A COST MODEL FOR RAY TRACING HIERARCHIES

K. R. Subramanian and Donald Fussell

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-90-04 March 1990

A Cost Model for Ray Tracing Hierarchies

K. R. Subramanian Donald Fussell

Department of Computer Sciences
The University of Texas at Austin

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.

CR categories and Subject Descriptors:
I.3.3 [Computer Graphics]: Picture/Image Generation;
I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism;

General Terms: Algorithms, Graphics.

Additional Key Words and Phrases: Ray tracing, computer graphics, bounding volume, extent, separating plane, hierarchy, search structure, traversal.

1 Introduction

In recent years, ray tracing has become established as an important rendering technique. Its ability to determine visible surfaces, compute shadows, model reflection and transmission of light and a host of other effects[5] has contributed to its growing use. 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 must be done several times per pixel, the exact number 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. Bounding volumes[14][16][19][20], space partitioning structures[6][8][13], item buffers[19], shadow buffers[11] and ray coherence techniques[1][12][17] all have proven effective at improving the efficiency of ray tracing.

Many of these techniques have employed data structures for speeding up the search for a closest intersection on a ray. Data structures which support efficient geometric search allow us to look at only a small percentage of the scene to determine the closest intersection. Octrees[8], BSP trees[13], and nested bounding volumes[9] are examples of explicitly hierarchical search structures of this type, while ARTS[6] is a non-hierarchical search structure.

While all these methods are being used with great success, except in the case of nested bounding volumes, no work has revealed how deep the hierarchical data structures should be constructed to achieve optimum results. Heretofore 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 ARTS 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[13], octrees[8], and bounding volume hierarchies[16][9]. In addition to these, we consider a hierarchical technique related to the first two which uses $k-d$ trees[7][15] as well as ARTS[6][4], a uniform grid subdivision technique. 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[13] implementation of the BSP tree, 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.

To determine a ray-object intersection from the BSP tree, the origin of the ray is inserted into the root of the tree. Comparing this against the slicing planes in the tree, the leaf node (voxel) containing this point can be determined. All primitives in this voxel are intersected with the ray. If an intersection is found within this voxel, the closest of these is the required intersection. Otherwise, the face through which the ray exits this voxel is determined by intersecting the ray against all the faces of this voxel. The ray is then extended a small amount, depending on the size of the smallest voxel in the tree. This end point is put back into the root of the tree and the same process continues.

The octree hierarchy used by Glassner[8] 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 traversal is also identical to the BSP tree. The termination problem in the octree is thus the same as in the BSP tree.

The $k-d$ tree[2][3] hierarchy used in [7] is also implemented as a binary tree using axis-aligned partitioning planes, but it has greater flexibility in the location of the partitioning planes as well as the choice of the partitioning dimension. The initial node is simply the scene extents. Nodes are partitioned by planes whose locations are chosen by performing a binary search of potential plane positions within the node extents along all three dimensions. The plane that best balances the number of primitives on both sides of the partition and contains the least number of primitives straddling it is the desired partitioning plane. Subdivision continues as 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.

To traverse the $k-d$ tree. a ray is intersected with the partitioning plane stored at the root of the hierarchy. By looking at the ray parameters (its origin and direction), it is simple to determine the order in which the regions should be examined. Regions are examined in order of increasing distance from the ray origin. If the ray intersects the bounding volume for a node (either its extents as determined by partitioning planes or the bounding box of its included objects, as appropriate), the node's children are processed next, otherwise the node's siblings which are further from the ray origin are processed next. The entire traversal can be done in ray parameter space. This traversal can also be used with the octree or BSP tree hierarchies.

The ARTS[6] method subdivides space like the octree, but the subdivision is uniform, resulting in 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 to step through the voxels, starting from the ray origin. Though the structure does not adapt itself to the scene, it enables the use of a traversal algorithm which can be implemented efficiently using incremental techniques. The non-adaptive property of this structure can potentially result in large regions of empty voxels, which will be 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[9] automatic bounding volume hierarchy takes an important step in this direction. The bounding volume surface area is shown to be intimately related to the cost of the hierarchy. The hierarchy is hence built with a view to minimize the total bounding volume surface area. Objects are inserted into the hierarchy, one at a time, where they would cause the least increase in the total bounding volume surface area of the hierarchy. To accomplish this, each object is considered a prospective child of each node that will 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 in the tree where the increased cost of the hierarchy is minimized.

To traverse the bounding volueme hierarchy, the ray is intersected with the bounding volume stored at the root node. If the ray intersects it, then all the node's children have to be examined. If not, its entire subtree can be removed from further consideration. This process continues recursively until there are no more nodes to be considered. Throughout the process, a record of the closest intersection, if any, is maintained and reported at the end of the traversal. Although, there are no parameters that need to be decided before we build the hierarchy, we will attempt to see if the hierarchy's performance is affected by a variation in its maximum height.

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 ray 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, $C_{sc}(t, s)(t$ is a termination parameter and s is a search structure). This usually involves bounding volume or object intersections.
2. The cost in traversing the search structure, $C_{tr}(t, s)$. This usually takes the form of comparing the ray coordinates to the region being examined, in order to determine the search order of the traversal. Depending on the method being used, this cost could be in terms of bounding volume or plane intersections, or it could be just incremental calculations.

As we start subdividing the scene, 1 decreases and 2 increases(or vice versa, depending on what t is). The rates of increase/decrease of costs will determine the performance of the search structure. In Fig. 1, $C_{sc}(t, s)$ and $C_{tr}(t, s)$ are two cost functions. To get the optimum value for t, we equate the absolute values of the slopes of the two functions and solve for t_{opt}.

$$C'_{sc}(t, s) + C'_{tr}(t, s) = 0$$

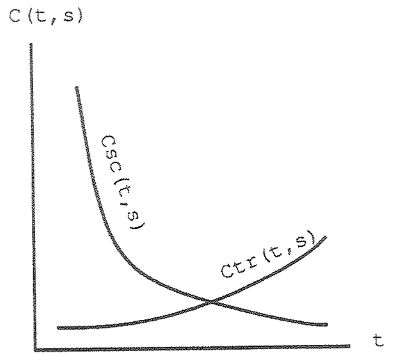

Fig. 1. Search Structure costs.

or

$$C'_{sc}(t,s) = -C'_{tr}(t,s)$$

At $t = t_{opt}$, $C_{sc}(t_{opt}) + C_{tr}(t_{opt})$ will be a minimum.

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

Our next step is to determine estimates for $C_{sc}(t,s)$ and $C_{tr}(t,s)$. To be of any practical use, the expressions that we obtain must be dependent on the characteristics of the scene that we are trying to render.

Let us look at $C_{sc}(t,s)$, 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 could determine this is to try to compute the number of primitives examined by a ray on the average. So

$$C_{sc}(t,s) = C_{pr} \sum_{i=0}^{n} \sum_{r=0}^{R_i} n_{pr_i}(i,r,t,s)$$

where

C_{pr} = cost of examining a primitive for intersection.

$n_{pr}(i,r,t,s)$ = number of primitives examined by ray i in region R.

n = total number of rays spawned.

R_i = number of regions examined by ray i.

t = termination parameter.

s = search structure.

Determining $n_{pr}(i, r, t, s)$ before the ray trace is not easy. However, the dependency of $n_{pr}(i, r, t, s)$ on region r can be removed by approximating n_{pr} by an average region primitive count.

$$\begin{aligned} C_{sc}(t, s) &= C_{pr} \sum_{i=0}^{n} \sum_{r=0}^{R_i} n_{pr}(i, t, s) \\ &= C_{pr} \sum_{i=0}^{n} R_i n_{pr}(i, t, s) \end{aligned}$$

where

$n_{pr}(i, t, s)$ = average number of primitives per region.

R_i, the number of regions examined by each ray, is also difficult to obtain before the ray trace. Again, approximate R_i by R, the average number of regions traversed by each ray before it terminates.

$$\begin{aligned} C_{sc}(t, s) &= C_{pr} \sum_{i=0}^{n} R_i n_{pr}(i, t, s) \\ &= C_{pr} \sum_{i=0}^{n} R n_{pr}(t, s) \\ &= C_{pr} n R n_{pr}(t, s) \end{aligned}$$

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

Since we are now dealing with average quantities, it is sufficient if we can determine the expected number of primitives examined by each ray. Dividing the above equation by the total number of spawned rays n, the cost becomes

$$C_{sc}(t, s) = C_{pr} R n_{pr}(t, s)$$

n_{pr} can be determined by examining the regions of the search structure containing collections of object primitives. The only other unknown quantity, R, the expected number of regions examined by each ray, will be estimated as follows.

In determining R, we must bear in mind that the intersection search is an ordered search along the path of the ray. Once an intersection is found, then 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. Fig. 2 illustrates a ray starting at its origin O and being traced in the direction $\vec{d}$. Regions 1 through 4 are in the path or close to the ray and will be examined in this order. The question is, how many

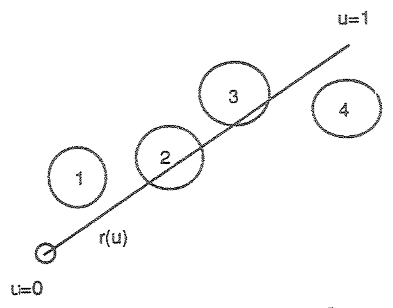

Fig. 2. Ray traversal.

of these regions will be examined? For this, we need some knowledge of the probability of a ray-primitive intersection in each of these regions.

Let p_j represent the probability that the ray intersects 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. So

P(1 region will be examined) $= 1$
P(2 regions will be examined) $= (1 - p_1)$
P(3 regions will be examined) $= (1 - p_1)(1 - p_2)$
....
....
P(k regions will be examined) $= (1 - p_1)(1 - p_2)....(1 - p_k)$

The number of regions examined is given by the following relation,

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

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

$$\begin{aligned} R &= MAX(1, \sum_{i=1}^{k} ip \prod_{j=1}^{i-1} (1 - p)) \\ &= MAX(1, \sum_{i=1}^{k} ip(1 - p)^{i-1}) \end{aligned}$$

$$= MAX(1, \sum_{i=0}^{k} ip(1-p)^{i-1})$$

A closed form solution to the sum of this series is derived in the appendix. For large values of k, this series approaches $(1/p)$. The final expression we get is as follows.

$$\begin{aligned} R &= MAX(1, \frac{1}{p}[\{1-(1-p)^{k+1}\} - p(k+1)(1-p)^k]) \\ &\approx MAX(1, 1/p) \\ &\approx 1/p \end{aligned}$$

since $1/p \geq 1$.

3.3 Determining $C_{tr}(t,s)$

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

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

where

$R =$ expected number of regions examined by the ray

$C_r(t,s) =$ average traversal cost expended per region.

Thus, the higher R is, the more $C_{tr}(t,s)$ will be.

3.4 Determining Region Probability p

The average region probability p is the probability with which a primitive in a region will be intersected by an incoming ray. 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[9]. If the ray directions are assumed to be uniform, then the probability with which a ray will penetrate a convex region is approximately proportional to its surface area[18]. In fig. 3, A, B, C and D are convex regions. The probability that a ray will penetrate

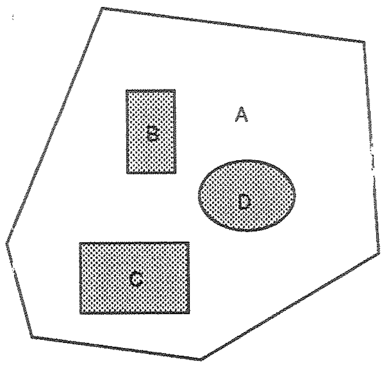

Fig. 3.

region B or C given that it penetrates A is given by

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

and so on.

The region probability can be computed by enclosing the collection of primitives by a convex bounding volume. Since most of 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 simpler approximation. Lastly, 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.

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, is calculated in a different way. The method is outlined in detail in [9]. Consider the simple bounding volume hierarchy in fig. 4. 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.

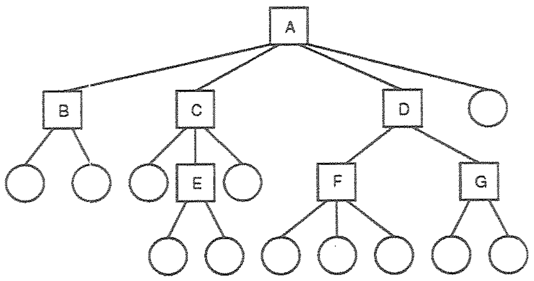

Fig. 4. A Bounding Volume Hierarchy

#nodes examined at level 0 = 1.

#nodes examined at level 1 = $4P(A|A)$.

#nodes examined at level 2 = $2P(B|A) + 3P(C|A) + 2P(D|A)$.

#nodes examined at level 3 = $2P(E|A) + 3P(F|A) + 2P(G|A)$.

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

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

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

where C_{pr} is the cost of a bounding volume test and $n_{pr}(t, s)$, the average number of primitives at each leaf node. In Goldsmith's method, $n_{pr}(t, s) = 1$. The above equation represents the total cost. It is a simple matter to separate it into the scene cost and traversal cost, to be consistent with our earlier notation.

4 Implementation

Ray counts are in thousands

Scene	Tetra	DNA	Arches	Balls	Geo58
Objects	1024(P)	410(S)	4818(P)	7382(PS)	306(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.

We have implemented the BSP, octree, $k-d$ tree, ARTS and the Automatic Bounding Volume Hierarchy methods to test the cost model. All experiments were conducted on a MIPS 2000 running System V UNIX[1]. Five different datasets were used as test cases. Details of these datasets are given in table 1. P stands for polygons and S for spheres in table 1. Images of these models are shown in figures 8 through 11. The termination parameter used in all these methods except ARTS is the maximum height. For the ARTS method, the termination parameter is the grid resolution n_gr.

The cost model was tested as follows.

```
for each method
{
 for each scene,
 {
  for each value of the termination parameter
  {
   build the search structure
   record all information necessary to evaluate
   the cost model
   evaluate the cost model
   ray trace the scene and record run statistics
  }
  compare theoretical and experimental results.
 }
}
```

5 Experimental Results

Before we get into the results for each method, some general notes need to be made. In our implementation, each primitive is surrounded by a bounding volume which is an axis-aligned parallelepiped. Bounding

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

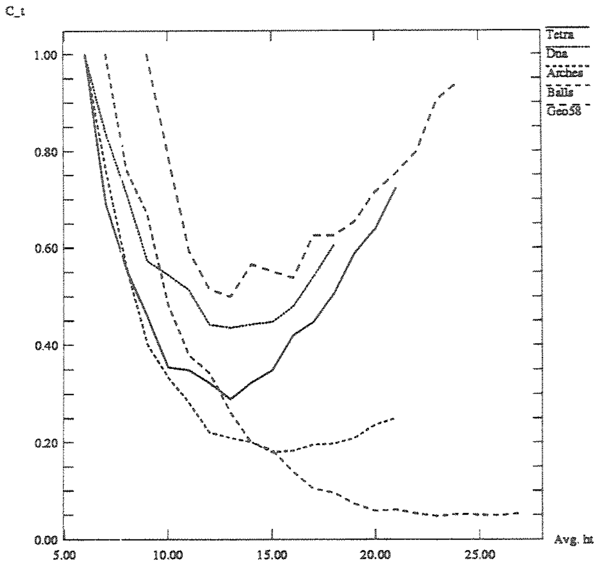

Fig. 5. BSP costs.

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. The termination parameter is the maximum height for BSP and octree methods. For ARTS, it is the grid resolution. The region probabilities are computed as explained in the previous section. The average number of primitives at each voxel is calculated by averaging the primitive object count at each of the leaf nodes of the search structure. For ARTS, all voxels are leaf nodes.

In all the methods except the automatic bounding volume hierarchy method, the ray trace stops as soon as an intersection is found. Also, duplicate object intersection tests are avoided by maintaining ray signatures in all object records. An object is intersected only if it has not been examined by the current ray.

In tables 2 through 6, $C = C_{sc} + C_{tr}$. Both theoretical(C_{th}) and experimental((C_{exp})) costs are shown. h_{th} is the predicted height, h_{exp} is the optimal height recorded by the experiments. Run times are measured in minutes.

Scene	C_{th}	C_{exp}	h_{th}	h_{exp}	$Time$(min.)
Tetra	455.37	184.98	13	9-12	1.25
DNA	249.37	202.99	13	9-12	2.83
Balls	432.45	408.99	23	20-23	25.39
Arches	1260.54	520.31	15	15	4.06
Geo58	389.3	463.30	16	16-19	8.53

Table 2. BSP tree statistics.

5.1 Applying the Cost Model

5.1.1 BSP Tree

Our implementation of the BSP tree hierarchy is very similar to Kaplan's[13]. One difference is that each subdivision step does not necessarily produce eight octants. If for example, after 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 [7] 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 need to be 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 so as to account for the work done to reach the leaf nodes.

Fig. 5 shows the cost characteristics obtained using the cost model for several scenes. Table 2 shows the results of the using the cost model to predict the height at which the subdivision has to be stopped for optimal performance. 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 most of these cases, optimal performance occurs well beyond the logarithm of the number of objects.

5.1.2 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

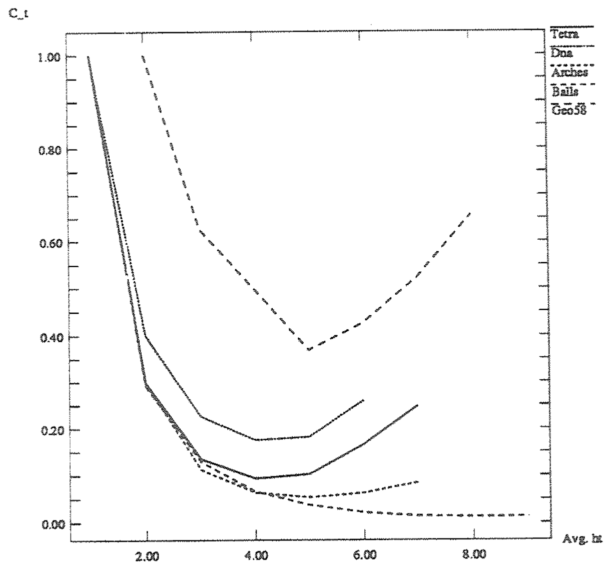

Fig. 6. Octree costs.

Scene	C_{th}	C_{exp}	h_{th}	h_{exp}	$Time$(min.)
Tetra	480.37	215.02	4	4	1.40
DNA	251.43	216.79	4	3	2.70
Balls	448.66	459.39	8	8	30.11
Arches	1243.51	544.97	5	5	4.35
Geo58	437.65	582.69	5	6-7	10.33

Table 3. Octree statistics.

method used is the same as in the BSP tree method. Fig. 6 and Table 3 show the results. As expected, the performance is slightly worse 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.

5.1.3 ARTS method

Our implementation of ARTS(uniform subdivision) follows very closely the algorithms outlined in [4]. 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. A modified form of Bresenham's line algorithm is used. 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.

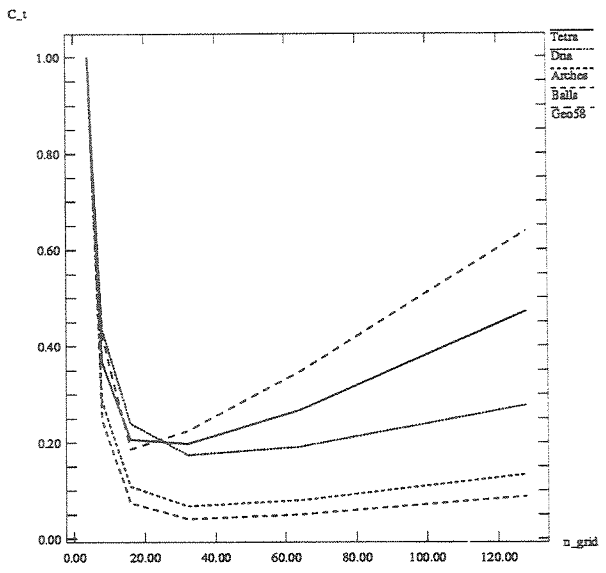

Fig. 7. ARTS costs.

Scene	C_{th}	C_{exp}	n_gr_{th}	n_gr_{exp}	Time(min.)
Tetra	269.40	215.63	32	16	1.17
DNA	93.72	156.75	32	16	2.85
Balls	404.89	1125.35	32	64	39.17
Arches	435.39	471.85	32	32-64	3.45
Geo58	138.41	1158.01	16	32-64	14.48

Table 4. ARTS statistics.

Figure 7 and Table 4 illustrate the results of this method on some typical scenes. 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 dimension each time we subdivide. The predictions are off by one level in three of these test cases. This is due to the errors incurred in computing the region probabilities. In our implementation, we used polyhedral bounding boxes[14] to enclose the primitives when clipping it to the 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.

5.1.4 $K-d$ Tree

This method also uses a binary tree structure like the BSP tree, the chief differences being the greater

Scene	C_{th}	C_{exp}	h_th	h_exp	Time(min.)
Tetra	141.11	142.20	10	10	0.90
DNA	155.74	192.06	12	12	2.19
Balls	183.57	206.08	16	16	11.43
Arches	288.36	376.44	16	16	3.44
Geo58	149.22	163.89	11	11	3.75

Table 5. K-d tree statistics.

Scene	C_{th}	C_{exp}	h_th	h_exp	Time(min.)
Tetra	358.6	376.86	10	10	1.11
DNA	881.95	1013.63	8	8	4.44
Balls	726.95	1014.72	10	10	23.64
Arches	1397.33	1543.14	10	10	6.38
Geo58	424.47	533.85	6	6	5.86

Table 6. ABV Hierarchy statistics.

flexibility in the location of the partitioning planes and the choice of the partitioning dimension, at any node of the hierarchy. It also supports bounding volumes at nodes of the hierarchy that will result in cutting down the ray search space, thus combining the advantages of space partitioning methods and bounding volume hierarchies. Details of implementation can be found in[7][15].

Fig. 9 and Table 5 illustrate the results of the cost model on different scenes. In each case, optimal performance is reached only at the height at which the original structure would not have subdivided any further. Thus, the model strives to confirm these results. One difference from the octree and BSP hierarchies is the presence of bounding volumes in the internal nodes of the hierarchy. To model this into the theoretical 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.

5.1.5 Automatic Bounding Volume Hierarchy method

We attempted to see if the height of the hierarchy affected the performance. Table 6 illustrates the results of the cost model on the same set of scenes. As in the $k-d$ tree, the cost is minimized when the tree is built with the original heuristic used by Goldsmith. Except for the Tetra scene, all the other scenes have a large number of secondary rays. Thus, to the theoretical cost we need to add the cost necessary to go down the hierarchy. This is the cost that is shown in Table 6.

6 Validating the Model

Scene	Method	Csc_{th}	Csc_{exp}
Tetra	BSP	$9928.28e^{-0.30h}$	$5473.92e^{-0.35h}$
Balls	BSP	$111311.80e^{-0.29h}$	$338605.37e^{-0.34h}$
Arches	ARTS	$22755.86e^{-0.31h}$	$17947.19e^{-0.27h}$

Table 7. Cost functions(theoretical).

Scene	Method	Ctr_{th}	Ctr_{exp}
Tetra	BSP	$17.34e^{0.19h}$	$27.92e^{0.13h}$
Balls	BSP	$19.98e^{0.11h}$	$70.41e^{0.06h}$
Arches	ARTS	$127.24e^{0.0142h}$	$100.62e^{0.0144h}$

Table 8. Cost functions(experimental).

Looking at the cost figures in tables 2 to 6 we notice that in some of these cases, the theoretical costs do not agree well with the experimentally obtained costs. However, the predicted termination point remains quite accurate. In this section, we will show that for the sake of predicting termination criteria, this is not of too much concern.

To predict an optimum subdivision level, we are interested in the rates at which C_{sc} and C_{tr} change and not so much in their absolute values. To illustrate this, refer to tables 6 and 7. Here we have obtained analytical expressions for C_{sc} and C_{tr} using an exponential least squares fit. For instance, take the first case(Tetra). If we multiply the experimental characteristics by the constant, $e^{0.055}$, we get

$$\begin{aligned} C_{scexp} &= 5473.92e^{-0.295h} \\ C_{trexp} &= 27.92e^{0.185} \end{aligned}$$

thus matching the exponents of the theoretical characteristics. Also, $(9928.28/5473.92) \approx 1/(17.34/27.92)$. Thus, the ratios of the scale constants of the respective characteristics are close reciprocals of each other, which will result in matching predictions. In general, the differences between the theoretical and experimental costs are too small to cause the predictions to be off by the required subdivision level.

7 Conclusions

We have presented a cost model which can be used for determining proper termination criteria for some of the most commonly used ray tracing hierarchies. These can be computed as the hierarchies are being built, and are dependent on the statistical characteristics of the input scene. Experimental results obtained by applying the model to the BSP, octree, ARTS, $k-d$ tree and the automatic bounding volume hierarchy methods have shown that the model provides an effective means of building automatic termination criteria into these subdivision techniques which results in optimal or near optimal performance in all cases used. 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.

8 Acknowledgements

Thanks to the MIPS Computer Corp. for the loan of their MIPS 2000 workstation. Thanks also to Sampath Rangarajan for help with the analysis. A.T.Campbell and Chris Buckalew provided thoughtful comments. Gordon Fossum created the arches model. The balls model is due to Eric Haines[10]. The architecture model(Geo58) was created by Marcos Novak.

9 Appendix

We will derive a closed form solution for R, the expected number of regions examined by a ray in terms of p, the average region probability and k, the maximum number of regions.

$$\begin{aligned}
R &= \sum_{i=0}^{k} ip(1-p)^{(i-1)} \\
&= p\sum_{i=0}^{k} i(1-p)^{(i-1)} \\
&= -p\sum_{i=0}^{k} \frac{d}{dp}(1-p)^{i} \\
&= -p\frac{d}{dp}\sum_{i=0}^{k} (1-p)^{i} \\
&= -p\frac{d}{dp}\left[\frac{1-(1-p)^{k+1}}{1-(1-p)}\right] \\
&= \frac{1}{p}\left[\{1-(1-p)^{k+1}\} - p(k+1)(1-p)^{k}\right]
\end{aligned}$$

$R \approx 1/p$ for large values of k.

References

[1] James Arvo and David Kirk. Fast ray tracing by ray classification. *Computer Graphics*, 21(4):269–278, July 1987.

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

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

[4] 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.

[5] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *Computer Graphics*, 18(3):137–145, July 1984.

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

[7] 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.

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

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

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

[11] Eric A. Haines and Donald P. Greenberg. The light buffer: A shadow testing accelerator. *IEEE Computer Graphics and Applications*, pages 6–16, September 1986.

[12] Paul S. Heckbert and Pat Hanrahan. Beam tracing polygonal objects. *Computer Graphics*, 18(3):119–127, July 1984.

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

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

[15] K.R.Subramanian. Fast ray tracing using k-d trees. Master's thesis, Department of Computer Sciences, The University of Texas at Austin, December 1987.

[16] Steven M. Rubin and Turner Whitted. A 3-dimensional representation for fast rendering of complex scenes. *Computer Graphics*, 14(3):110–116, 1980.

[17] L. Richard Speer, Tony D. DeRose, and Brian A. Barsky. A theoretical and empirical analysis of coherent ray tracing. *Graphics Interface*, pages 11–25, May 1985.

[18] L. Stone. *Theory of Optimal Search*, pages 27–28. Academic Press, New York, 1975.

[19] Hank Weghorst, Gary Hooper, and Donald P. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52–69, January 1984.

[20] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.