ROTATION AND TRANSLATION OF OBJECTS REPRESENTED BY OCTREES

Tsai-Hong Hong and Michael O. Shneier[†]

Sensory-Interactive Robotics Group
National Bureau of Standards
Gaithersburg, MD 20899

ABSTRACT

This paper describes an algorithm for performing arbitrary translations and rotations of objects represented by octrees. Given an octree in a standard position and a transformation, the algorithm builds a new tree in a top down fashion, visiting each node in the new tree only once, and constructing only those nodes that appear in the final tree. It works by projecting the transformed space over the original tree, and labeling the new nodes according to the labels of the nodes in the underlying untransformed tree.

1. Introduction

Octrees are finding applications in a number of areas of computer graphics and robot vision because of the ease with which certain operations can be performed (for example, rendering and hidden surface removal in graphics [3, 5, 8], and collision avoidance and path planning in robotics[2, 6, 10]). The advantages of the octree arise because it is a true volumetric representation, and yet can be stored in a compact form.

An octree [7, 8] is a recursive decomposition of a *cubic space into subcubes. Initially, the whole space is* represented by a single node in the tree, called the root node. If the cubic volume is homogeneous (full or empty), then the root is not decomposed at all, and comprises the complete description of the space. Otherwise, it is split into eight equal subcubes (octants), which become the children of the root. This process continues until all the nodes are homogeneous, or until some resolution limit is reached. Nodes corresponding to cubic regions that are completely full are called object (or full) leaf nodes. Nodes corresponding to empty regions are called background (or empty) leaf nodes, and nodes corresponding to mixed regions (non-leaf nodes) are called non-terminal (or gray) nodes.

The octree has advantages over other volumetric representations because of the speed with which volume elements can be located and their relative positions established. Octrees, however, suffer from two major disadvantages. First, they do not provide an exact representation for objects, and, second, the size of the tree representing an object depends critically on the position of the object in the space. Thus, translating or rotating an object can dramatically increase or decrease the size of the tree. This paper provides an algorithm for computing such a new octree resulting from an arbitrary translation and rotation of an initial tree (or trees), which attempts to minimize the effects of these disadvantages.

For static objects, a compact representation can be constructed by choosing a favorable orientation and origin for the object when building the tree. It is advantageous to use such a representation in the algorithm, because the computation time depends on the size of the source tree. Because of the finite resolution of the octree representation, a transformation that moves and reorients an object is not guaranteed to preserve the shape of the object exactly. Successive transformations will, in general, deform the shape more and more. It is thus preferable to maintain a single, master tree to represent the object, and to perform all transformations on that tree. Successive transformations are then performed by keeping track of the current position of the object, and constructing a new transformation as the composition of this position and the desired motion. Applying the new transformation to the master tree each time minimizes the errors introduced into the new tree [2]).

Nevertheless, errors are introduced into the new tree, and must be controlled. The errors manifest themselves as fluctuations in the shape of the boundary of the object. These should be kept as small as possible, and their cumulative effect on the volume of the object should be bounded. The transformation must also maintain the topology of the object. That is, holes should not be introduced or filled in, and the object should not become disconnected. Meeting these constraints requires significant computation. To

[†] Current address: Philips Laboratories, 345 Scarborough Road, Briarcliff Manor, NY 10510.

achieve reasonable performance, the algorithm first applies a set of fast tests to weed out simple cases, and then applies a complete screening to those nodes that remain.

The next section describes the algorithm. It is followed by a discussion, examples of the implementation, and conclusions.

2. The Algorithm

The goal of the algorithm is to take an original, master octree and an arbitrary translation and rotation (expressed as a homogeneous matrix in our implementation) and to produce a new, transformed, octree that represents the result of applying the transformation to the master octree.

The master octree is oriented parallel to the coordinate axes. This implies that all the delimiting surfaces of cubes and subcubes are described by simple equations. The transformed tree may have arbitrarily oriented cubes with respect to the master tree. These cubes must, however, be mapped back into a tree with cubes parallel to the coordinate axes. In this respect, the problem is similar to that of geometric correction of images [9].

The algorithm relies on the observation that transforming an object by some transformation M produces the same representation as transforming the underlying coordinate system by M^{-1}. It performs this inverse transformation on a new root cube, expressing the transformed octree in the same coordinate system as the master tree. The new root cube is superimposed on the root of the master octree, and the new tree is constructed in top-down order, building new nodes by testing the new volumes against the volumes with which they overlap in the old, master octree. The algorithm constructs only those nodes that are needed for the final representation, and no coalescing of intermediate nodes is required. At the highest resolution in the tree (lowest level), special processing is carried out for partially filled cubes.

For every node, the algorithm attempts to set two flags, one for an intersection with the background, and one for intersection with an object. If, at any time, both flags are set, the node can immediately be split, without looking at any further nodes (except at the lowest level, when no splitting is possible). This constraint is used to restrict the number of nodes examined at each stage. Once one of the flags has been set, no further nodes of that type will be examined. A second constraint restricts the nodes to be examined to those whose volumes in the master tree overlap the new node, that is, only neighboring nodes or parents can affect the value assigned to a flag. Nodes that meet these constraints are called *qualifying* nodes or cubes.

In the following algorithm, the untransformed octree is referred to as the master octree. Nodes in the tree are referred to as cubes when their volumes or surfaces are being discussed. Given a translation and rotation defined by a homogeneous matrix, and a master octree to be transformed, the process works as follows.

1. Construct a new root for the transformed tree by applying the inverse of the given transformation to the set of planes defining the root cube of the master octree. The inverse is guaranteed to exist because the transformation is a rigid body motion. There are six planes to be transformed,

$$a_1x+b_1y+c_1z+d_1=0$$

$$\vdots$$

$$a_6x+b_6y+c_6z+d_6=0$$

(where, in the master tree, all but one of the a_i, b_i, c_i are 0 in each equation). Each plane is transformed to give the new sides of the transformed cube. Note that the children of this cube have surface equations that are simply computed as known offsets from these planes (i.e., only the values of $d_1 \ldots d_6$ change.

2. We want to establish that the transformed cube intersects only with empty nodes, in which case it is empty, or that it intersects only with full nodes, in which case it is full. Otherwise, the cube must be subdivided. We start by examining the vertices of the transformed cube. If any vertex is outside the root node of the master octree, then the transformed tree is said to intersect with the background by definition (we consider the region surrounding the octree to be empty - if this is not appropriate for a particular application, only minor changes to the algorithm will result). If we can establish that the cube also intersects with an object node, we can immediately subdivide it, and recursively examine its children in the same way. If at least one vertex is inside the master octree, and one is outside, the transformed tree is immediately subdivided. The tests described below are performed on all children (and on the root of the transformed tree if it is not yet subdivided). For each child, the tests always begin with the root of the master octree, and work down the appropriate subtrees.

3. For each qualifying cube at a given resolution level in the master tree (initially the root cube), find the centers of the master cube and the transformed cube (in the master coordinate frame). Compute the radii of the inscribed spheres for each cube, and of their enclosing spheres. Find the smaller of the cubes, and see if its center is inside the larger cube. If it is, they intersect (Figure 1).

4. If the distance between the centers of the cubes is less than the sum of the radii of the inscribed spheres, then the cubes intersect (Figure 2). If the distance is greater than the sum of the radii

of the enclosing spheres, the cubes do not intersect (Figure 3). Otherwise, the intersection region lies in the shell between the two spheres (Figure 4), and a more detailed examination must be made to determine whether or not the cubes intersect. This is done as follows.

Intersect each edge in the master cube with each surface in the transformed cube. The intersection tests are easy because each edge is parallel to one of the three coordinate axes. Any intersection must lie both within the area bounded by the edges of the surface and within the endpoints of the edge not including the endpoints themselves (Figure 5). The endpoints are excluded because, for an intersection, the edges are required to pierce the surface rather than simply touch it. The same intersection tests must be done using the edges from the transformed cube and the surfaces from the master cube to account for cases such as that in Figure 6. In this case, as opposed to the previous one, the surface equations are simple, while the edge equations are not.

5. If a cube intersects with an empty node, a flag for empty intersection is set. If it intersects with a full node, this is also flagged. If both flags are set, the cube can immediately be split, and its children recursively tested, because the node cannot be a leaf node. A node that totally contains a gray node is also split. If only one flag is set at any resolution level, after examining all qualifying nodes at that level, the new node is a leaf node of the appropriate type.
6. At the lowest level in the tree (highest resolution) nodes that are not homogeneous require special treatment. We compute an estimate of the amount of the total volume of the node that is occupied by the object. This can be thresholded, or kept for later processing. The estimate is computed by artificially subdividing the nodes into 8 regions. The center of each resulting cube is transformed and, if it falls within a full node in the master tree, the subcube is said to be full. A count of the number of full subcubes is used as an estimate of the occupied volume of the terminal cubes. In the examples presented below, this value was thresholded to decide whether or not the node should be displayed.

3. Discussion

For each node in the new tree, the algorithm traverses a subtree of the master tree to the level of the new node. As soon as the node is found to intersect with both a background and an object node, the node is split, a new gray node is created, and the algorithm is repeated on the children. Only in the case that the node is a leaf is a complete traversal of the subtree required to the level in the master tree at which the new node overlays only nodes of one color. In this case, no children are generated. We note that child nodes need only be intersected with nodes in the master tree starting at the level of their parents, because the decision to split the parent implies that the necessary information for labelling its children lies below it in the tree. Thus, for each new node in the transformed tree, a narrow cone is examined, starting at the parent level, and ending at the level at which the label of the node is determined (possibly the highest resolution level in the master tree). The breadth of the cone is limited to the set of nodes in the master tree that intersect the new node's volume (in practice, the parent node, its neighbors, and their descendents).

Weng and Ahuja [11] describe an algorithm to perform arbitrary rotations and translations on octrees that works in a manner different from the one we describe here. Their approach is to project the object nodes in the master tree into the new tree, and to construct the necessary new nodes by traversing the new tree. While their approach has the same computational complexity as ours (described below), it suffers from some disadvantages. These arise because of the local nature of projecting the individual object nodes, as opposed to the global projection of the new tree over the old. Each object node projected into the new tree interacts only with the partially constructed new tree. As a result, it is possible for subtrees to be generated that will later be merged into leaf nodes when adjacent object nodes are projected. These subtrees may have to be traversed many times before finally being deleted. In contrast, our method never creates extra nodes, and traverses the master tree, which is usually compact, and contains all the necessary information for informed decisions.

The local nature of Weng and Ahuja's approach also affects the treatment of partially filled nodes at the highest resolution level. Their approach is to declare a node to be an object leaf if its centroid is inside or on the boundary of a projected object node. If two adjacent projected nodes that do not meet this condition would nevertheless together fill the new node, this cannot be taken into account. Our method computes the label for the node based on the percentage of the node that is full, from whatever source. This has a small practical benefit due to potential inaccuracies arising from arithmetic roundoff errors in the projection computations, which might perturb the boundary of a node enough to place the centroid outside two projecting nodes, which together cover the new node.

Other previous work has dealt with special cases of the transformation problem. Several authors have treated the case of pure translation and rotations by multiples of 90 degrees [1, 4, 7, 8]. The reason for these restrictions has been the desire to maintain an exact representation. Arbitrary transformations suffer from a disadvantage, in that applying their inverse does not necessarily result in the original octree. If a sequence of transformations is to be applied, for example, in moving an object through a region of space, something must be done to prevent the shape of the object from changing radically as the transformations are applied. Our solution, and that of Boaz and Roach [2], and Weng and Ahuja [11], has been to maintain a cumulative transformation from a standard position, and always perform the resultant transformation on a standard representation of the object. This reduces the errors to those incurred in constructing the cumulative transformation, which gives rise to an error in position, rather than one in shape. Because of floating point arithmetic, this has, in general, a much smaller effect on the results than successive transformations would have if applied directly to successive octrees, which are defined in a discrete space.

Dealing with the limited resolution is the major difficulty with the octree representation. For some applications, such as robot path planning, it is desirable to declare all highest resolution leaf nodes to be full if they intersect with any part of an object. This is less costly than the approach taken by Boaz and Roach [2], in which a shell of nodes around each object node is projected into the octree, and intersections with these are considered to be filled. Care must be taken in other applications, such as rendering of fine line features, that lines are not lost or broken.

The complexity analysis of the algorithm can be sketched by analogy with that of Weng and Ahuja as follows. Let n be the side length of the master root cube. Then the depth of the tree is $O(\log n)$. Let K be the number of nodes in the new tree. In generating the new tree, we might at worst have to traverse the master tree to its full depth for each new node created, requiring $O(4^{\log n})$ operations, (see [11]). Under the assumption that the number of black nodes at a level in the tree is proportional to the total number of nodes possible at that level, it follows by analogy with the algorithm of Weng and Ahuja [11] that the average time complexity of the algorithm is bounded by $O(K \log n)$.

4. Implementation

The algorithm was implemented in the "C" programming language on a VAX-11/780 computer. Master octrees were built for a number of simple objects. No effort was made to construct the most compact possible master trees, but the largest surfaces of the objects were aligned with the coordinate axes of the space (except in the case of the cylinder, where this is not possible). Figures 7, 8, and 9 show the results of several experiments. In each image, three views of an object are shown, projected into the same octree. The transformations used are shown in Figure 10. Note that the final octree represents the projection all three instances of the object, and is not a composite image constructed from three trees for display purposes. The times given in Table I are for the total construction of the complete tree (all three objects). They are in CPU seconds on a loaded VAX. It is clear that in some cases, many more nodes are required to represent the objects in their new positions than in the master tree, and that most of the new nodes appear at the boundaries of the objects. For all the objects, nodes at the highest resolution are displayed as object nodes if they were half or more filled after transformation.

5. Conclusions

An algorithm has been described for performing arbitrary rotations and translations on objects represented as octrees. The algorithm constructs only those nodes that appear in the final tree, and makes decisions about splitting nodes at the highest possible level. By traversing the master tree instead of the transformed tree, advantage can be taken of compact coding of that tree. The algorithm attempts to minimize errors in the transformation by always working from the master tree, and maintains the topology of objects to the limit of its resolution.

References

1. N. Ahuja and C. Nash, Octree representation of moving objects. *Computer Vision, Graphics, and Image Processing 26*, 1984, 207-216.

2. M. Boaz and J. Roach, An oct-tree representation for three-dimensional motion and collision detection. Virginia Polytechnic Institute and State University, Technical Report, June 1984.

3. L. J. Doctor and J. G. Torborg, Display techniques for octree-encoded objects. *IEEE Computer Graphics and Applications 1* 3, July 1981. 29-38.

4. C. R. Dyer, A quadtree translation algorithm. Computer Science Technical Report, University of Wisconsin, Madison, August, 1984.

5. A. S. Glassner, Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications 4* 10, October 1984, 15-22.

6. M. Herman, Fast, three-dimensional collision-free motion planning. Robot Systems Division, National Bureau of Standards, 1985.

7. C. L. Jackins and S. L. Tanimoto, Oct-trees and their use in representing three-dimensional objects. *Computer Graphics and Image Processing 14*, 1980, 249-270.

8. D. Meagher, Geometric modeling using octree encoding. *Computer Graphics and Image Processing 19,* 1982, 129-147.

9. A. R. Rosenfeld and A. C. Kak, *Digital Picture Processing,* Second Edition, Volume 2, Academic Press, New York, 1982.

10. R. Ruff and N. Ahuja, Path planning in a three dimensional environment. Proc. 7th ICPR, Montreal, Canada, July 1984, 188-191.

11. J. Weng and N. Ahuja, Octree representation of objects in arbitrary motion. Proc. CVPR, San Francisco, CA, 1985, 524-529.

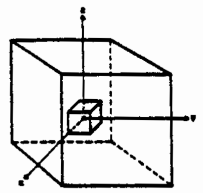

Figure 1. If the center of the smaller cube is within the larger cube, the cubes intersect.

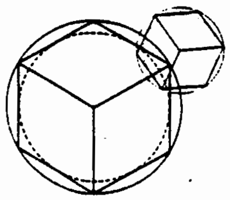

Figure 2. When the distance between the centers of the cubes is less than the sum of the radii of the inscribed spheres, the cubes intersect.

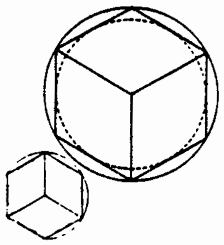

Figure 3. When the distance between the centers of the cubes is greater than the sum of the radii of the enclosing spheres, the cubes do not intersect.

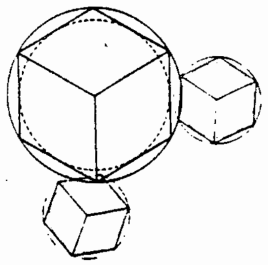

Figure 4. When the intersection region lies in the shell between the two spheres, more detailed examination is required before the cubes can be said to intersect.

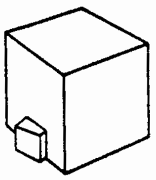

Figure 5. When an edge of a node in the master tree (small node) intersects a surface in the transformed tree, the transformed node must be split.

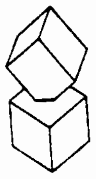

Figure 6. Even if no edge in the node from the master tree intersects a surface in the transformed tree, the nodes can intersect.

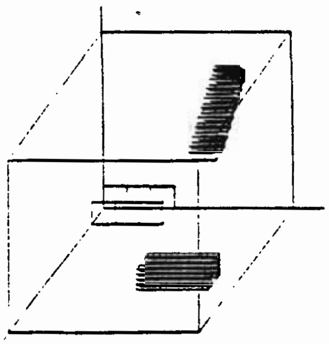

Figure 7. Three views of a rectangular parallelepiped (box) projected into the same octree. The transformations are $T_1, T_2,$ and T_3 from Figure 10. For the display, the tree was expanded to level 6.

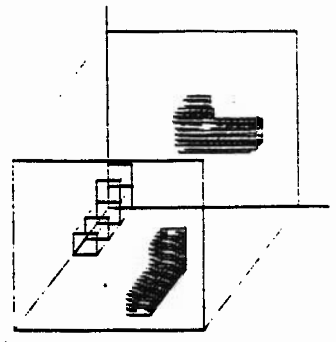

Figure 8. Three views of an L-shape projected into the same octree. The transformations are $T_1, T_4,$ and T_5 from Figure 10. For the display, the tree was expanded to level 6.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 270 \\ 0 & 1 & 0 & 256 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0.7071 & 0.7071 & 0 \\ -1 & 0 & 0 & 384 \\ 0 & -0.7071 & 0.7071 & 384 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 256 \\ 0 & 1 & 0 & 270 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -0.9848 & -0.1736 & 128 \\ 1 & 0 & 0 & 256 \\ 0 & 0.1736 & 0.9848 & 256 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 256 \\ -1 & 0 & 0 & 384 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

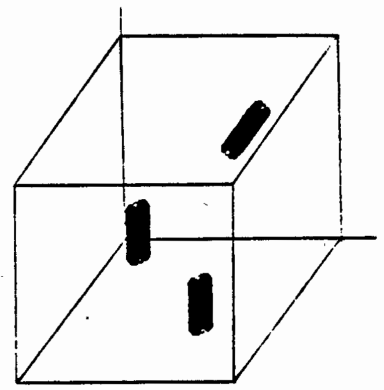

Figure 9. Three views of a cylinder projected into the same octree. The transformations are $T_1, T_4,$ and T_6 from Figure 10. For the display, the tree was expanded to level 7.

Figure 10. The homogeneous transformations used for the examples of Figure 7, 8, and 9. From top to bottom, they are $T_1, T_2, T_3, T_4, T_5, T_6$.

Object (Transformation)	Nodes in Master Tree	Nodes in Transformed Tree	Time For Transformation	Time Per Node
box (T_1&T_2&T_3)	25	545	6.25	0.011
L-shape (T_1&T_4&T_5)	25	713	8.05	0.011
cylinder (T_1&T_4&T_6)	969	2233	127.4	0.057

Table I.
Examples of the times required for transforming various objects.