CVGIP: IMAGE UNDERSTANDING
Vol. 58, No. 1, July, pp. 23–32, 1993

Rapid Octree Construction from Image Sequences

RICHARD SZELISKI

Digital Equipment Corporation, Cambridge Research Lab, One Kendall Square, Building 700, Cambridge, Massachusetts 02139

Received April 15, 1991; accepted September 18, 1992

The construction of a three-dimensional object model from a set of images taken from different viewpoints is an important problem in computer vision. One of the simplest ways to do this is to use the silhouettes of the object (the binary classification of images into object and background) to construct a bounding volume for the object. To efficiently represent this volume, we use an octree, which represents the object as a tree of recursively subdivided cubes. We develop a new algorithm for computing the octree bounding volume from multiple silhouettes and apply it to an object rotating on a turntable in front of a stationary camera. The algorithm performs a limited amount of processing for each viewpoint and incrementally builds the volumetric model. The resulting algorithm requires less total computation than previous algorithms, runs in close to real-time, and builds a model whose resolution improves over time.

1. INTRODUCTION

This paper describes a new approach to creating 3D models of objects from multiple views. In our system, an object rotates on a turntable in front of a stationary camera. We use the silhouettes—the binary classification of each image into object and background—to compute a bounding volume for the object. This algorithm enables us to obtain a quick and rough model of the object in close to real-time. This model can be used in conjunction with a more sophisticated shape-from-motion algorithm that relies on the detailed analysis of optic flow [33]. In this context, it serves both as an initial model for shape refinement and as a nonlinear (bounding volume) constraint on the final shape.

The automatic acquisition of 3D object models is important in many applications. These include robotics manipulation, where the object must first be described or "learned" before it can be recognized or manipulated; computer aided design (CAD), where automatic model building can be used as input to the CAD system; and computer graphics animation, where it can facilitate the task of the animator by giving him easy access to a large catalog of real-world objects. All of these applications become much more interesting if the acquisition can be performed quickly and without the need for special equipment or environments.

The problem of building object models from multiple views has a long history in computer vision [2, 12, 36]. Octrees were first introduced as an efficient representation for geometric models by Jackins and Tanimoto [17] and Meagher [22] (see also [Carlbom *et al.*, [6]). A good survey of octree representations can be found in [29], and a survey of construction and manipulation techniques in [7].

The construction of volumetric description from multiple views was first described by Martin and Aggarwal [20] (the representation used was a collection of "volume segments"). Chien and Aggarwal [9] constructed an octree from three orthographic projections (see also [14, 18, 25] for newer 3-view reconstruction algorithms). This approach was extended to 13 standard orthographic views by Veenstra and Ahuja [35]. Hong and Shneier [16] and Potmesil [27] both used artibrary views and perspective projection. In both of these papers, an octree of the conic volume formed by the silhouette and the center of projection is computed for each viewpoint, and the octrees from all of the viewpoints are then intersected. Both approaches project the octree cubes into the image plane to perform the object/silhouette intersection. In contrast to this, Noborio *et al.* [24] and Srivastava and Ahuga [32] perform the intersections in 3-space, which eliminates the need for perspective projection. Both of these approaches use a polygonal approximation to the silhouette (a recent paper by Srivasan *et al.* [31] uses a variant of the polyhedral representation). Additional research has also been done on building octrees from range data [10] and the recognition of object models from octrees [8].

The new approach described in this paper was designed with two goals in mind: to process each image as it arrives (on-line) and to produce a coarse model quickly and refine it as more images are seen (incremental). A secondary goal of this research was to find highly parallelizable algorithms. Our aproach is similar to Potmesil's [27], but instead of building a separate octree for each view, we intersect each new silhouette with the existing model.

1049-9660/93 $5.00

More importantly, we do not build a full-resolution octree after each view. Instead, we build a coarse octree description first, and then refine it as the object continues to rotate in front of the camera (we can, if desired, store the initial image sequence and reuse it). This makes our approach similar to other recent incremental algorithms [21]. As we will see, this approach significantly reduces both the individual computation per image and the total computation performed, since less cubes are generated and tested.

Our approach also uses a very efficient 2D image plane intersection test based on bounding squares and the chessboard distance transform [28] (Potmesil uses a quadtree or non-overlapping rectangle description). The 3D projection operations, computation of the distance transform, and bounding box intersection test are easily parallelizable and can thus take advantage of modern massively parallel architectures [19]. The 3D perspective projection is also well suited to graphics accelerators [11] or fast floating-point microprocessors [15].

We begin in Section 2 with a brief review of the octree representation. In Section 3 we describe how our algorithm hierarchically constructs the octree. In Section 4 we describe our fast image plane intersection test. In Section 5 we show some results of our algorithm operating on synthetically generated images. We analyze the complexity of our algorithm in Section 6. In Section 7 we switch gears and discuss a number of practical image processing issues: silhouette computation (adaptive thresholding), the automatic determination of turntable rotation, and the calibration of the camera position. This is followed by some results obtained from real images (Section 8). In Section 9 we discuss a number of extensions, including moving the camera and repositioning the object on the turntable. We close in Section 10 with a comparison between our new algorithm and other approaches.

2. OCTREE MODELS OF SHAPE

An octree is a tree-structured representation that can be used to describe a set of binary valued volumetric data enclosed by a bounding cube. The octree is constructed by recursively subdividing each cube into eight subcubes, starting at the root node (a single large cube). Each cube in an octree can be one of three *colors*. A *black* node indicates that the cube is totally occupied (all of its data = 1), and a *white* node indicates that it is totally empty (all of its data = 0). Both black cubes and white cubes are leaf nodes in the tree. A *gray* cube lies on the boundary of the object and is only partially filled. It is an interior node of the tree and has eight equally sized children of different colors. Figure 1a shows a graphical view of a small octree, and Fig. 1b shows its associated colored tree representation.

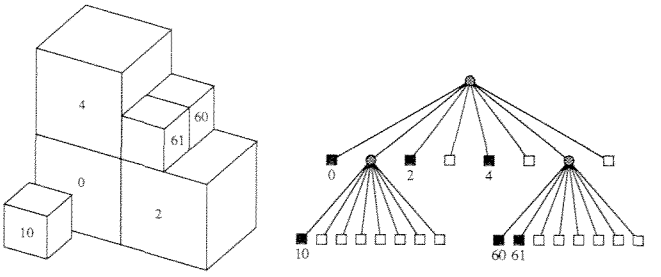

FIG. 1. A simple two-level octree and its tree representation

Octrees are an efficient representation for most volumetric objects since there is a large degree of coherence between adjacent *voxels* (volume elements) in a typical object. A number of schemes have been developed to efficiently represent and encode octrees [7]. For our application, we use a very simple representation, where all of the cubes at a given resolution level are kept in the same array. Each cube explicitly represents its x, y, and z coordinates as well as its color and a pointer to the first of its children (for a gray cube). This representation is well suited to the commonly occurring operations in our algorithm. In particular, it enables the fast projection into the image of all the cubes at a given resolution level.

3. HIERARCHICAL OCTREE CONSTRUCTION FROM MULTIPLE VIEWS

Our basic algorithm constructs the octree in a hierarchical coarse-to-fine fashion. At a given octree resolution, the inner loop of the algorithm incrementally constructs the 3D volume by testing projected octree cubes against a sequence of silhouettes. After one complete revolution of the object, those cubes whose occupancy is uncertain are subdivided. Because the octree is completely constructed at one resolution before refining the next one, we create less cubes that would eventually be trimmed off.

In more detail, we start each new revolution of the object on the turntable with a collection of black cubes, all at the given (finest) resolution level. To initialize the algorithm, we start with a small collection (e.g., $8 \times 8 \times 8$) of black cubes. Cubes that are black are believed to lie completely within the object, white cubes are known to lie outside of the object, and gray cubes are ambiguous (this is different from the usual octree coloring scheme, where gray cubes are known to have children of differing colors). For each new image that is acquired, we project all of the current cubes into the image and test whether they lie totally within or outside of the silhouette (Section 4). We then update the color of each cube according to the following table:

Old color ⇒ / Test result ⇓	Black	Gray	White
Inside	black	gray	white
Ambiguous	gray	gray	white
Outside	white	white	white

Note that cube colors can only change from black to gray or white, or from gray to white (i.e., the volume is "carved away").

For our hierarchical approach to work, the tests of projected cubes against the silhouettes can be crude, but they must not classify a cube as wholly inside or outside unless this is certain. Projected cubes that are erroneously classified as "ambiguous" will be colored gray (unless they are already white), and the decision as to their true color will be deferred to the next finer resolution level. This allows our intersection tests to be more sloppy and also makes our algorithm more robust against noise.

After one resolution level has been processed completely, i.e., the object has been seen from all of its views (which usually corresponds to a complete revolution), we refine the gray cubes by splitting them into eight pieces. These new cubes are first colored black and are then processed identically to the cubes at the previous resolution level. Note that since all of the possible views are used before proceeding to the next level, there is no need to update nodes that belong to levels coarser than the one currently being analyzed. However, testing the cubes (parents) at coarser levels first can greatly reduce the total number of cubes tested, since small cubes which fall well within the silhouette are never tested (see Section 6). After the final octree model has been computed, if we wish to save storage space, we can go back and compact the octree by converting gray nodes into black or white nodes if all of their children are the same color.

4. IMAGE SILHOUETTE INTERSECTION TESTS

To make the above approach practical, we must have an efficient way to project the octree cubes into image-space coordinates and to test for intersection with the object silhouette. The perspective projection step is not a critical computational bottleneck, since it is trivially parallelizable, and specialized hardware exists to perform it [11]. In our current implementation, we simply project all eight corners of each cube using floating point arithmetic.

An opaque cube projected into the image plane will in general form a six-sided polygon. Performing an accurate test of this hexagon against the silhouette (which we represent as a binary image) can be quite time consuming. Instead, we use a coarser test based on the hexagon's bounding box, which may sometimes fail to detect a true inclusion or exclusion. This is acceptable in our overall framework, since this kind of "mistake" simply colors a cube gray, thus deferring the decision to the next finer resolution level.

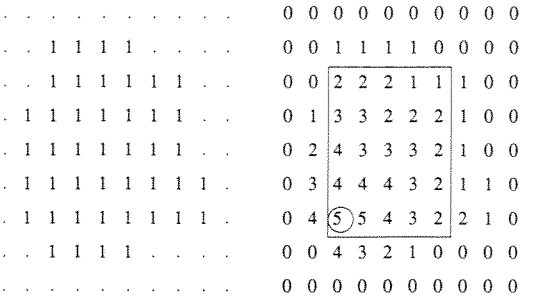

FIG. 2. The half-distance transform and its use in inclusion testing. A sample binary image is shown on the left, and its half-distance transform is on the right. The circled 5 in the distance map indicates that a 5×5 square is the largest square inside the silhouette whose lower left corner is at that pixel.

In more detail, each projected cube is converted into a *bounding square*. This allows us to do the inclusion and exclusion tests using a single lookup into two distance maps constructed from the silhouette and its complement. These distance maps are one-sided versions of the chessboard distance transform [28]. Each point in the distance map contains the size of the largest square starting at that pixel that fits completely within the silhouette (Fig. 2). These *half-distance transforms* for the silhouette and its complement can be computed using a single raster-scan pass over the image, or using an $O(\log d)$ step parallel algorithm, where d is the diameter of the largest region in the silhouette (Appendix A).

One more detail which must be resolved is how to compute the bounding square from the bounding box of the projected cube (which is defined by the minimum and maximum projected x and y values). We could center the bounding square over the bounding box, or push it flush left or right (top or bottom) against the bounding box. We have chosen the latter approach, testing both bounding squares for possible inclusion or exclusion, since this will more often result in a successful test.

5. SYNTHETIC MODEL RESULTS

To determine the viability of our new algorithm and to quantify its performance, we decided to first test it on a series of synthetically generated images. Our test models were constructed by combining superquadric parts [26, 30]. The models were rendered from a variety of viewpoints using a 3D graphics modeling system. The rendered images, along with the simulated turntable angle and camera position, were then given to the octree construction algorithm. The complete simulation system was run online, which enabled the observation of the simulated input images and of the resulting octree models simultaneously.

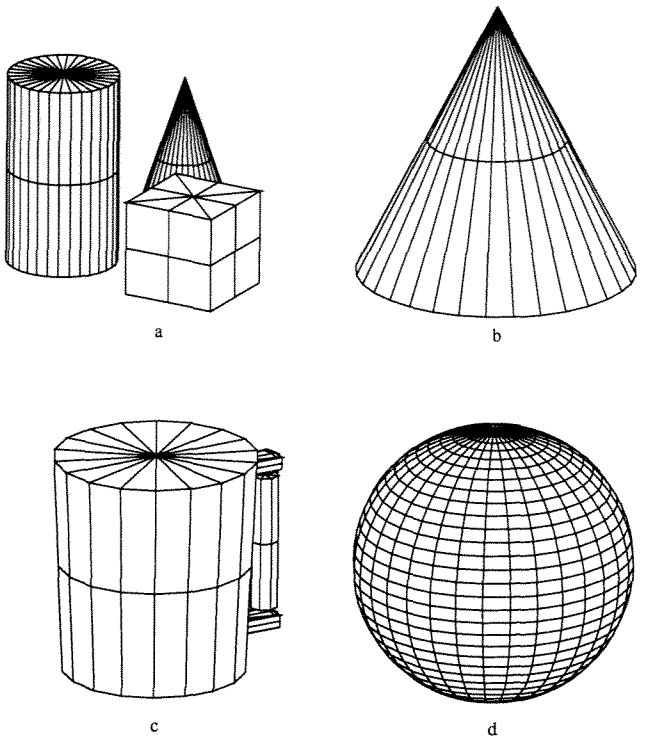

FIG. 3. Superquadric test models: (a) blocks, (b) cone, (c) cup, (d) sphere.

Figure 3 shows four of the superquadric models that we used. Figure 4 shows the corresponding octree models constructed from 32 evenly spaced views (11° increments). For these simulations, the camera was at 4.0 (times the cube size) from the object center, the inclination (from horizontal) was 20°, and the field of view was 30°. The octrees shown are rendered at 32^3 resolution, even though they were computed to 64^3 resolution. As we can see from these examples, the shape of these objects is recovered fairly well, although there are occasional "bulges" on some flats sides of the objects (due to a limited number of viewpoints) or at the top (unavoidable from this camera position).

To obtain a more quantitative measure of the algorithm's performance, we counted the number of cubes of each color created at each resolution level. As we can see from Table 1, at the finer levels, roughly 25% of the cubes are black, 25% are white, and 50% are gray. Since each gray cube becomes split into eight children, the number of cubes roughly quadruples at each level. This is consistent with Meagher's [22] finding that the number of cubes is proportional to the object surface area (measured in units of the finest resolution).

We can verify if this is true by computing the surface area of each of the test objects (Table 2). Note that the flat areas at the bottom of the objects do not contribute significantly to the octree cube count, but they do add significantly to the surface area. To verify the "accuracy" of our octree construction, we can compare the total volume of the octree with that of the original synthetic model

TABLE 1
Cube Counts for Octrees Computed from Synthetic Models (32 Views)

Model	Number of cubes (%black/%gray)				
	Level 2	Level 3	Level 4	Level 5	Level 6
Blocks	64 (0/61)	312 (7/64)	1600 (13/56)	7152 (15/46)	26520 (18/48)
Cone	64 (0/75)	384 (16/50)	1536 (16/50)	6176 (21/52)	25696 (23/50)
Cup	64 (0/97)	496 (16/45)	1792 (19/56)	8040 (20/54)	34560 (22/50)
Sphere	64 (13/88)	448 (16/61)	2176 (21/57)	9888 (24/52)	41248 (25/51)

TABLE 2
Total Cube Counts, Surface Area, and Volumes for Octrees

Model	Cube count	Analytic area	Octree volume	Analytic volume
Blocks	35648	2.3844	0.1685	0.1552
Cone	33856	2.5321	0.2643	0.2601
Cup	44952	2.9734	0.3445	0.3102
Sphere	53824	3.1278	0.5267	0.5190

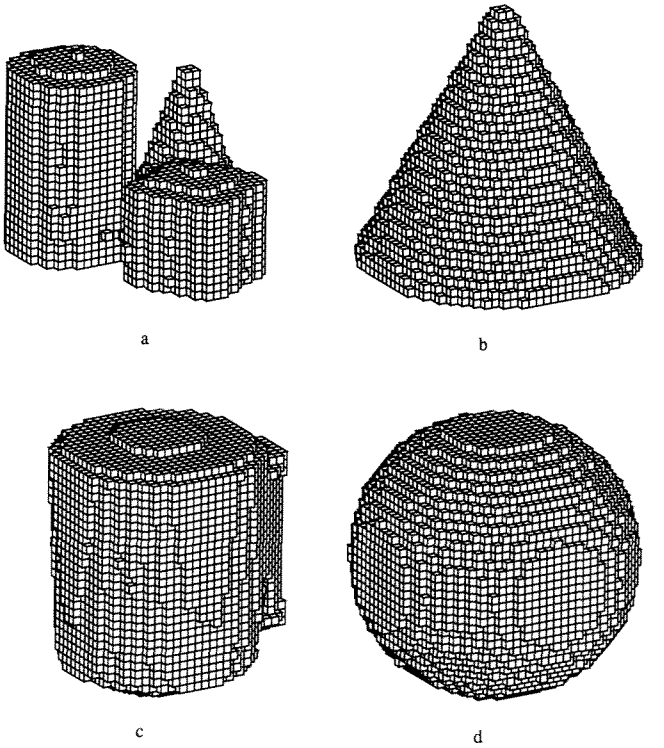

FIG. 4. Derived octree models: (a) blocks, (b) cone, (c) cup, (d) sphere.

TABLE 3
Cube Counts for Octrees Computed from Synthetic Models (Single View)

Model	Number of cubes (%black/%gray)				
	Level 2	Level 3	Level 4	Level 5	Level 6
Blocks	64 (0/83)	424 (21/63)	2120 (23/55)	9376 (25/52)	39232 (26/51)
Cone	64 (11/78)	400 (25/56)	1800 (25/53)	7568 (26/51)	30672 (26/50)
Cup	64 (22/78)	400 (33/53)	1680 (26/53)	7088 (26/50)	28376 (26/50)
Sphere	64 (28/72)	368 (35/56)	1656 (29/53)	6984 (29/49)	27376 (26/50)

(Table 2), as was done in [35, 27, 32]. For most of the volumes, these figures compare favorably, despite the restricted range of viewpoints.

6. COMPLEXITY ANALYSIS

The hierarchical octree construction algorithm developed in this paper is designed to operate in close to real-time. It should therefore perform less computation for each image acquired than previous approaches such as Potmesil's, and hopefully perform less computation in total. In practice, this is what occurs, because instead of building a complete octree from each image, only those cubes already on the surface of the model are refined.

As we saw previously, the expected number of cubes for an octree is proportional to its surface area (for a sufficiently smooth object). For most objects, this area should be smaller than the surface area of the viewing cone formed by the camera and the object silhouette. To see if this occurs in practice, we can count the number of cubes in the octrees formed from a single silhouette (Table 3), which tells us the number of cubes in a viewing cone. Compared to the octree cube count from our hierarchical octree algorithm (Table 1), we see that the single-view count generally exceeds the 32-view count. The exception is for large simple objects that fill most of the octree volume.

This comparison only gives us a rough idea of the computation performed by both the hierarchical approach and the previous single resolution methods. To compare the complexity of our new algorithm with previous algorithms, we implemented Potmesil's algorithm, which constructs a full-resolution octree and trims it against successive silhouettes. We also added code to the octree construction algorithms to count the number of cubes projected and tested against the silhouette. Table 4 shows the number of cubes tested at each level for each of the four synthetic models using four different algorithms:

I. Our hierarchical construction algorithm, testing only cubes at the finest level.

II. The hierarchical construction algorithm, testing all cubes in top-down traversal to the current finest level. The children of cubes that fall within the current silhouette are not tested.

III. The hierarchical construction algorithm, testing only cubes at the finest level whose parent's inclusion tests were "ambiguous" in the current silhouette. This

TABLE 4
Number of Cubes Tested for Octrees Computed from Synthetic Models

Model	Alg.	Number of cubes tested				
		Level 2	Level 3	Level 4	Level 5	Level 6
Blocks	I	1453	8091	40993	170717	671433
Blocks	II	1741	9239	33902	98748	259544
Blocks	III	1453	7639	25510	68209	173822
Blocks	IV	1741	9864	39731	133689	445235
Cone	I	1602	8750	36470	159232	662893
Cone	II	1890	8748	30020	91606	263118
Cone	III	1602	6924	21908	63766	178197
Cone	IV	1890	9055	34030	116592	388390
Cup	I	2014	10785	46108	207520	894627
Cup	II	2302	10011	34056	111262	336377
Cup	III	2014	7739	25008	79260	233503
Cup	IV	2336	9871	37070	127227	420651
Sphere	I	2048	11793	58141	263372	1098931
Sphere	II	2336	11628	42709	137718	417634
Sphere	III	2048	9292	31749	97370	289049
Sphere	IV	2336	11628	45210	156800	531054

technique is applicable when the same set of silhouettes is being recycled at each resolution level, e.g., when working from a stored image sequence. The visibility of each cube in each silhouette is passed as a bit vector to a cube's children.

IV. A traditional octree construction algorithm [27], where a complete (fine resolution) octree is constructed from each silhouette.

As we can see from Table 4, the hierarchical octree construction algorithms with top-down traversal (Algorithms II and III) test fewer cubes than the traditional single-resolution algorithm (Algorithm IV). Some care must be taken in interpreting these figures. For Algorithms I–III, the figures in each column show the amount of processing performed at each level. The total amount of computation is obtained by summing all columns from the coarsest level up to the finest level desired. For Algorithm IV, the total amount of computation performed is the number shown in the table, since the algorithm executes with only a single finest resolution level. Even when the total number of cubes tested is computed, Algorithm IV still performs about 50% more tests than Algorithm III.

From these figures, we can conclude that the hierarchical approach has two advantages over traditional single-level octree construction algorithms. First, with a small amount of computation, we can rapidly obtain a crude shape estimate. Second, the total amount of computation is reduced, because only cubes already on the surface and close to the silhouette edge are tested at each iteration.

7. IMAGE PREPROCESSING

Before we can apply our shape from silhouette algorithm to real image sequences, we must first address a number of low-level processing and calibration issues. These include the calibration of the camera position, the determination of turntable rotation, and the silhouette computation (object/background segmentation).

7.1. *Camera Parameter Calibration*

Before the object acquisition system can be run, we must determine the camera parameters, i.e., the camera position, orientation, and focal length. We do this using a known reference pattern (a hexagon) placed on the turntable (Fig. 5a) [5, 37]. First, we extract the edges by finding the zero crossings in the band-pass filtered image and using the gradient of the filtered image to compute the edge orientations and to throw away weak edges (Fig. 5b). Next, we use the Hough transform with gradient directions [3] to find the straight lines in the image (Fig. 5c). We group these lines into triplets of approximately parallel and evenly spaced lines (these lie on nearly straight lines in Hough space) and find the best three

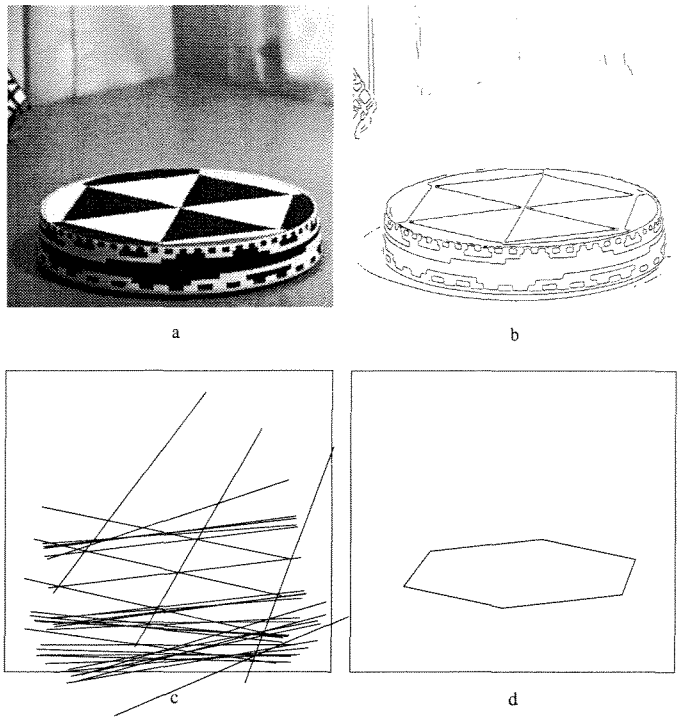

FIG. 5. Camera parameter computation: (a) hexagonal calibration pattern, (b) edges extracted, (c) lines fitted, (d) hexagon found.

triplets which define a visible hexagon in the image (Fig. 5d). The center point and six vertices of the hexagon are then used to obtain a non-linear least-squares estimate for the unknown camera parameters (inclination, twist, turntable center, and focal length).

7.2. *Orientation Estimation*

To estimate the current orientation (rotation angle) of the turntable, we look for the 8-bit binary pattern affixed to the side of the turntable. The sequence of binary patterns follows a Gray code, where only one bit changes at a time. This prevents an inconsistent code from being read when several bits change asynchronously (either due to camera noise or if the binary code is not perfectly vertical in the image). To find this strip, we take horizontal projections of gray levels across a small number of columns (currently 32) centered around the middle of the image and look for large variations between the minimum and maximum gray levels. Because the top and bottom bits vary quickly (we permuted the bits in the Gray code, as shown in Fig. 6), we are guaranteed to find at least one black to white or white to black transition in these bits, and they can be used to frame the complete 8-bit pattern.

Once we have determined the vertical extent of the Gray-code pattern, we find in each column the 8-bit binary code corresponding to the pixels in this strip (for now, by simply choosing eight equally spaced samples within

FIG. 6. Input image of object (cup), turntable, and position coding ring.

the strip) and convert it into a value between zero and 255. The binary values from all of the columns are averaged to obtain a position estimate that has better than 8-bit resolution (about 0.1° accuracy). This number, normalized to a 0° to 360° range, is the output of our turntable position estimator.

7.3. *Adaptation and Thresholding*

To compute the silhouette, we first adapt the imaging system to an empty background scene by acquiring a few images and computing the minimum, maximum, mean, and variance values at each pixel. With our current setup, we can perform this computation in near real-time (0.8 frames/second). These measurements are then used to precompute a range of normal values for each background pixel. When thresholding the image, any pixel falling outside of this range is assumed to be part of the object.

To obtain better silhouettes, we perform one or two steps of local shrinking operations [28] to remove isolated misclassified pixels. We have also found it useful to lower the threshold in the bright parts of the image corresponding to the turntable top, since these usually have cast shadows from the objects which might be misclassified as object rather than background. The result of our thresholding algorithm applied to the input image (Fig. 6) is shown in (Fig. 7). Further improvements could be made to the thresholding algorithm by using color images, since the probability of misclassification is reduced because of the larger three-dimensional color space, and since highlights and specularities (going from dark to light) are not forced to cross the background color. An alternative solution, which requires a slight modification of the image

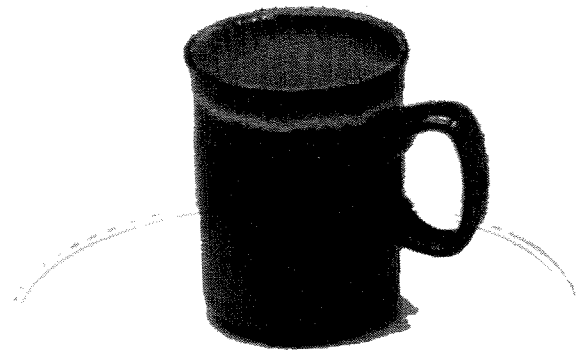

FIG. 7. Silhouette image: background = white; object = black/gray.

acquisition environment, is to use backlighting or to use a background of unusual color.

The overall sequence of steps involved in precalibrating the system is thus fairly simple. First, we place the hexagonal calibration disc on the turntable top and compute the camera parameters. Next, we find the location of the Gray-code strip by looking at projected intensity distributions in the central columns of the images. The setup is now fully calibrated and will report back the turntable angle for each image. We then adapt to an empty turntable and start thresholding images. The result is a system that can be operated by a naïve user such as a CAD designer or graphics artist.

8. REAL IMAGE RESULTS

We have tested our algorithm on a number of real image sequences. Figure 6 shows a sample image of a cup sitting on the turntable. Applying the adaptation and thresholding techniques described previously, we obtain the silhouette image shown in Fig. 7. Figure 8 shows the final model constructed from the cup image sequence. The final reconstruction captures the overall shape of the cup quite well. The major source of errors in the reconstruction is due to the specularities that cause erroneous gray nodes in the middle of the model (these are not visible in Fig. 8, which renders gray nodes at the lowest level as opaque).

Figure 9 shows another three-dimensional model (a small microphone stand) which was reconstructed from a sequence of 24 silhouettes. The overall shape of the model is reconstructed fairly well, including the cavity for the microphone which is visible in some of the images.

9. EXTENSIONS

There are a number of efficiency optimizations that could be added to our algorithm. To reduce the cost of the cube corner projections, we could:

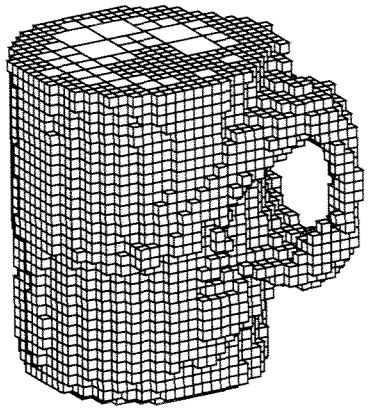

FIG. 8. Octree model derived from real cup images.

1. cache the projections, since each corner is usually used more than once;
2. project only one corner and use fixed vectors to define the other seven corners;
3. use recursive subdivision of the projected octree cubes to obtain the image coordinates of the points at the desired resolution.

Note that these last two approaches do not generate the exact corner positions, but this is perfectly acceptable, so long as a larger bounding box results.

Adding gray level or color value to the cubes in the octree (*texture mapping*) would enhance the realism of the reconstruction. However, this is not a trivial task, since many different intensities could map to a single cube, both within a single frame and between frames [27]. We could also use this stage to check for inconsistencies, such as white cubes that become visible later on.

Our octree construction algorithm would be more powerful if we could change the position of the camera and/or the object. The former case is easier to handle. We simply recalibrate the system and continue processing with the new camera parameters (we could also use more than one camera simultaneously). An interesting empirical question is whether we should still use the same hierarchical strategy as before (coarse to fine refinement), or whether we should process just the finest level—or maybe even the whole tree—for each new silhouette. The pure hierarchical algorithm described in this paper could then be replaced with a more flexible control strategy, which would decide how many whole or part levels to process, depending on the real-time constraints imposed by the rate of incoming images.

The change in object orientation caused by repositioning it on the turntable is also simple to handle if the object motion is known. In this case, we can transform the octree to its new orientation [1, 38] and continue processing as before. An example of such a known transformation would be if the object had only a few stable poses. Determining the object motion from the volumetric data itself is more difficult [8], especially when both the old and new object models contain areas that are extraneous. This remains an interesting area of future research.

Another interesting area of research is the interaction between the bounding volume generated by this algorithm and the deformable surface model computed from optic flow [33]. In this complementary approach to shape from rotation, knowledge of the optic flow and object motion provides a sparse and somewhat noisy estimate of points on the object's surface which are integrated using an elastic surface model. The bounding volume described by the octree can be used to provide inequality contraints on the shape of the deformable model.

Another approach to extracting three-dimensional shape from silhouettes is to track silhouette edges over three or more frames to estimate the location and curvature of points on the surface of the object [13, 34]. Such surface measurements can be used to further refine the volumetric model since they imply the existence of *free space* between the surface and the camera [4].

As a final processing step, it would be desirable to convert the volumetric description produced by this method to a smooth surface-based description [23] or directly to a parts-based model which uses more global primitives [26].

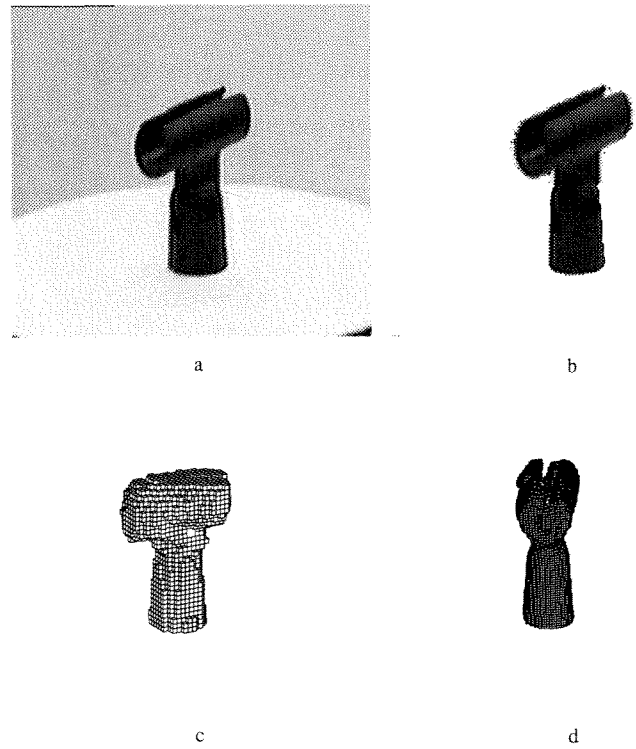

FIG. 9. Microphone stand image sequence: (a) input image, (b) silhouette image, (c) octree at resolution 5, (d) octree at resolution 6.

10. DISCUSSION

Given the current performance (speed) of our algorithm, how usable is it in practice? The microwave turntable that we use to rotate our object turns at about 0.5 revolutions per minute. The time to acquire, process, and display a 256 × 240 image on our workstation (DECstation 5000/200) is about 0.8 s per frame for adaptation and thresholding, and 1.25 s for an incremental octree construction step of moderate complexity. This means that we can process about 40 views per rotation, which is perfectly adequate for obtaining a bounding volume.

Ideally, we would like to speed up the turntable rotation (and the processing) by a factor of 10. This should be achievable with the next generation of faster RISC processors. Because our algorithm is essentially data parallel (both at the pixel and cube levels), it would be easy to obtain true real-time performance with a SIMD parallel machine.

A more important question is whether our system is useful and easy to use. Compared to previous octree construction algorithms, our new hierarchical approach requires less computation because it only examines those cubes close to the surface of the object. It provides a coarse model of the octree rapidly, which improves as the object continues to rotate in front of the camera. Compared to the alternatives of manually entering a model using a 3D pointer or using structured light, our system is easier to use and requires less equipment. However, it still has the same limitation as previous silhouette-based volumetric construction algorithms, such as a limited precision and the inability to detect concavities in the object. To be truly useful, our method will have to be combined with more sophisticated shape from motion algorithms that track surface markings on the object. As part of such a shape from rotation system, it can quickly and automatically provide 3D shape descriptions of real object for geometric modeling applications.

APPENDIX A: SERIAL AND PARALLEL HALF-DISTANCE TRANSFORMS

Our octree construction algorithm uses the half-distance transform to test the bounding boxes of projected cubes against the silhouette images. The half-distance transform is a one-sided version of the chessboard distance transform [28]. Given a binary image, it computes for each pixel the size of the largest square containing all ones starting at that pixel.

The serial version of the algorithm computes

$$f^{(1)}(i,j) = \min(f^{(0)}(i,j), 1 + \min_{(u,v)\in N_3} f^{(1)}(i+u, j+v))$$

in reverse raster order, where $f^{(0)}(i,j)$ is the input binary image, and $N_3 = \{(0,1), (1,0), (1,1)\}$. A simple parallel version of the algorithm can be drived from the PD algorithm described in [28, p. 356],

$$f^{(m+1)}(i,j) = f^{(0)}(i,j) + \min_{(u,v)\in N_4} f^{(m)}(i+u, j+v),$$

where $N_4 = \{(0,0), (0,1), (1,0), (1,1)\}$. This algorithm takes d steps to converge, where d is the diameter of the largest region (the maximum value in the distance map). A variant on this algorithm is

$$f^{(m+1)}(i,j) = \min(f^{(m)}(i,j), 1 + \min_{(u,v)\in N_3} f^{(m)}(i+u, j+v)),$$

where initially $f^{(0)}(i,j)$ are set to zero or infinity for pixels outside/inside the silhouette.

A quicker version of the preceding parallel algorithm uses two log d steps

$$\begin{aligned} f^{(2m+1)}(i,j) &= \min(f^{(2m)}(i,j), s \\ &\quad + \min_{(u,v)\in N_2} f^{(2m)}(i+su, j+sv)) \\ f^{(2m+2)}(i,j) &= \min(f^{(2m+1)}(i,j), s \\ &\quad + f^{(2m+1)}(i+s, j+s)), \end{aligned}$$

where $s = 2^m$ and $N_2 = \{(0,1), (1,0)\}$. This algorithm doubles the distance to the neighbors every two iterations.

If the number of parallel processors available is less than the number of pixels—for example, when using a MIMD machine or a smaller SIMD array—it pays to reduce the total amount of computation performed. In this case, we can use a hierarchical version of the parallel half-distance algorithm. We use the pyramidal algorithm,

$$\begin{aligned} f^{(2m+1)}(s_x i, s_y j) &= \min(f^{(2m)}(s_x i, s_y j), s \\ &\quad + \min_{(u,v)\in N_2} f^{(2m)}(s_x i + su, s_y j + sv)) \\ f^{(2m+2)}(s_x i, s_y j) &= \min(f^{(2m+1)}(s_x i, s_y j), s \\ &\quad + f^{(2m+1)}(s_x i + s, s_y j + s)), \end{aligned}$$

in two fine-to-coarse-to-fine sweeps. In the first sweep, $s_y = 1$, and $s_x = s = 2^m$ going up the pyramid and $s_x = s = 2^{(2M-m)}$ going down, where $M = \log_2 d$ is the height of the pyramid. In the second sweep, the roles of s_x and s_y are reversed. This algorithm first propagates the distances horizontally and diagonally, subsampling the points by two horizontally. The algorithm then repeats the procedure subsampling vertically. Because the number of points being updated is reduced by two every two steps, the total number of operations is $O(n)$.

REFERENCES

1. N. Ahuja and C. Nash, Octree representation of moving objects, *Comput. Vision Graphics Image Process.* **26,** 1984, 207–216.
2. H. Baker, Three-dimensional modelling, in *Fifth International Joint*

Conference on Artificial Intelligence (IJCAI-77), Cambridge MA, August 1977, pp. 649–655.

3. D. H. Ballard and C. M. Brown, *Computer Vision*, Prentice–Hall, Englewood Cliffs, NJ, 1982.
4. R. C. Bolles, H. H. Baker, and D. H. Marimont, Epipolar-plane image analysis: An approach to determining structure from motion, *Int. J. Comput. Vision* **1,** 1987, 7–55.
5. B. Caprile and V. Torre, Using vanishing points for camera calibration, *Int. J. Comput. Vision* **4**(2), 1990, 127–139.
6. I. Carlbom, I. Charkravarty, and D. Vanderschel, A hierarchical data structure for representing the spatial decomposition of 3-D objects, *IEEE Comput, Graphics Appl.* **5**(4), 1985, 24–31.
7. H. H. Chen and T. S. Huang, A survey of construction and manipulation of octrees, *Comput. Vision Graphics Image Process.* **43,** 1988, 409–431.
8. C. H. Chien and J. K. Aggarwal, Identification of 3D objects from multiple silhouettes using quadtrees/octrees, *Comput. Vision Graphics Image Process.* **36,** 1986, 256–273.
9. C. H. Chien and J. K. Aggarwal, Volume/surface octrees for the representation of three-dimensional objects, *Comput. Vision Graphics Image Process.* **36,** 1986, 100–113.
10. C. H. Chien, Y. B. Sim, and J. K. Aggarwal, Generation of volume/surface octree from range data, in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'88), Ann Arbor, MI, June 1988.* pp. 254–260, IEEE Computer Soc. Press, New York, 1988.
11. J. H. Clark, The geometry engine: A VLSI geometry system from graphics, *Comput. Graphics (SIGGRAPH' 87)* **16**(3), 1982, 65–72.
12. F. P. Ferrie and M. D. Levine, Intergrating information from multiple views, in *IEEE Computer Society Workshop on Computer Vision, Miami Beach, FL, Dec. 1987*, pp. 117–122, IEEE Comput. Soc. Press, New York, 1988.
13. P. Giblin and R. Weiss, Reconstruction of surfaces from profiles, in *First International Conference on Computer Vision (ICCV' 87), London England, June 1987*, pp. 136–144, IEEE Comput. Soc. Press, New York, 1987.
14. U. G. Gujar and I. V. Nagendra, Construction of 3D solid objects from orthographic views, *Computers & Graphics* **13**(4), 1989, 505–521.
15. J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, Los Altos, CA, 1990.
16. T.-H. Hong and M. O. Shneier, Describing a robot's workspace using a sequence of views from a moving camera, *IEEE Trans. Pattern Analy. Mach. Intell.* **PAMI-7**(6), 1985, 721–726.
17. C. L. Jackins and S. L. Tanimoto, Oct-trees and their use in representing three-dimensional objects, *Comput. Graphics Image Process.* **14,** 1980, 249–270.
18. Lavakusha, A. K. Pujari, and P. G. Reddy, Linear octrees by volume intersection, *Comput. Vision Graphics Image Process.* **45,** 1989, 371–379.
19. J. Little, G. E. Blelloch, and T. A. Cass, Algorithmic techniques for computer vision on a fine-grained parallel machine, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-11**(3), 1989, 244–257.
20. W. N. Martin and J. K. Aggarwal, Volumetric description of objects from multiple views, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-5**(2), 1983, 150–158.
21. L. H. Matthies, T. Kanade, and R. Szeliski, Kalman filter-based algorithms for estimating depth from image sequences, *Int. J. Comput. Vision* **3,** 1989, 209–236.
22. D. Meagher, Geometric modeling using octree encoding, *Comput. Graphics Image Process.* **19,** 1982, 129–147.
23. O. Monga, N. Ayache, and P. T. Sander, From voxel to curvature, in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'91), Maui, HI, June 1991*, pp. 644–649, IEEE Computer Soc. Press, New York, 1991.
24. H. Noborio, S. Fukada, and S. Arimoto, Construction of the octree approximating three-dimensional objects by using multiple views, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-10**(6), 1988, 769–782.
25. A. G. Pai, H. Usha, and A. K. Pujari, Linear octree of a 3D object from 2D silhouettes using segment tree, *Pattern Recognit. Lett.* **11,** 1990, 619–623.
26. A. P. Pentland, Perceptual organization and the representation of natural form, *Artif. Intell.* **28**(3), 1986, 293–331.
27. 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.
28. A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, Academic Press, New York, 1976.
29. H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison–Wesley, Reading, MA 1989.
30. F. Solina and R. Bajcsy, Recovery of parametric models from range images: The case for superquadrics with global deformations, *IEEE Trans. Pattern Anal. Mach. Intell.* **12**(2), 1990, 131–147.
31. P. Srivasan, P. Liang, and S. Hackwood, Computational geometric methods in volumetric intersections for 3D reconstruction, *Pattern Recognit.* **23**(8), 1990, 843–857.
32. S. K. Srivastava and N. Ahuja, Octree generation from object silhouettes in perspective views, *Comput. Vision Graphics Image Process.* **49,** 1990, 68–84.
33. R. Szeliski, Shape from rotation, in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'91), Maui, HI, June 1991*, pp. 625–630, IEEE Computer Soc. Press, New York, 1991.
34. R. Vaillant, Using occluding contours for 3D object modeling, in *First European Conference on Computer Vision (ECCV'90), Antibes, France, April 1990*, pp. 454–464, Springer-Verlag, New York/Berlin, 1990.
35. J. Veenstra and N. Ahuja, Efficient octree generation from silhouettes, in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVRP'86), Miami Beach, Florida, June 1986*, pp. 537–542, IEEE Comput. Soc. Press, New York, 1986.
36. B. C. Vemuri and J. K. Aggarwal, Representation and recognition of objects from dense range maps, *IEEE Trans. Circuits Systems* **CAS-34**(11), 1987.
37. L. L. Wang and W. H. Tsai, Camera calibration by vanishing lines for 3-D computer vision, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-13**(4), 1991, 370–376.
38. J. Weng and N. Ahuja, Octree of objects in arbitrary motion: Representation and efficiency, *Comput. Vision Graphics Image Process.* **39,** 1987, 167–185.