] M. Shneier, E. Kent, and P. Mansbach, ''Representing workspace and model knowledge for a robot with mobile sensors,'' in *Proc. Seventh Int. Conf. Pattern Recognition*, Montreal, P.Q., Canada, July 1984, pp. 199-202.

[13] S. Srivastava and N. Ahuja, ''An algorithm for generating octrees from object silhouettes in perspective views,'' in *Proc. IEEE Workshop Computer Vision*, Miami Beach, FL, Nov. 30-Dec. 2, 1987, pp. 363-365.

[14] J. Weng and N. Ahuja, ''Octrees of objects in arbitrary motion: Representation and efficiency,'' *Comput. Vision, Graphics, Image Processing*, pp. 167-185, Aug. 1987.

Narendra Ahuja (S'79-M'79-SM'85), for a photograph and biography, see this issue, p. 136.

Jack E. Veenstra was born in Grand Rapids, MI, on April 4, 1961. He received the B.A. degree in mathematics and computer science from Calvin College, Grand Rapids, MI, and the M.S. degree in computer science from the University of Illinois at Urbana-Champaign in 1986.

Since February, 1986, he has been working for AT&T Communications and Information Systems in Naperville, IL. His current interests include operating systems and languages for parallel processing machines.