Automatic Termination Criteria for Ray Tracing Hierarchies

K. R. Subramanian Donald S. Fussell

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

Abstract

A common problem in space subdivision hierarchies used in ray tracing is determining the proper termination criteria to stop subdivision. We propose a cost model based on scene characteristics that can be used to predict the correct termination point to optimize performance. The characteristics are determined as the hierarchy is being built. The model is applied to a variety of space subdivision schemes to test its accuracy. Experimental results indicate the power and usefulness of this model when applied to some standard ray tracing benchmarks.

Key Words: Ray tracing, bounding volume, extent, partitioning plane, search structure, traversal.

1 Introduction

Ray tracing has become established as an important and popular rendering technique for synthesizing photo-realistic images. However, ray tracing, if not carefully done, can be a computationally expensive technique. Consequently a great deal of research has focused on discovering efficient ways to perform ray tracing.

The principal expense in ray tracing lies in determining a ray's closest intersection to its origin with an object in a scene. This may be done several times per pixel, the exact number of times depending on the effects being generated and the scene being rendered. Research on efficient algorithms has quite properly focused on minimizing the cost of these intersection calculations. Many of the resulting techniques have employed data structures for speeding up the search for a closest intersection on a ray. Data structures that support efficient geometric search allow us to look at only a small percentage of the scene to determine the closest intersection. Octrees [7], BSP trees [11][14], and nested bounding volumes [8] are examples of explicitly hierarchical search structures of this type, while the uniform subdivision method [5][3] is a non-hierarchical search structure. Hybrid schemes have been explored in [15][10].

While all these methods are being used with great success, except in the case of nested bounding volumes, termination criteria for hierarchy construction algorithms have been totally ad hoc, leaving open the question of whether methods employing hierarchical search structures have really been exploited to their full potential. In this paper, we address this problem and propose methods to terminate subdivision at the most advantageous depth. A cost model is developed to relate the computational costs of various techniques to appropriate parameters for determining that a search structure of sufficient depth has been constructed. This model is built using statistical characteristics of the input scene. It is evaluated as the hierarchical structure is being built and stops subdivision when the cost reaches a minimum. We have applied this model to some commonly used ray tracing hierarchies, as well as to the uniform subdivision method. Experimental results illustrate the accuracy and usefulness of our model for optimizing the performance of these search structures.

2 The Problem

We will be concerned with several of the common hierarchical structures being used in ray tracing, including BSP trees, octrees, and bounding volume hierarchies. We also consider uniform subdivision and a space subdivision technique that uses *k-d* trees [6][13]. We begin with a brief description of each of these methods.

A BSP tree is any binary tree structure used to recursively partition space. In Kaplan's implementation of the BSP tree [11], axis-aligned planes are used to partition space. At each step of the subdivision, three slicing planes are used to divide space into eight equal sized octants. The recursive subdivision continues until the voxels contain either a small number of primitives or their size becomes smaller than a set threshold. In order to optimize performance, this threshold must be set correctly. If the threshold is too high, then large numbers of primitives end up in each voxel; if it is too low, getting to the leaf nodes from the root of the tree is more expensive. Note that the subdivision is adaptive, in that only voxels that contain primitives are subdivided. Thus, the structure adapts itself to the input scene.

The octree hierarchy used by Glassner [7] performs a subdivision identical to Kaplan's BSP tree. Each level of the octree corresponds to three levels of the BSP tree. The difference lies in the way Glassner stores the octree. While Kaplan builds a binary tree, Glassner uses a hash table, which results in considerable savings in pointer space. The termination problem in the octree is thus the same as in the BSP tree.

We have performed an extensive study of some of the important properties of search structures used to accelerate ray tracing [13]. This has led to the development of a highly adaptive search structure based on the *k-d* tree, a special case of BSP trees introduced by Bentley [1][2]. Our *k-d* tree structure uses axis-aligned partitioning planes like many other space subdivision methods. Our study has shown that the SA (Surface Area) heuristic for deciding the locations of the partitioning planes used in [14] combined with the judicious use of bounding volumes at some nodes of the hierarchy to prune excessive void space provides the good performance.

The SA heuristic minimizes the following function in trying to locate a space partitioning plane:

$$f(b) = \mathrm{SA}_l(b).n_l + \mathrm{SA}_r(b).n_r \qquad (1)$$

where b is a partitioning plane, n_l, n_r are counts of objects on both sides of b, and $SA_l(b), SA_r(b)$ are surface areas on both sides of b. In our implementation, $SA_l(b)$ and $SA_r(b)$ are measured by the surface areas of the bounding volumes that enclose the objects on either side of the partition. Once a partitioning plane is determined, the sets of objects on each side of the plane are recursively partitioned so long as there is at least one primitive on each side of the partitioning plane and at least one of them is completely on one side of the plane. Insertion of a plane creates a pair of nodes, the union of whose extents equals the parent's extents. In cases where a new node's extents are much larger than the bounding box of the objects included in the node, this bounding box is stored in the node.

The uniform subdivision method subdivides space into a 3-d grid of equal-sized voxels. To determine the closest intersection of a ray with an object in the scene, a modified form of Bresenham's line algorithm in three dimensions is used traverse the voxels efficiently along the ray. The non-adaptive nature of this structure can result in large regions of empty voxels, which are expensive to traverse. However, restricting the subdivision can also put large numbers of primitives in some of the voxels. Thus, it is difficult to know the correct amount of subdivision without any knowledge of the scene characteristics.

The major problem in bounding volume hierarchies is finding suitable clusters of primitives to build the hierarchy from the point of view of performance. Goldsmith's [8] automatic bounding volume hierarchy (ABV) takes an important step in this direction. The bounding volume surface area is shown to be intimately related to the cost of the hierarchy. To construct the hierarchy of volumes, each object is considered a prospective child of each node to be searched. When the search reaches the leaf nodes, the new node and the leaf node are proposed as siblings of a new non-leaf node, replacing the old leaf node. After the search, the object is inserted into the tree where it causes the least increase in the total bounding volume surface area.

3 The Cost Model

Ray tracing hierarchies are built for the sole purpose of speeding up the intersection search. All of these structures help in drastically reducing the search space of each ray. This is accomplished in two different ways:

1. The search is ordered along the path of the ray, starting from its origin. This helps in terminating the search once an intersection is found.
2. The search examines only parts of the scene that are close to the ray. Even if no intersection is found, only a fraction of the scene would have been examined.

However, using a search structure introduces a new expense: the cost of traversing it. So long as the cost in traversing the structure is overwhelmed by the gains in reducing the ray-search space, we are improving performance. The question is, what is the cutoff point?

3.1 Search Structure Costs

We can identify two major costs involved in using a search structure:

1. The cost in examining the scene, $C_{sc}(h,s)$(h is the height and s is a search structure). This is the cost of performing ray-object intersections and ray-bounding volume intersection tests (when object primitives are enclosed by bounding volumes).
2. The cost in traversing the search structure, $C_{tr}(h,s)$. This is the cost of going down the hierarchy to the leaf nodes. Depending on the actual search structure, this could involve partitioning plane intersections, bounding volume tests in the internal nodes of the hierarchy, or just compares between ray coordinates and the partitioning planes. It also accounts for the cost involved in determining the next region along the path of the ray to be searched.

Other costs in ray tracing such as building the search structure and lighting calculations are not significant when compared to the total run time. As we start subdividing the scene, $C_{sc}(h,s)$ decreases and $C_{tr}(h,s)$ increases. The rates of increase/decrease of these two costs will determine the performance of the search structure. We are interested in terminating the search structure at the height that minimizes $C_{sc}(h,s) + C_{tr}(h,s)$.

3.2 Determining $C_{sc}(h,s)$

Our next step is to determine estimates for $C_{sc}(h,s)$ and $C_{tr}(h,s)$. To be of any practical use, the expressions that we obtain must be dependent on characteristics of the scene being rendered that can be determined easily.

Let us look at $C_{sc}(h,s)$, the cost involved in examining the scene. What we want to know is, at any particular level of subdivision, what portion of the scene is examined by each ray. One way we can determine this is to try to compute the number of primitives examined by a ray on the average. So

$$C_{sc}(h,s) = \frac{C_{pr}}{n}\sum_{i=0}^{n-1}\sum_{r=0}^{R_i(h,s)} n_{pr}(i,r,h,s)$$

where
C_{pr} = cost of examining a primitive for intersection.
$n_{pr}(i,r,h,s)$ = number of primitives examined by ray i in region r.
n = total number of rays spawned.
$R_i(h,s)$ = number of regions examined by ray i.
h = height of search structure.
s = search structure.

Determining $n_{pr}(i,r,h,s)$ before ray tracing is not easy. However, the dependency of $n_{pr}(i,r,h,s)$ on region r can be removed by approximating it by an average region primitive count. The dependency of $R_i(h,s)$ on i also makes it a quantity difficult to compute before ray tracing. Approximating $R_i(h,s)$ by $R(h,s)$, the expected number of regions visited by any ray before it terminates, the scene cost becomes

$$\begin{aligned} C_{sc}(h,s) &= \frac{C_{pr}}{n}(nR(h,s))n_{pr}(h,s) \\ &= C_{pr}R(h,s)n_{pr}(h,s) \end{aligned}$$

where $n_{pr}(h,s)$ = average number of primitives in each region of the search structure s of height h.

$n_{pr}(h,s)$ can be determined by summing up the object counts at the leaf nodes of the hierarchy (for the uniform subdivision method, the object counts in each voxel). A weighted average of these counts is calculated, the weights being a measure of the size of the regions. For instance, the bounding volume surface area can be used to weight the object counts. This is necessary since larger sized regions have a higher probability of being visited more often (as we will show next). C_{pr} can be taken as the average cost of a bounding volume intersection test. In our implementation, all object primitives are enclosed by bounding volumes. The only other unknown quantity, $R(h,s)$, the expected number of regions examined by each ray, will be estimated as follows.

In determining $R(h,s)$, we must bear in mind that the intersection search is ordered along the path of the ray. Once an intersection is found, processing stops for that particular ray. How quickly this might happen depends on the scene complexity, in terms of how dense or space filling the primitives in the scene are.

Let p_j represent the probability that the ray has an intersection with a primitive in region j. Then $(1-p_j)$ will be the probability that the ray will not intersect any primitive in region j. Also, let us assume that the ray is within the scene of interest, so that at least one region must be examined for intersection. The expected number of regions visited by a ray under these conditions is given by the following relation,

$$R(h,s) \quad = \quad MAX(1, \sum_{i=1}^{k} ip_i \prod_{j=1}^{i-1}(1-p_j))$$

Again, we can use an average region probability instead of the $p_j s$. Let this probability be p, a weighted average of the probabilities accounting for the different sizes of the regions. The above expression becomes

$$\begin{aligned} R(h,s) &= MAX(1, \sum_{i=1}^{k} ip \prod_{j=1}^{i-1}(1-p)) \\ &= MAX(1, \sum_{i=0}^{k} ip(1-p)^{i-1}) \\ &\approx 1/p \qquad \text{for large } k. \end{aligned}$$

3.3 Determining Region Probability p

The average region probability p is the (average) probability with which an incoming ray penetrates any object primitive in a region Determining this accurately is very expensive and might be impossible since it depends on the geometry of the region, the primitives in it and the ray distribution. Thus we need to find a reasonable approximation.

An approximation to this has already been used by Goldsmith [8]. If the ray directions are assumed to be uniform, then the conditional probability that a ray will penetrate a convex region B given that it penetrates enclosing convex region A is equal to the ratio of the average projected area of B to that of A. It can be shown that the average projected area of a convex region is equal to one quarter of its surface area [4][13], so

$$P(B|A) = \frac{Area_B}{Area_A}$$
$$P(C|A) = \frac{Area_C}{Area_A}$$

The region probability can be estimated by enclosing the collection of primitives by a convex bounding volume. Since most space subdivision methods produce convex partitions, the regions are already convex. The ratio of primitives' bounding volume surface area to that of the region gives an estimate of the conditional probability. A more accurate value of p can be obtained by enclosing individual primitives with bounding volumes, thus accounting for the void space between primitives. However, if two bounding volumes overlap, then the overlap area has to be subtracted out since it cannot be counted twice. In our implementation, we use the ratio of bounding volume surface areas to estimate the conditional probability. Each of the region probabilities must be weighted by the region size (again, the region surface area can be used) when we compute the average probability. For instance, if A_i

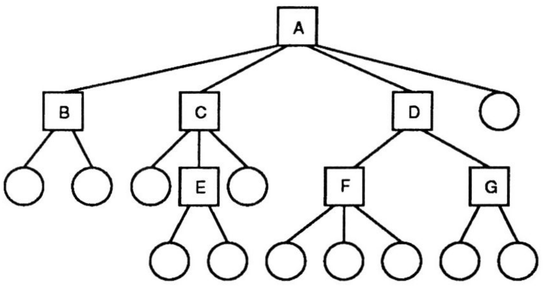


Figure 1: A Bounding Volume Hierarchy

Ray counts are in thousands

Scene	Tetra	DNA	Arches	Balls	Geo58
Objects	1024	410	4818	7382	306
Obj. Type	P	S	P	PS	P
Lights	1	1	2	3	3
Total rays	299	445	437	1819	847
Visual rays	262	323	317	722	392
Shadow rays	37	122	120	1097	455
Hit rays	44	75	73	873	278

Table 1: Statistics of Test Scenes.

represents the bounding volume surface area and E_i, the surface area of the extent of the same region ($A_i \leq E_i$), then the region probability is given by

$$p = \frac{\sum_{i=1}^{n} E_i * \frac{A_i}{E_i}}{\sum_{i=1}^{n} E_i} = \sum_{i=1}^{n} A_i/E_i$$

3.4 Determining $C_{tr}(h,s)$

The traversal cost $C_{tr}(h,s)$, in general, is bounded by the following form.

$$C_{tr}(h,s) = R(h,s) * C_r(h,s)$$

where
$R(h,s)$ = expected number of regions examined.
$C_r(h,s)$ = average traversal cost expended per region.

$C_r(h,s)$, in general, involves the cost of determining the region containing the origin of the ray and the cost of identifying the next region visited by the ray if the intersection search is unsuccessful in the current region. For hierarchical structures, the cost of reaching the leaf nodes from the root has to be calculated. This is found by multiplying the work done per node by the average height of the hierarchy. The average height of the hierarchy is also a weighted quantity, since paths leading to nodes which are larger in size will be visited more often by rays when compared to paths leading to smaller sized regions. For the uniform subdivision method, $C_r(h,s)$ is determined by the work done in moving from the current voxel to the next voxel visited by the ray.

3.5 Modification for Bounding Volume Hierarchies

In bounding volume hierarchies, the traversal of the hierarchy is usually not along the path of the ray. The expected number of regions examined by each ray, $R(h,s)$, is calculated in a different way. The method is outlined in detail in [8]. Consider the simple bounding volume hierarchy in Fig. 1. Square nodes are internal nodes containing the bounding volume of the subtree of the scene. Circle nodes representing primitive objects are the leaf nodes.

Assume only rays intersecting the root bounding volume are of interest. The expected number of regions examined is given by (refer to [8] for details)

$$\begin{aligned} R(h,s) &= 1 + 4P(A|A) + 2P(B|A) + 3P(C|A) + \\ &\quad 2P(D|A) + 2P(E|A) + 3P(F|A) + 2P(G|A). \end{aligned}$$

where $P(I|J)$ represents the conditional probability that node I is intersected given that J has already been intersected. The conditional probabilities are determined by using the approximation given in the previous section. So the total cost is

$$C = C_{pr} R(h,s) n_{pr}(h,s)$$

where C_{pr} is the cost of a bounding volume test and $n_{pr}(h,s)$, the average number of primitives at each leaf node. In Goldsmith's method, $n_{pr}(h,s) = 1$. The above equation represents the total cost.

4 Implementation and Experimental Results

We have implemented the uniform subdivision method, BSP tree, octree, automatic bounding volume hierarchy and the *k-d* tree methods to test the cost model. All experiments were conducted on a Sun 4/280 workstation running SunOS UNIX[1] 4.0.3. Five different data-sets were used as test cases. Details of these datasets are given in Table 1, where P stands for polygons and S for spheres. Images of these models are shown in Plates 1 through 5. Several of these are standard benchmarks [9] available in the public domain. The termination parameter used in all these methods (except for uniform subdivision, where it is the grid resolution *n_grid*) is the maximum height.

In our implementation, each primitive is surrounded by a bounding volume which is an axis-aligned parallelepiped. Bounding volumes around collections of primitives are also axis-aligned parallelepipeds. C_{pr}, the cost of testing a primitive is taken to be the cost of a bounding volume test. In our implementation C_{pr} = 15.5 floating point operations. In all the methods except the ABV hierarchy, the search stops as soon as an intersection is found. Also, duplicate object intersection tests are avoided by

[1]UNIX is a trademark of AT&T Bell Laboratories.

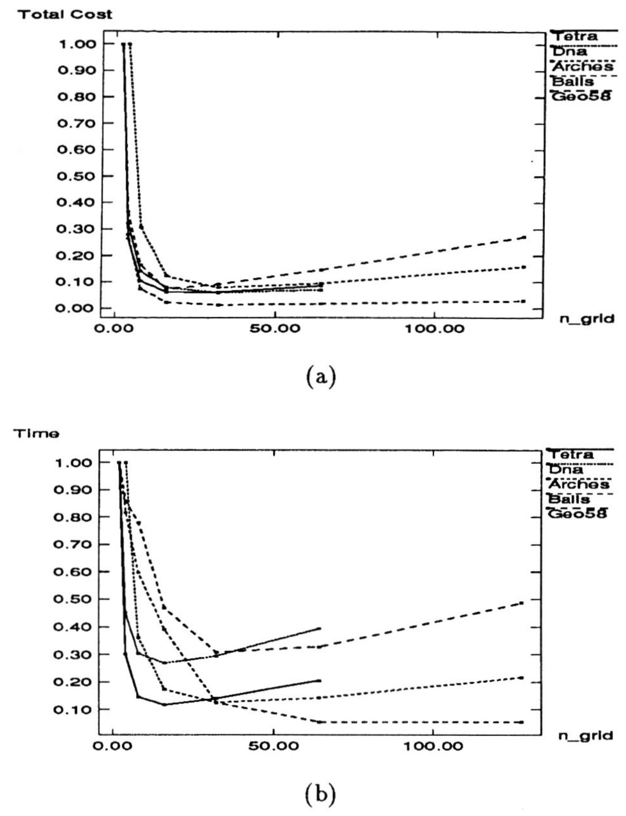


Figure 2: Unif. Subd. Perf. Characteristics (a) predicted (b) Actual

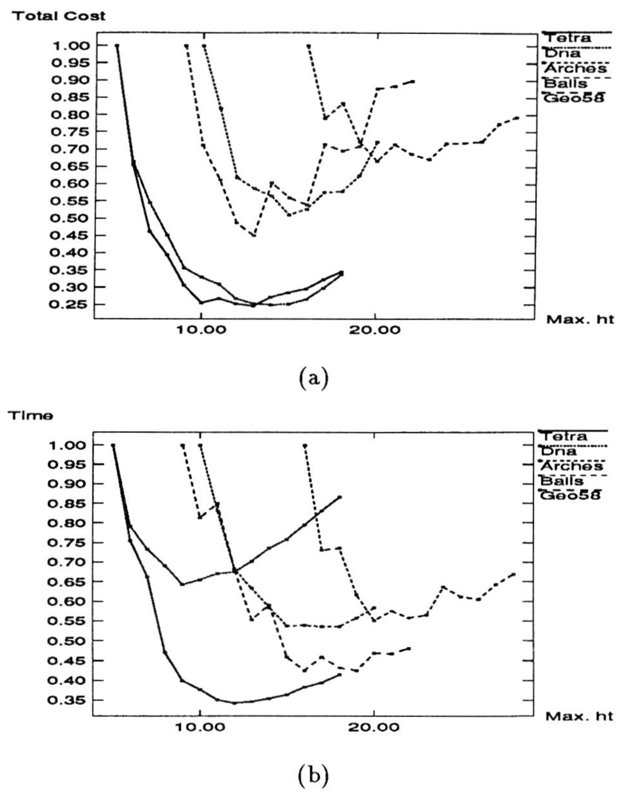


Figure 3: Kaplan-BSP Perf. Characteristics (a) Predicted (b) Actual

maintaining ray signatures in all object records. An object is tested for intersection with a ray only if it has not been examined by the current ray.

As we start building the hierarchy, $C_{sc}(h,s)$ and $C_{tr}(h,s)$ are computed for each value of the maximum height. Initially, the decrease in $C_{sc}(h,s)$ will overwhelm the increase in $C_{tr}(h,s)$. What we are looking for is the point where the total cost reaches a minimum. It is possible that there might be several local minima (for example, the BSP and octree structures exhibit this characteristic). If these are all close to each other, we could pick any one of the minima with little difference on performance.

For each method, we plot both the predicted and actual performance characteristics as a function of the termination parameter. For the predicted characteristic, the total cost $= C_{sc} + C_{tr}$. This is plotted against the maximum height (the grid resolution, for the uniform subdivision method). The actual characteristic is a plot of the running time (which is a measure of the total cost) versus the termination parameter. The total cost has been normalized from 0.0 to 1.0 so as to fit all the test cases in the same graph plot.

4.1 Uniform Subdivision Method

Our implementation of uniform subdivision method follows very closely the algorithms outlined in [3]. In this structure, all voxels are of uniform size. Traversing a ray involves identifying the voxels along the path of the ray in the three-dimensional grid. In our implementation, it takes about 6.5 floating point operations to move from one voxel to the next, on average. Thus $C_r = 6.5$. C_{pr} is the cost of a bounding volume test, as before. The region probabilities and the average leaf object counts are computed as explained previously.

Fig. 2 shows the performance characteristics and results for the uniform subdivision method. Here n_grid is the resolution of the grid in each of the three dimensions. For testing the cost model, the resolution is doubled in all three dimensions each time we subdivide. In our implementation, we used polyhedral bounding boxes [12] to enclose the primitives when clipping to voxels. The clipped points were used in computing an axis-aligned bounding box within the voxel. The surface area of this box was used in computing the region probability. More accurate methods of determining the surface area of the primitive within the voxel will improve the predictions.

4.2 BSP Tree

Our implementation of the BSP tree hierarchy is very similar to that of Kaplan's [11]. One difference is that each subdivision step does not necessarily produce eight octants. If for example, a plane has subdivided the original space into two equal sized voxels and one of them does not contain any primitives, then it is not subdivided by the remaining two planes. This results in a smaller number of empty regions, thus making the structure more adaptive to the scene.

The traversal method used in our implementation is described in [6]. In this method, the ray is intersected with the separating plane stored at each node of the hierarchy to decide the order in which the regions are traversed. On the average, our implementation requires 4.5 floating point operations ($C_r = 4.5$) to make this decision. This has to be multiplied by the average height of the hierarchy, to account for the work done to reach the leaf nodes.

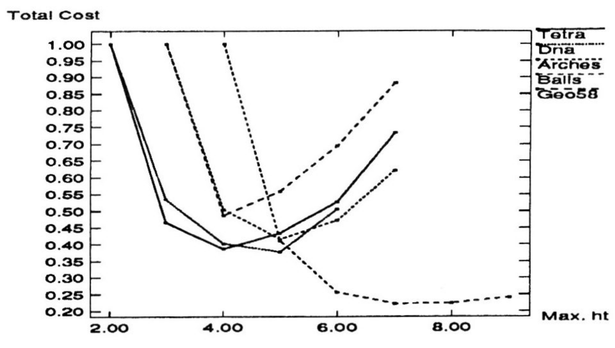
Fig. 3 shows the performance characteristics and results for the BSP tree structure for several scenes. Overall, the predicted heights are very close to the optimal heights obtained by the experiments. In general, for the BSP tree structure, there is a small range of heights at which the performance stays relatively constant. It is interesting to note that in some of these cases, optimal performance occurs well beyond the logarithm of the number of objects.

4.3 Octree

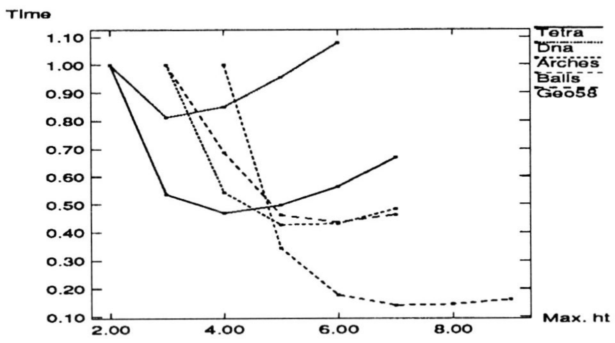
To implement the octree, we modified our BSP tree implementation so that at each step of the subdivision, eight equal sized voxels were created. The data was then collected as in the BSP tree case. The traversal method used is the same as in the BSP tree method. Fig. 4 shows the performance characteristics and results of using the cost model. As expected, the performance is slightly worse (Balls and Geo58 scenes) than the BSP tree case because of the additional empty voxels that need to be processed in the octree. Note that each level of the octree corresponds to three levels of the BSP tree.

4.4 Automatic Bounding Volume Hierarchy

Goldsmith's automatic bounding volume hierarchy also stops building the hierarchy beyond a certain height since a heuristic search determines an insertion point in the hierarchy for each object primitive. Fig. 5 illustrates the performance characteristics of the automatic bounding volume hierarchy and results of using the cost model. In the DNA model, there is a height at which the cost reaches a minimum (which illustrates that even the ABV hierarchy is not immune to the termination problem), while in all the other cases, the cost remains flat after a certain height. Since all the scenes except Tetra have large numbers of secondary rays, the cost of going down the hierarchy has to be added to the predicted cost.



(a)



(b)

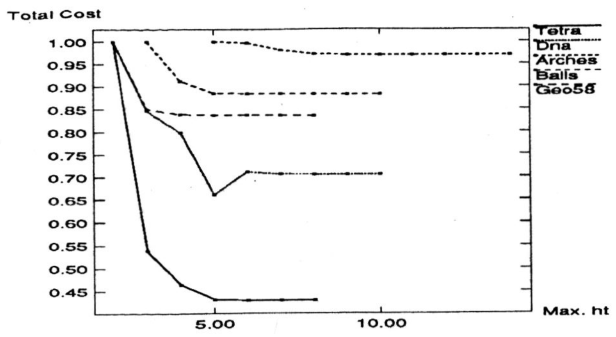
Figure 4: Octree Perf. Characteristics (a) Predicted (b) Actual

4.5 K-d Tree

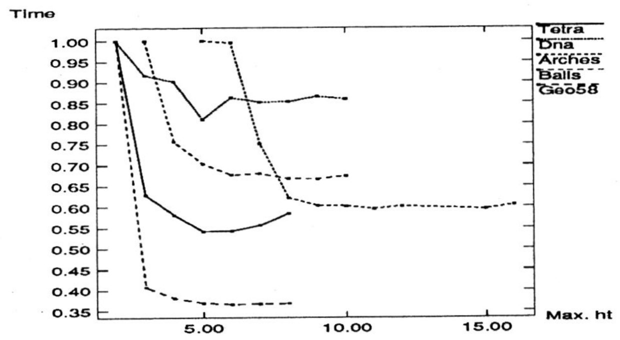
Fig. 6 illustrates the actual characteristics of the *k-d* tree. Notice that the greater flexibility in locating partitioning planes makes the cost go down smoothly as the subdivision level increases (in contrast to the ripples in the Kaplan-BSP characteristics). The characteristics flatten out when the construction algorithm automatically terminates subdivision. In each case, optimal performance is reached only at the height at which the original structure would not have subdivided any further. One difference from the octree and BSP hierarchies is the presence of bounding volumes in the internal nodes of the hierarchy. To include this in the predicted cost, we need to determine the average number of bounding volumes along any path of the *k-d* tree. This value is multiplied by the cost of a bounding volume test (C_{pr}) and then added to traversal cost.

5 Conclusions

Overall, the termination predictions are quite accurate. In the cases where the prediction is off the experimentally obtained optimal point for subdivision termination, the difference in performance is usually quite small. One thing to notice in the performance characteristics is that



(a)



(b)

Figure 5: ABV Hierarchy Perf. Characteristics (a) Predicted (b) Actual

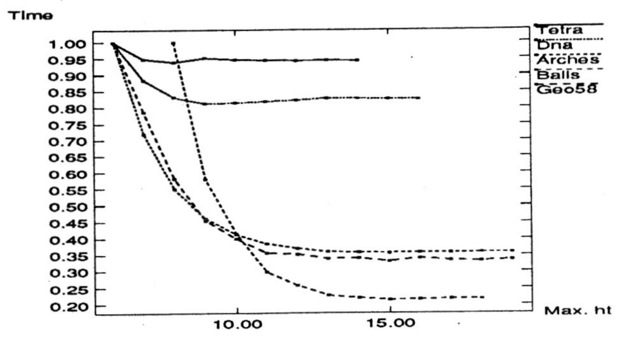


Figure 6: K-d Tree Perf. Characterstics - Actual

the absolute values of the cost do not agree well with the experimentally obtained values. This is not of too much concern since the relative changes in the characteristics from point to point are more important than their absolute values. Addition of our technique to the uniform subdivision method, BSP trees and octrees, and ABV hierarchies provides an effective means of building near optimal automatic termination criteria into these subdivision techniques, and we have validated the criterion used in the *k-d* method. Thus a major unknown factor affecting the performance of such techniques has been eliminated, allowing them to tune themselves for optimal performance with high confidence.

References

[1] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9), September 1975.

[2] Jon Louis Bentley. Data structures for range searching. *Computing Surveys*, 11(4), December 1979.

[3] John G. Cleary and Geoff Wyvill. Analysis of an algorithm for fast ray tracing using uniform space subdivision. *Visual Computer*, 4(2):65–83, July 1988.

[4] H.C. Van de Hulst. *Light Scattering by Small Particles*, page 60. Dover Publications, Inc., New York, 1981.

[5] Akira Fujimoto, Takayuki Tanaka, and Kansei Iwata. Arts: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, April 1986.

[6] Donald Fussell and K.R.Subramanian. Fast ray tracing using k-d trees. Technical Report TR-88-07, Department of Computer Sciences, The University of Texas at Austin, March 1988.

[7] Andrew S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, October 1984.

[8] Jeff Goldsmith and John Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, pages 14–20, May 1987.

[9] Eric A. Haines. A proposal for standard graphics environments. *IEEE Computer Graphics and Applications*, pages 3–5, November 1987.

[10] David Jevans and Brian Wyvill. Adaptive voxel subdivision for ray tracing. *Graphics Interface*, May 1989.

[11] Michael R. Kaplan. The uses of spatial coherence in ray tracing. *ACM SIGGRAPH Course Notes 11*, July 1985.

[12] Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. *Computer Graphics*, 20(4):269–278, August 1986.

[13] K.R.Subramanian. *Adapting Search Structures to Scene Characteristics for Ray Tracing*. PhD thesis, Dept. of Computer Sciences, The University of Texas at Austin, December 1990.

[14] J. David Macdonald and Kellog S. Booth. Heuristics for ray tracing using space subdivision. *Visual Computer*, 6(3), June 1990.

[15] John M. Snyder and Alan H. Barr. Ray tracing complex models containing surface tesselations. *Computer Graphics*, 21(4):119–128, July 1987.

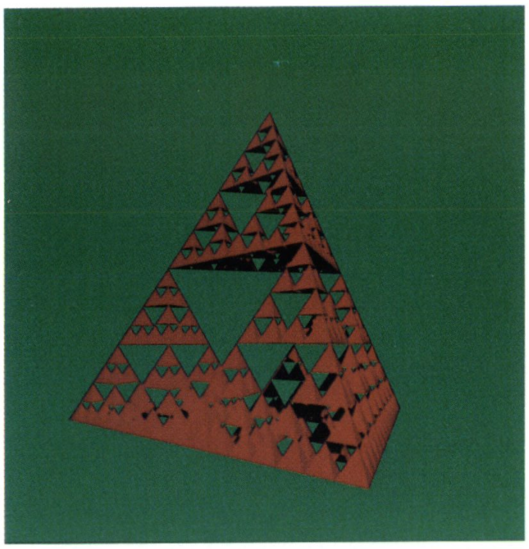


Plate 1. Tetra.

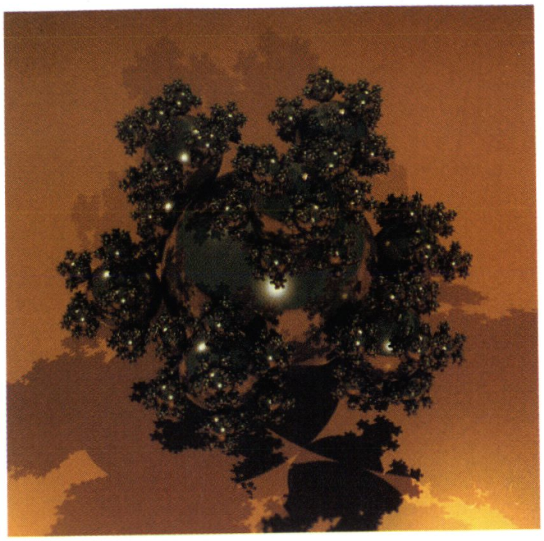


Plate 4. Balls.

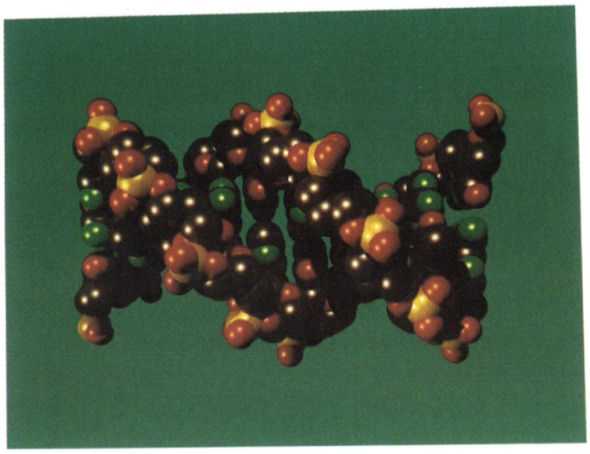


Plate 2 DNA.

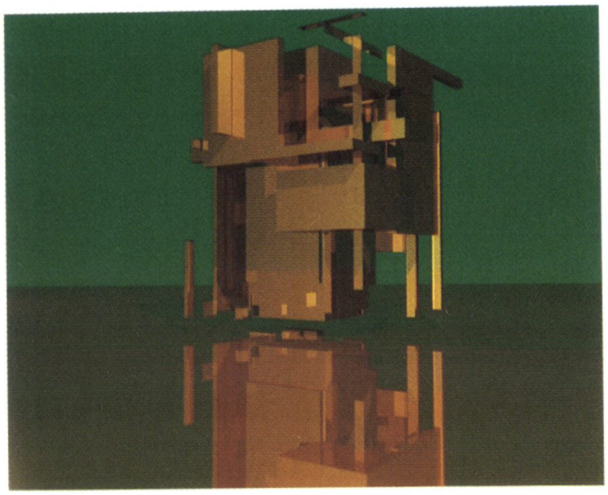


Plate 5. Geo58.

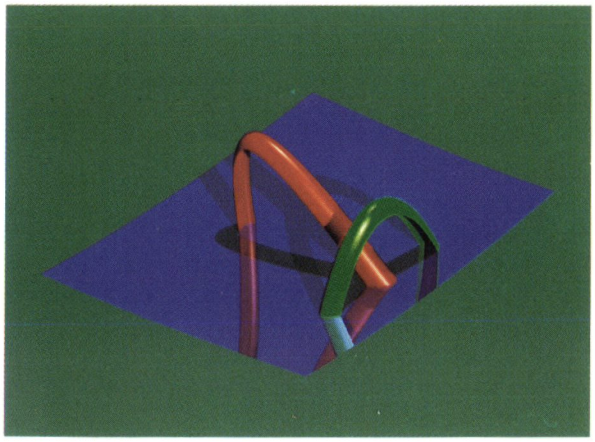


Plate 3. Arches.