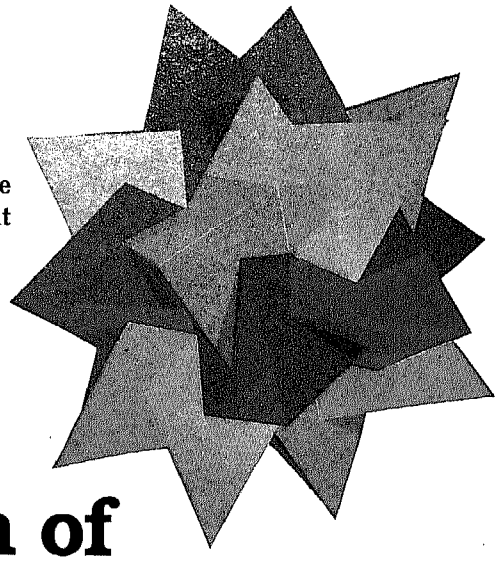


Poly1. A stellation of the icosahedron. It is also five interlocking tetrahedra. The model is lighted by eight light sources.

Automatic Creation of Object Hierarchies for Ray Tracing

Jeffrey Goldsmith
Jet Propulsion Laboratory

John Salmon
California Institute of Technology

Intersection calculations dominate the run time of canonical ray tracers. A common algorithm to reduce the number of intersection tests required is the intersection of rays with a tree of extents, rather than the whole database of objects. A shortcoming of this method is that these trees are difficult to generate. Additionally, manually generated trees can be poor, greatly reducing the run-time improvement available. We present methods for evaluation of these trees in approximate number of intersection calculations required and for automatic generation of good trees. These methods run in $O(n \log n)$ expected time where n is the number of objects in the scene. We report some examples of speedups.

Due to the increased demand for visual realism in computer generated images, ray tracing has become a very popular rendering algorithm. Ray tracing programs simulate the interaction of light with the environment, thus simply determining such optical effects as reflection, refraction, and shadowing. The basic algorithm[1,2] traces rays from an eyepoint through a simulated screen at a set of objects to be seen.

Since ray tracing is a brute force algorithm, taking very little advantage of global information about the picture to be rendered, it is very costly in computer time. Whitted[3] discovered that 75 percent of the time required for simple scenes was taken in the calculations that determined which objects were hit by each ray. The original algorithm required intersecting each ray with each object in the scene. Since then the algorithm has been improved so that each ray need not be intersected with each object.[4-8] Each method involves intersection calculations with simple bounding volumes to determine if a more complex intersection calculation can be avoided. Warren, Glassner, Fujimoto, and Kaplan[4,5,7,8] differ from Rubin and Whitted[6] in that they trade space (computer memory) for time in attempts to speed up the algorithm. In a limited space machine, such as a hypercube parallel processor,[9] these methods are currently not practical to use, though in the future that will likely not be so, as memory space on these machines increases. The method of hierarchical extents[6] uses a tree search to find the objects that are hit by a ray. In the best case it yields $O(\log n)$ intersection calculations per ray. Since tree nodes are small in comparison to object data, only about 30 percent extra space is required to store the hier-

0272-1716/87/0500-0014$01.00 © 1987 IEEE

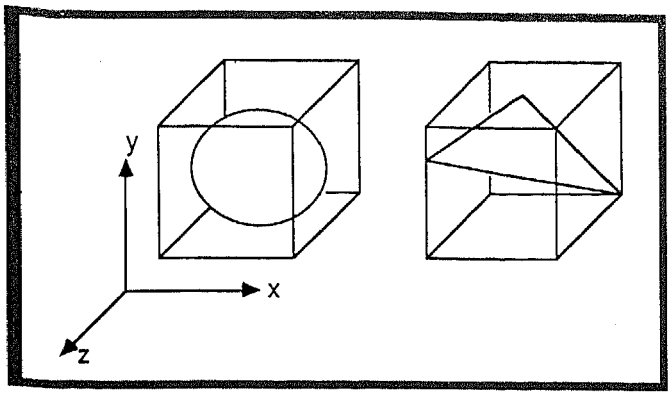


Figure 1. Some objects surrounded by their bounding boxes.

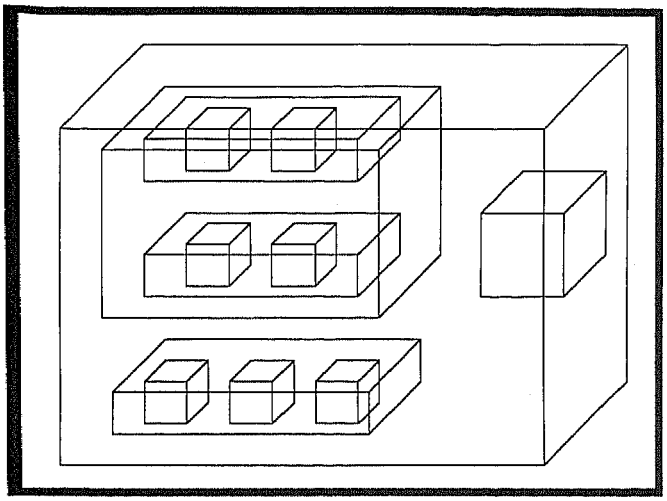
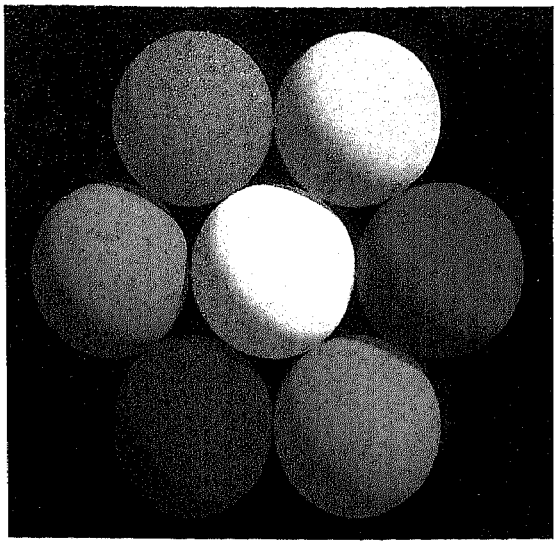


Figure 2.The bounding boxes represented by the structure of the tree you will see in Figure 4. Note that the boxes fit much more tightly than they are drawn here.



Seven spheres. Seven diffusely reflecting spheres.

archy required for machine use in a typical implementation.

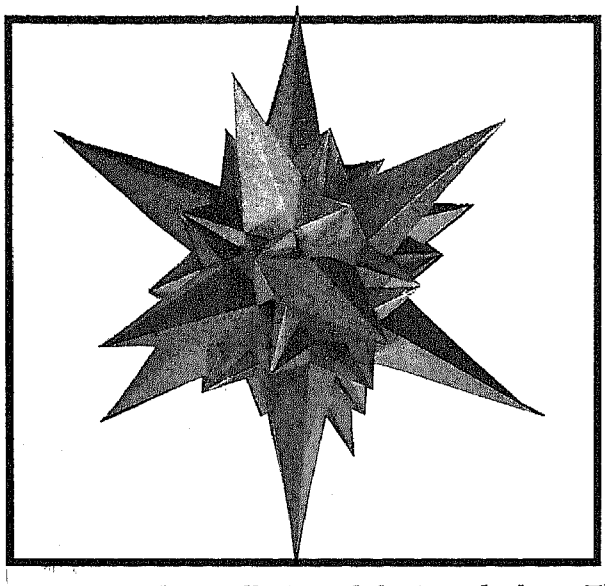
To use the method of hierarchical extents, a tree of bounding volumes must be constructed. Bounding volumes are simple geometric objects that fit around the objects that make up the model. Bounding volumes are chosen to be objects that are simple to intersect with a ray, such as spheres or rectangular prisms that have sides parallel to the coordinate planes. See Figure 1 for some examples. These bounding volumes are combined into a tree by picking some of them and surrounding them with another bounding volume. This process is repeated recursively until a bounding volume is generated that surrounds the whole scene. See Figure 2 for an example of a hierarchy of bounding volumes.

Many different trees can be built for a given scene, and they require differing computation times to render the image. The time required to render a simple image can easily vary by a factor of fifty due simply to the choice of different trees. Thus, it is important to find a way to choose a tree that reduces that time. Normally, the tree is built from the structure used to model the scene. Sometimes, trees are built manually for simple or combined models. A tree so generated is generally not a particularly good one, because building these trees manually is very difficult. This is due to the large amount of data that must be manipulated all at once, as well as the difficulty of performing complex tasks in three dimensions.

Trees that a modeler generates are also usually not good because they were not generated to speed up the ray-tracing operation, but instead to simplify modeling of the scene. Trees built for modeling are generally too simple, and often have large branching ratios, whereas ray-tracer trees tend to be better if they have small branching ratios to generate as many tree-pruning operations as possible. In fact, for a checkerboard, a binary tree is the optimal configuration for the tree; yet few would choose to model it using two repeated polygons and a very deep hierarchy.

Since it is possible to use many different trees to render a scene, and since manual construction of trees is tedious and not as effective as desired, computer programs have been written to build these trees automatically. The simplest such algorithm constructs a complete *n*-ary tree, filling the leaf nodes with objects in some simple order. Not surprisingly, this method yields poor results, since it takes no model information into account.

Another method, the median cut algorithm, divides the scene into halves along some spatial axis and surrounds each half with a bounding volume. It then repeats this procedure recursively on each half. This method works better than the previous one, but it still does not adequately account for the intended use of the tree dur-



Poly2. Another stellation of the icosahedron. This, too, is illuminated by eight lights. Both Poly1 and Poly2 were computed by Roy Williams on a hypercube.

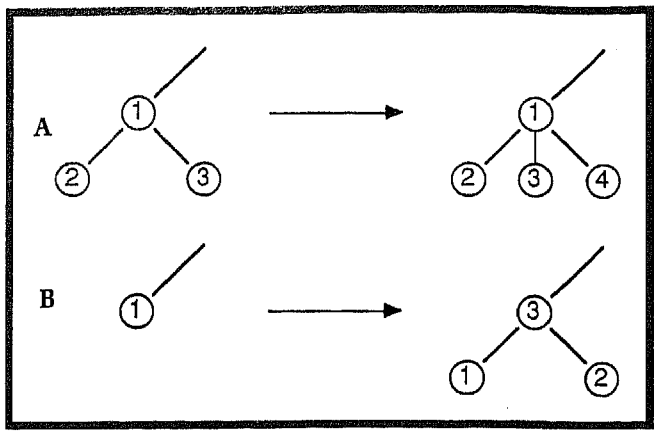


Figure 3. In (a) is the insertion of node 4 as child of node 1. In (b) is insertion of node 2 at position of leaf node 1 to create new node 3.

ing the rendering process. In the next section we describe how to build good trees based on a metric described in detail in the subsequent section.

Automatic generation of trees

A useful tree generation algorithm must be applicable to scenes with hundreds of thousands of objects; thus, any algorithm that runs in $O(n^2)$ time or worse is likely to be too slow. This means that when considering placement of a node, one must not use information about all of the other nodes, just some small portion of them. This constraint will not permit the optimal tree to be found. For most models, however, many trees exist, with rendering costs that are only slightly higher than the optimal one. This is because the number of possible trees that exist is exponentially proportional to the number of objects, and local changes in the structure of the tree tend to have small effects on the overall cost. Thus, generating a suboptimal tree is of little consequence, as long as it is much better than trees generated by other methods, since the choice of trees only affects the amount of computation needed to render the image, and small differences of time are not critical.

The general strategy used to construct the tree is a heuristic tree search. Objects are added successively and the tree is searched to find a suitable insertion point for each new node. Since not all nodes in the tree can be considered as a point for insertion, the search must follow only a few paths. The choice of subtrees to search from a given node is determined by the smallest increase in surface area of the node's bounding volume that would occur if the new node were to be inserted as a child of it. (See the next section for a justification of this heuristic.) During the search process, two or more children of a node may have the same increase in bounding volume surface area after adding the new node. This occurs most frequently when the children are large and the increase is zero, usually near the root of the tree. In that case, all equivalently costly subtrees can be searched to find the best location for insertion, or the tie can be broken by closeness of the object to the center of the old bounding volume, or even random selection. In practice, only the first two or three levels of the tree have nodes with big enough bounding volumes for this to happen, and then only near the end of the construction. Thus, all those paths usually can be searched without undue cost. Also, since the tree setup usually takes only a small fraction of the computer time that rendering the image takes, many paths can usually be searched without significant extra time.

At each level of the tree during the search, the new node is considered a prospective child of each node that will be searched (see Figure 3a). The tree is evaluated with the proposed insertion and the location with the smallest increase in tree cost is saved. When the search reaches leaf nodes, the new node and the leaf node are proposed as siblings of a new nonleaf node constructed in the position of the old leaf node (see Figure 3b). When the search is complete, the node is inserted in the tree wherever the increased cost of the tree will be minimized.

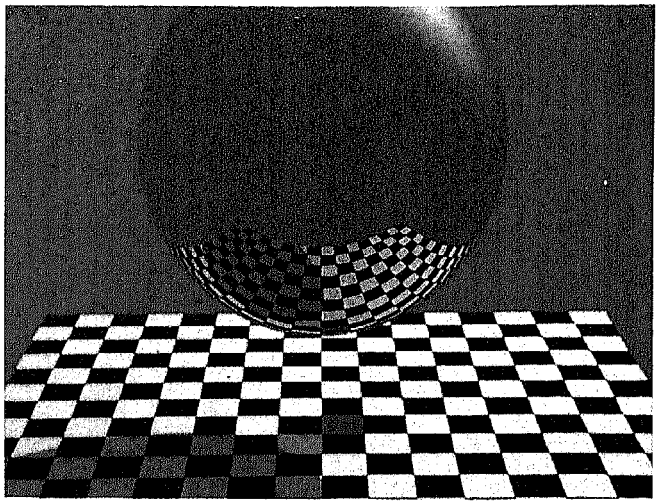
Thus, for each object to be inserted, an $O(\log n)$ tree search must be done. This yields a total asymptotic time complexity of $O(n\log n)$.

Since the tree is being constructed without complete knowledge of the model, the order in which objects are added is significant. Several data orders were tested. The obvious choice is the order in which the modeler supplies objects. This has some spatial coherence and some

Table 1. Expected number of intersections/rays of tree generated.

Model:	7spheres	Poly1	Poly2	Checker	Spheres	Antenna
No Hierarchy	8	21	61	258	379	1099
Manual	7.07	21	61	14.0	153	354
Model Order	5.94	19.9	12.9	10.1	32.0	63.2
Sorted Order	6.53	20.0	15.9	13.3	32.0	55.2
Ave. Shuffled	6.21	19.9	14.3	9.43	40.5	44.8
Best Shuffled	5.94	19.9	12.4	8.78	36.7	42.4
Worst Shuffled	6.32	19.9	17.4	10.8	48.2	47.4
Time (seconds)	.011	.067	.581	4.04	4.50	17.0

This is a summary of the approximate speedups available via different automatic hierarchy generation methods. The third through seventh row data are for the automatic tree construction algorithm described in this paper, and they are indicated by the data order used for the test. The values are in number of intersection calculations per ray, regardless of implementation. Expected improvement can be found as the ratio of before and after values. Note that the run time of Poly2 will not be significantly improved. This is due to all the polygons intersecting almost all the others. The last row lists the amount of VAX 750 computer time used to generate the hierarchies in rows three through seven.



Checker. 256 polygons and one sphere.

randomness. Sometimes, however, the modeler's output is in very bad order for the tree generation program. To try to correct for this, the data is shuffled once[10] before being used as input to the program. Since a random seed is used for the shuffle algorithm, the tree produced by the algorithm can be represented by that seed, just one additional number. With shuffled data, the resulting trees tend to be slightly worse than ones generated from data in model order, but for models without a symmetry represented in the original data order, by trying several seeds, a tree can be found that is better than the one generated from unshuffled data.

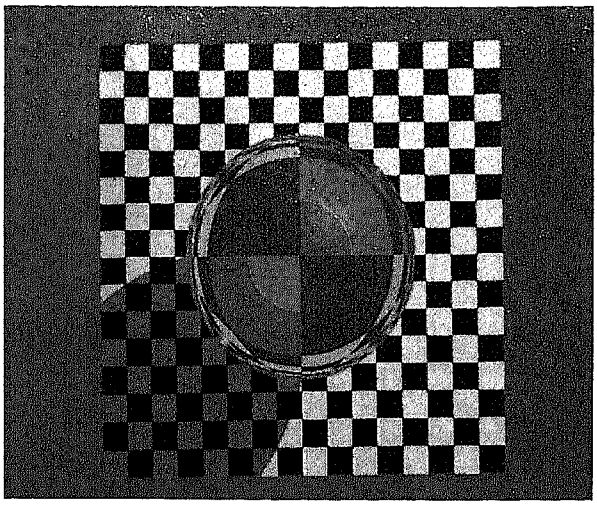
The effect of sorting the input data along a line was also investigated. It seems that sorting is the worst thing to do to the data, because the top levels of the thus generated tree do not represent the whole scene adequately. They are based on only a local section of the database and cannot be revised with this algorithm. Sorted data tends to yield results similar to worst case shuffled data.

The results of each of these programs are summarized in Table 1. Though no results are available from other methods for it, a model with 16,373 objects was run through the automatic tree generator. The resulting hierarchy predicted (and ran with) an average of 30.1 intersection tests per ray.

Evaluation of bounding box trees

In the tree construction algorithm we needed to estimate the additional cost of insertion of an object into the hierarchy. This required evaluating the cost of the whole tree, at least initially. To achieve the desired results— that is, a speedup of the rendering operation—it is necessary to use a cost function based on the intended use of the tree. Ray-tracing hierarchies are used to avoid intersecting rays with the objects in the hierarchy by finding that a ray fails to hit a simple bounding volume in the hierarchy, and thus determining that all objects below that node in the tree can be eliminated from further consideration. To do this, bounding volumes must be tested for intersection with the ray, incurring a cost in computer time. The quality of a tree is determined by the number of bounding volume intersections incurred during rendering, using the tree. Thus, we will evaluate a tree as the expected number of bounding volume intersections required to determine which objects to test for intersection during an average traversal (ray).

The simplest bounding volumes to intersect are the sphere and the rectangular prism with all sides parallel to coordinate planes. Each requires about ten floating-point operations on average to check for an intersection. The most effective bounding volumes are more complex,[11] and vary with the primitive objects they surround,[12] but the advantages of improving the structure of the hierarchy will be seen regardless of the types of bounding volume used, since the structure of the hierarchy does not affect the size of the bounding volumes around the primitive objects (leaves). The examples all



Checker. Same model as before, but with transparent sphere, illustrating spherical aberration and internal reflection.

use only orthogonal prisms, but mixed types would work as well by prorating each bounding volume by its cost to intersect.

The first step in estimating the number of intersection calculations that will be needed to intersect rays with a hierarchy is determining the probability with which an arbitrary ray will hit a given bounding volume. Since any ray that does not hit the root-level bounding volume will require exactly one intersection calculation, only rays that do hit it need be considered. Thus, the conditional probability that a ray hits a bounding volume if it hits the root bounding volume can be used instead.

For rays with an endpoint at a fixed distance from a bounding volume, the probability that an arbitrary one will hit the bounding volume is proportional to the solid angle subtended by the surface of the bounding volume. For convex objects, such as prisms and spheres, at large distances this is approximately proportional to the surface area of the object.[13] For an orthogonal prism of size l by m by n, the surface area is $2lm+2ln+2mn$; for a sphere of radius r the surface area is $4\pi r^2$. For more complex bounding volumes, approximations to their surface area can be used.

The relationship between the surface area of a bounding volume and the likelihood that an arbitrary ray emanating at some large distance will hit it is approximately linear. Since all bounding volumes are contained within the root node's bounding volume, the conditional probability of a ray hitting a given node if it hits the root node can be approximated as the area of the given node's bounding volume divided by the area of the root node's bounding volume. In general, this division need not be done during the generation of the tree, since only comparisons between conditional probabilities are needed in most cases.

To compute the expected number of intersection calculations of a whole tree, the conditional probabilities must be scaled correctly, but the division can be factored out and done only once. In fact, since bounding volume surface areas are only used in ratios, they need only be calculated to within a constant factor. If all the bounding volumes for a model are orthogonal prisms, their areas can be computed as $(l+m)n+lm$, which only costs two multiplications and two additions to compute. If only spherical bounding volumes are used, r^2 can be used as their area. (l, m,n, and r are as above.)

These conditional probabilities can be combined with the structure of the tree to estimate how many nodes of the tree will be hit by an arbitrary ray. Only rays that hit the root node are considered, since the structure of the tree is irrelevant to all other rays, and the structure of the tree has no effect on the size of the root bounding box. This assumes that the tightest fit volume of the given type is used for the set of objects it contains, which is natural for orthogonal prisms and space slices.[12] If a ray hits the root, then k more intersection calculations need to be done, where k is the number of children it has.

In fact, if any node is hit, the minimum number of additional intersection calculations that must be done is equal to the number of children it has. Since the cost of performing intersection calculations with the bounding volumes of the children of a node is incurred by a ray hitting that node, and the conditional probability of hitting it is as above, the total average cost of a node in units of intersection calculations is the product of the number of children it has and its surface area divided by the root node's surface area. Note that the root node's cost is its number of children, and the cost of all leaf nodes is zero; however, each leaf node's existence adds to its parent's cost. The estimated number of intersections required to intersect a ray with a tree is the sum of the costs of its nodes.

Figure 4 contains a simple example of a hierarchy. The numbers on the nodes are the areas of the bounding volumes. The expected number of intersection calculations required to intersect a ray with this tree is 7.3, broken down as follows. One intersection is required for the root node, which is assumed to be a hit. This causes three more intersections at level two. The leftmost node at level two has probability .6 (6/10) and two children, yielding a cost of 1.2 intersections. The second node at that level has probability .3 and three children, for .9 more calculations. The third node has no children and thus no additional cost. Two level-three nodes have children. They each have two children and probabilities of .2 and .4, giving costs of .4 and .8, respectively. Thus, the tree costs one intersection at level zero, three at level one, 2.1 at level two, and 1.2 at level three, for a total of 7.3.

Evaluating the cost of a tree is an $O(n)$ calculation. During the construction of the tree, different possible trees

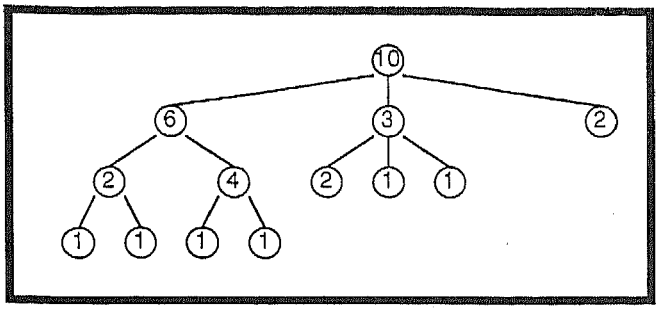
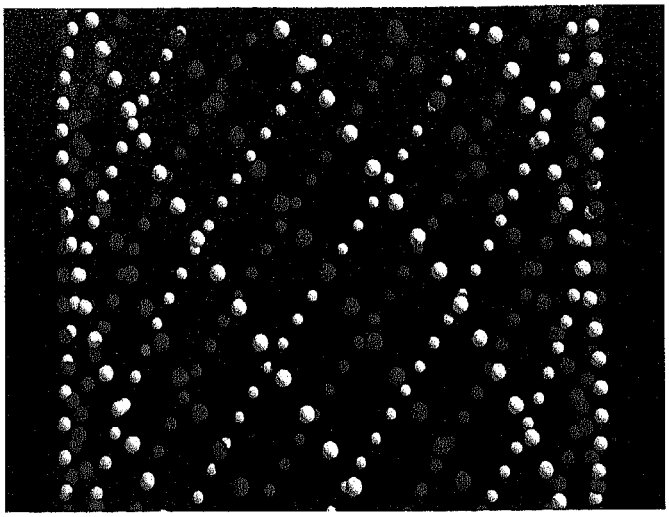


Figure 4. A tree with 7.3 expected intersections per ray. Numbers in the nodes are bounding volume areas.



Spheres. 378 spheres arranged as a twisted cylinder.

may need to be evaluated; an incremental cost for the addition of a node can be calculated in $O(\log n)$ time. This calculation can also be done while building the tree using the algorithm in the previous section, incurring no additional time complexity. If the node is being added as a child of another node (see Figure 3a), then the incremental cost of its parent is $(area_{new} - area_{old})k + area_{new}$ where k is the number of siblings the new node will have, and the areas are those of the parent's bounding volume before and after proposed insertion. If a new node is being combined with a leaf node to cause a new parent node to be created (see Figure 3b), its incremental cost is $2area_{new}$. Also, the increased cost to grandparents and other ancestors must be included. It is just $(area_{new} - area_{old})k$ of these ancestor nodes, where k is the number of children they have. Note that this is just a term in the general node cost increase.

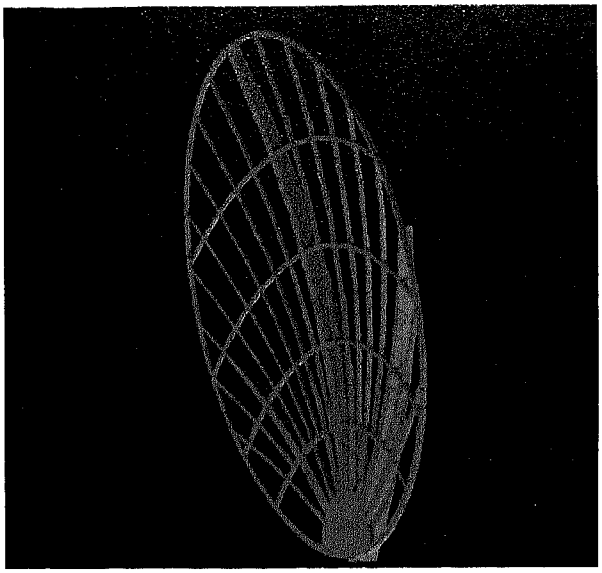
To test our estimate of a tree's cost, the number of bounding volume intersections done was measured for some simple pictures. Only rays shot from the eye (primary rays) were used for the test because secondary (shadow, reflection, and refraction) rays emanate from a surface and thus must intersect that object's (and all its ancestors') bounding volumes. This causes the minimum number of intersection tests needed for a secondary ray to be at least equal to the depth in the tree of the node that contains the object from which it emanates. Thus, the average number of intersection tests needed for a secondary ray should be somewhat higher than it would be for a truly arbitrary ray. Subtracting the average depth of the tree from the number of intersections found for secondary rays yields results about the same as for primary rays, only somewhat noisier. Also, only primary rays that hit the largest bounding volume were counted, since others are not affected by tree structure. Since primary rays tend to be similar to each other, three test runs were done, from three different directions.

Table 2 shows the results of these tests. Overall, the predicted results agree fairly well with the measured results. Data for all (including secondary) rays is also included for comparison. Note that the intersection count for secondary rays is about log(the number of objects) higher than it is for primary rays. (See, especially, data for Poly2.)

Table 2.
Intersections per ray for some model trees.

Test	Pixel Rays	Total Rays	P-ray Ints	Total Ints	Ints/ P-ray	Ints/ Ray	Expected Ints/Ray
Model: 7spheres							
1	96	152	516	859	5.38	5.65	5.94
2	96	156	656	986	6.83	6.32	5.94
3	196	316	956	1662	4.88	5.26	5.94
total	388	624	2128	3507	5.48	5.62	5.94
Model: Poly2							
1	224	560	3362	12516	15.0	22.4	15.0
2	196	644	3077	14484	15.7	22.5	15.0
3	224	632	2860	13506	12.7	21.4	15.0
total	644	1836	9299	40506	14.4	22.1	15.0
Model: Spheres							
1	3480	3948	150336	171343	43.2	43.4	42.6
2	3480	3938	205668	226829	59.1	57.6	42.6
3	3480	3936	95352	116506	27.4	29.6	42.6
total	10440	11822	451356	514678	43.2	43.5	42.6

The number of expected intersections per ray is approximately the same as the actual number of them for primary rays. The expected number is a little low for secondary rays. Poly2 has eight light sources, so its secondary ray count is very high.



Antenna. The high-gain antenna from Planet A, a Japanese spacecraft investigating Halley's comet.

Applications

Several other techniques have been found to speed up the method of hierarchical extents.[11,12] The gains from improving the tree are independent of the gains from these other methods; so the speedups obtained by using more than one of these methods will multiply.

Better trees are especially valuable in animation. The small preprocessing cost of building the trees is further reduced by being distributed over many frames worth of rendering time. If objects move, the whole tree does not have to be rebuilt. All the static objects in the scene can be combined into a tree, and the moving objects can be added before each frame. Objects that do not move very far can be time/space bounded and included in the static tree.

The ability to estimate tree costs is valuable to some parallel processing applications. The tree evaluator allows one to determine the size of the workload represented by a portion of the hierarchy. If the model database must be split up to fit in the local memory of a processor of a parallel machine, then the intersection calculation, which is the bulk of the ray-tracing computation[13] will be split up. It is important that this split is into roughly equal parts, or the processors that do the smaller amounts of calculation will not be used effectively.[14] ■

Acknowledgments

The pictures were computed on the 64-node Mark II Caltech Hypercube. Thanks to Roy Williams for the models for Poly1 and Poly2. Thanks to the JPL Computer Graphics Lab for the production of the slides. Also, thanks to the reviewers for pointing out some details that we missed.

This project was funded by the JPL Director's Discretionary Fund, Department of Energy grants DE-AS03-ER13118 and DE-FG03-85ER25009, the Parsons Foundation, and the Systems Development Foundation.

References

1. A. Appel, "Some Techniques for Machine Renderings of Solids," *AFIPS Conf. Proc.*, SJCC, AFIPS, Reston, Va., 1968 pp. 37-45.
2. J. Kajiya, Tutorial on Ray Tracing "State of the Art in Image Synthesis," (seminar notes) *Computer Graphics* (Proc. SIGGRAPH 83), July 1983.
3. T. Whitted, "An Improved Illumination Model for Shaded Display," *Comm. ACM*, 1980, pp. 343-349.
4. L.V. Warren, "Geometric Hashing for Processing Complex Scenes," CS Dept. Memorandum, Univ. of Utah, Salt Lake City, 1985.
5. A. Glassner, "Space Subdivision for Fast Ray Tracing," *IEEE CG&A*, Oct. 1984, pp. 15-22.
6. S. Rubin and T. Whitted, "A 3-Dimensional Representation for Fast Rendering of Complex Scenes," *Computer Graphics* (Proc. SIGGRAPH 80), July 1980, pp. 110-116.
7. A. Fujimoto, "ARTS: Accelerated Ray-Tracing System," *IEEE CG&A*, April 1986, pp. 16-26.
8. M. Kaplan, "Space-Tracing, a Constant Time Ray-Tracer," *Computer Graphics* (Proc. SIGGRAPH 85), seminar notes from "State of the Art in Image Synthesis," July 1985.
9. G. Fox and S. Otto, "Algorithms for Concurrent Processors," *Physics Today*, May 1984, pp. 50-59.
10. D. Knuth, *The Art of Computer Programming*, Vol. 2, Addison Wesley, Reading, Mass., 1969, p. 139.
11. T. Kay and J. Kajiya, "Ray Tracing Complex Scenes," *Computer Graphics* (Proc. SIGGRAPH 86), Aug. 1986, pp. 169-278.
12. H. Weghorst, G. Hooper, and D. Greenberg, "Improved Computational Methods for Ray Tracing," *ACM Trans. on Graphics*, Jan. 1984, pp. 52-69.
13. L. Stone, *Theory of Optimal Search*, Academic Press, New York, 1975, pp. 27-28..
14. J. Goldsmith and J. Salmon, "A Ray Tracing System for the Hypercube," *Caltech Concurrent Computing Project Memorandum HM154*, Calif. Inst. Of Technology, Pasadena, Calif., 1985.

Jeffrey Goldsmith has been at Jet Propulsion Lab's Computer Graphics Laboratory since 1983. He is currently working on computer graphics on hypercube parallel processors as part of California Institute of Technology's Concurrent Computation Program. He was a contributor to *The Magic Egg*, the first computer-generated movie in Omnimax format, first shown at SIGGRAPH 84.

Goldsmith received his BS and MS in computer science from Rensselaer Polytechnic Institute in 1981 and 1983.

John Salmon is a graduate student in computation and neural systems at the California Institute of Technology. His research interests include parallel processing, computer graphics, and astrophysics. He received his BS in electrical engineering, computer science, and physics from the Massachusetts Institute of Technology in 1981 and an MS in physics from the University of California at Berkeley.

The authors can be contacted at the California Institute of Technology, Synchrotron Laboratory 206-49, Pasadena, California 91125.