Automatic Bounding Volume Hierarchy Generation Using Stochastic Search Methods

Kelvin Ng
University of British Columbia
Department of Computer Science
kng@cs.ubc.ca

Borislav Trifonov
University of British Columbia
Department of Computer Science
trifonov@cs.ubc.ca

Abstract

A bounding volume hierarchy is a common method for making ray tracing based rendering more efficient. Constructing good hierarchies, however, is a difficult problem as the number of hierarchies grows exponentially with the number of scene objects. In this paper, we attempted to apply stochastic search to techniques to solve the problem of automatic generation of bounding volume hierarchy. We have investigated the use of randomized constructive search method and the evolutionary algorithm to enhance two prominent heuristics for building bounding volume hierarchies.

1 Introduction and Background

Ray casting is a rendering method whereby primary rays are cast backwards from the camera into a 3D scene; after the nearest intersection with an object is found shadow rays are cast to each light source to determine lighting. Whitted [13] extended this idea into ray tracing. Rather than just having primary and shadow rays, there can be multiple bounces of a ray (due to reflection, refraction, transmission, etc.). Since at each intersection several rays can be spawned, one gets a tree-like structure and the number of intersection tests that have to be performed increases significantly. With further extensions like distributed ray tracing [5] (where many rays are cast in a Monte Carlo sampling scheme) rendering becomes very computationally demanding. There is a large volume of work dealing with improving the performance of ray tracers. [12] Since most of the computation is spent on performing intersection tests, this has been the main target for optimization. The biggest speedups come from methods that reduce the number of intersection tests that need to be performed for each ray. The two main approaches are space subdivision (such as binary space partitioning, octrees, and regular grids) and bounding volume hierarchies. [3] The latter is the subject of this project.

The intersection computation with a geometric primitive is often an expensive operation. Enclosing the primitive in a bounding volume, most commonly a tight-fitting box or sphere, can improve the situation; intersection with a box or sphere can be computed faster, and a ray needs to be

intersected with the contained object only if it hits the bounding volume. The simplest bounding volume is an axis-aligned bounding box (AABB), which is the choice for this project. The main advantage of bounding volumes is that a tree-like hierarchy can be created, where bounding volumes contain not only objects but also other bounding volumes. A properly constructed bounding volume hierarchy changes the complexity of finding the intersections per ray from linear to logarithmic in the number of objects in the scene. Smits [12] discusses efficient traversal and optimizations such as differentiating between shadow rays (where the check is for any intersection) versus other rays (where the nearest intersection is needed).

One of the earliest ideas for automatically constructing the bounding volume hierarchy comes from Kay and Kajiya [9]. Their idea is to build the hierarchy as follows: sort the bottom level BVs along one coordinate axis, then partition them at the median; repeat by alternating the process over all three axes. This method is quite simple and fast. However, there is justification for spending more effort on optimizing a bounding volume hierarchy. A more sophisticated pre-processing step generating the hierarchy pays off in the end because practically millions of rays will traverse the tree during rendering. Goldsmith and Salmon [8] proposed an alternative method of building the BVH which takes O(n log n) time in the number of objects. Unlike the median cut algorithm, Goldsmith and Salmon's algorithm minimizes a cost heuristic based on the surface area, rather than the volume, of each bounding volume and its parents. The tree is constructed by adding objects one by one to the hierarchy: for each object, the tree is searched for a location that tries to minimize the increase in cost. The resulting hierarchy is not restricted to being a binary tree and is not balanced so that some parts of the tree are deeper than others For example, large objects tend to be higher up in the hierarchy. The overall cost of the bounding volume hierarchy can be evaluated as follows: the probability of a random ray hitting a bounding volume is proportional to its surface area; the cost of the node representing this bounding volume is the product of this number and the number of its children; the overall cost is the sum of the costs at all nodes. The cost can be computed incrementally during hierarchy construction, taking logarithmic time for the addition of one node.

More recently, Müller and Fellner [10] proposed a hybrid approach. Similarly to the Kay and Kajiya algorithm the scene is recursively subdivided; however, instead of choosing the median of sorted objects, the subdivision that optimizes a cost function similar to that of Goldsmith and Salmon is chosen at each iteration (the approach is hybrid because some nodes with uniform object distribution are not further subdivided with bounding volumes, but with regular grids).

2 Theory

To the best of our knowledge, no theoretical analysis of the problem complexity has been published or referenced. However, as shown in the following paragraph, the number of hierarchies that can be built for any scene is exponential in the number of scene objects. Furthermore, we cannot find any reference to a polynomial-time complete algorithm for this problem. In fact, all algorithms proposed in the literature are heuristic methods rather than complete ones. It is doubtful that this problem can be solved in polynomial time. The existing algorithms often claim to be near-optimal *w.r.t.*their own objective function, but this has to be a pure conjecture as no effort has been made to determine the optimal value. Practically, it is unlikely that a universal definition

of optimality exists, as the actual time spent on ray-tracing varies depending on the type of ray tracing, the type of primitives used to represent scene object, and various other implementation specific components.

To see that the number of hierarchies is exponential in the number of object. let us consider that hierarchies are constructed by recursively partitioning objects into subscenes. Without loss of generality, in each step objects divided into halves along one axis only. Assume that there are n objects in the scene. Therefore, there are $n-1$ subdivision points. It is always possible to divide the scene such that there is only a single object on one side and the remaining $n-1$ objects on the other side. Furthermore, there are two cases of this, with the single-object subscene formed at either end of the sorted object list. Of course, there are more ways to split the objects, but what will be shown here is that the number of hierarchies is already exponential in the number of objects without taking them into account. It is not necessary to divide the single-object subscene, so we consider on the other one. There are $n-1$ objects in it and therefore $n-2$ splitting points. This is exactly the same scenario as the one we encountered at the top level, except that the number of object has decreased by one. Remember that there are two cases of this. Therefore, the number of hierarchies for a scene of n objects must be greater than or equal to two times the number of hierarchies for a scene of $n-1$ objects, for $n > 2$. In other words, if $f(n)$ represents the number of hierarchies as a function of number of objects, n, then

$$f(n) \geq 2 \cdot f(n-1)$$

Equivalently, the number of hierarchies grows exponentially with the number of objects.

3 Building Bounding Volume Hierarchy by Scene Object Division

3.1 The Original Heuristic Method

In a paper that presents an algorithm that builds a hybrid data structure combining bounding volume hierarchies and uniform spatial subdivisions for a given scene, Müller and Fellner described a heuristic method to automatically generate high-quality bounded volume hierarchies. The method builds a bounding volume hierarchy recursively in a top-down fashion [10]. The hierarchies produced are virtually binary trees, with two children in each inner node. At the start of this algorithm, scene objects are sorted along all three axes by the centre position of their immediate bounding box. All the possible subdivision points are then evaluated according to an evaluation function, which is based on the surface area of their resulting children bounding volumes and the intersection costs of scene objects contained inside. The best subdivision point is then selected, and two children nodes are constructed. Each node has a bounding volume that contains objects on its side of the subdivision point along the axis. Recursively, the same method is applied to both children nodes until some stopping criteria has been met. In their paper, the stopping criteria is that the cost of intersecting the objects inside the node falls below a certain threshold value. Without the knowledge of such a good value, and for the sake of simplicity, we have decided to continue dividing objects in subscenes until all leaf nodes contain a single object.

procedure buildBVH()

1. Sort boxes along all three axes.
2. Evaluate each subdivision points on all three axes.
3. Divide the objects at the best subdivision point encountered.
4. Make the new bounding boxes of the sublists the children of the current node
5. Recursively repeat steps 2-5 on the two sublists of objects, using the same ordered obtained in step 1, until there is only one object in the sublist.

end procedure

Table 1: Pseudo-code of the Müller and Fellner Heuristic

On the average, as in any tree structure, the number of levels is O(log$n$), where N is the number of leaf nodes. In this method, the evaluation of the subdivision points can be computed incrementally in O(n) time. Overall time complexity of this algorithm is thus O($n$log$n$). The evaluation function is defined to be

$$C_H(j, axis) = \frac{S_B(L_j)}{S_B(H)} \cdot \sum_{i \in L_j} C_{obj}(o_i) + \frac{S_B(R_j)}{S_B(H)} \cdot \sum_{i \in R_j} C_{obj}(o_i),$$

where

- $C_{obj}(o)$ is the average ray/object intersection costs intersecting a random ray with elementary object o
- $S_B(X)$ is the surface area of the bounding box associated to subscene X
- $axis \in \{X, Y, Z\}$
- H is a scene hierarchy, with L_j and R_j as its direct children created by dividing the objects along the $axis$, such that $|H| = n$, $|L_j| = j$, $|R_j| = n - j$, $j \in \{1, 2, \ldots, n-1\}$. $|H|$ is the number of objects stored in the leaves of hierarchy H.

If it can be assumed that there is a uniform distribution of random rays, the probability of a ray intersecting a bounding volume B given that is it intersects a surrounding bounding volume A is $\frac{S(B)}{S(A)}$. This explains the appearance of the surface area in the evaluation function. The cost of intersecting a ray with the subscene surrounded by the bounding volume B is then estimated to be the sum of the intersection costs of individual objects of the subscene. The cost of intersecting a ray with an elementary scene object can be predetermined empirically, as explained in an electronic correspondence from one of the authors, Gordon Müller.

As a side note, although popular, binary subdivision does not always produce the best possible hierarchy. For example, if there is a large object surrounded by clusters of much small objects,

it is desirable to have its bounding box near the top level in the hierarchy. However, any binary subdivision algorithm is forced to group the large object with one of the smaller clusters in the beginning. Multi-way subdivision methods, however, is outside the scope of this project.

3.2 Implementation of the Heuristic Method

In our implementation of all bounding volume algorithms, immediate bounding boxes are built for every scene object. Operations on the objects described below are actually performed on those bounding boxes. Using immediate bounding boxes not only simplifies the implementation and reduces the computation time, it is also beneficial to the actual ray tracing, especially if intersection costs of the scene objects are high.

To efficiently compute the evaluation function values or costs for all the subdivision points along an axis, one can scan the objects (or immediate bounding volume of the objects) from one end to another, extending the bounding box along the way by adding objects one by one and recording their costs. Then, repeat the same procedure but reverse the order of scanning. In addition, to find the total cost of a subdivision point, one has to add the costs found in the two passes. As expanding a bounding volume to include another one is a constant time operation, the overall time needed to determine the best subdivision point along an axis is O(n). Also, the sorting of the objects has to be performed once only in the beginning to preserve the O(n log n) time complexity of the overall method.

As mentioned previously, we allow subdivision to continue until all the leaf nodes contain a single object. As a result, we have noticed room for optimization on the evaluation function without sacrificing the quality of hierarchies generated by this method. Since the parent node is the same for all subdivision points, and the only purpose of the cost function is to compare the quality of those different subdivision points, it is not necessary to divide the surface areas of the children bounding volumes by that of the parent one. This alters the actual value of the cost function but preserves the relative ordering of hierarchies, which is all we need to know. By the same token, when computing the surface area of the bounding volumes, it is not necessary to multiply the area of the surfaces of the three different orientations by two. Both optimizations remove unnecessary operations that are repeated frequently.

Furthermore, even though there are three different axes, it is possible that the ordering is the same along two or more axes. To checking for same ordering, however, is a O(n) operation itself, albeit being considerably less expensive than evaluating the subdivision points. An important point is that if in a certain node it has been determined that the ordering is identical on two axes, the same can be concluded on all subsequent "offspring" nodes generated by it. Therefore, it is believed that to be advantageous to check for identical ordering along the three axes. Indeed, a quick experiment on two benchmark instances, the largest one Cloister and the smallest one escher, showed that run time is slightly reduced when this technique is applied. The real CPU time consumed dropped from 2.469s to 2.422s on Cloister, and from 0.266s to 0.2665s on escher. As expected, the advantage becomes more significant as the problem size grows,

Instance	Original	New
bijou	1.710e7	2.840e7
Cloister	1.845e9	2.632e9
escher	4.8195e5	7.335e5
konistil	1.027e14	2.137e14
pinecon2	1.361e6	1.651e6

Table 2: Comparison of bounding volume hierarchy costs obtained using a different cost function with those using the original one

3.3 Evaluation of Hierarchy Quality

Once an hierarchy has been built, it is possible to objectively evaluate its quality. If the assumption of uniform random ray distribution can be made, then the cost of a hierarchy can be calculated using the formula: $\sum_{i \in nodes} P_i \cdot Cost_i$, where P_i is the probability that a ray is intersecting with a node, and $Cost_i$ is the cost of intersection tests with its children. Intuitively, this is the sum of products of intersection cost and the probability that such intersection test is required for all nodes.

As the set of scene objects as well as their immediate bounding boxes is the same for all possible hierarchies built for them, the associated intersection cost is fixed and therefore can be excluded when comparing hierarchies. Furthermore, it is a fair assumption that the cost of intersecting a bounding volume in the hierarchy, which is an axis-aligned bounding box, is also fixed regardless of size and location. Taking into account the above assumptions, and the fact that the probability of intersection is proportional to the surface area, we can use the total surface area of all internal bounding volumes to measure the quality of a given bounding volume hierarchy.

A different yet simpler heuristic function has been tested but did not produce good hierarchies. The results can be found in table 2. It is a simple function that only considers the surface area of the subscene bounding volumes, ignoring the number of objects contained inside them. This can be viewed as a naive attempt to apply the objective function directly as the heuristic in the construction steps. This shows why the objective function may not always be a good choice for evaluation function, especially in the constructive search case, where the solution components are likely to be dependent on each other.

3.4 Applying Stochastic Local Search

This heuristic of building a hierarchical data structure for scene objects can be modelled as a constructive search method. It builds the hierarchy by dividing the objects into two halves at each point, adding levels to the hierarchy by recursively deciding the subscenes. This corresponds to building the solution by adding solution components one by one. The selection of the subdivision point at each step is guided by a heuristic function, and the best selection encountered is always selected. The heuristic function, which can also be called the evaluation function, aims at estimating the actual quality of possible choices. It differs from the final objective function that solution quality, which is the total surface areas of all internal bounding volumes. Moreover, the heuris-

Instance	Base	Rand 10	Rand 100
bijou	1.710e7	2.028e7	1.963e7
Cloister	1.845e9	2.013e9	1.978e9
escher	4.8195e5	5.3575e5	5.2116e5
konistil	1.027e14	1.188e14	1.111e14
pinecon2	1.361e6	1.587e6	1.513e6

Table 3: Comparison of bounding volume hierarchy costs by running the first randomized method with those by the base method. Rand 10 and Rand 100 refer to the best hierarchy cost obtained in 10 and 100 trials respectively.

tic function is adaptive, because its value depends on the solution components (other subdivision points) already present in the partially built hierarchy.

The above description of the heuristic fits into the model of a greedy constructive search method, as introduced in the textbook by Hoos and Stützle [11]. The leads to the conjecture that an improvement can be achieved by randomizing the search, inspired by the success obtained from applying the Greedy Randomized Adaptive Search Procedure (GRASP) [6, 7]. In the randomized construction phase of GRASP, the solution component with the optimal heuristic value is not automatically selected in each step. Instead, the component is randomly selected from a set of highly ranked solution components contained in a restricted candidate list (RCL). The RCL can be either defined by cardinality or value restriction. In the first case, the k best ranked solutions are included, while in the second case, the components within a factor of α of the best heuristic value is included. The second phase of GRASP is a perturbative local search phase, which is briefly discussed in section 5.

In our implementation of the base method, a function is invoked to find the best subdivision point along a particular axis in each construction step. The subdivision point with the best value is then chosen from the best ones, found along each of the three axes. Therefore, an obvious and straightforward way to extend the base method is to randomly pick from those three subdivision points. Unfortunately, this simple modification does not perform nearly as well as the original method, and failed to improve the solution quality on all of the benchmark instances even after a relatively large number of attempts (100). Table 3 shows the costs of hierarchies obtained on the test scenes by running this method.

Without any subsequent processing to improve the resulting hierarchies, the frequent deviation from the best point (two thirds of the time) generates poorly constructed trees. The problem is aggravated by the fact that other two points could be among lower ranked points when considering all possible ones along the three axes.

As a result, an adjustable noise setting is added to allow different probabilities of not picking the point with the best heuristic value. With a small noise setting of around 10 percent, randomized construction search occasionally produces better quality hierarchies than the greedy one, and the majority of them stay within five percent of the one built greedily. Table 4 contains the experimental results. It is interesting to see that in two instances (konistil and pinecon2), an improvement over the original method can be observed after a small number of attempts (10). While the im-

Instance	Base	Rand 10	Rand 100
bijou	1.710e7	1.713e7	**1.699e7**
Cloister	1.845e9	1.850e9	**1.828e9**
escher	4.820e5	4.837e5	**4.733e5**
konistil	1.027e14	**1023e14**	**1.023e14**
pinecon2	1.361e6	**1.360e6**	**1.359e6**

Table 4: Comparison of bounding volume hierarchy costs by running the first randomized method with a noise setting of 10 percent with those by the base method. Rand 10 and Rand 100 refer to the best hierarchy cost obtained in 10 and 100 trials respectively.

provement obtained may be small, it is always achieved when given enough number of trials.

A further attempt to enhance the performance is made by lifting the restriction of considering only the best points from the three axes. By modifying the program to store extra information, one can add flexibility by adopting an approach more similar to the one described in GRASP. In this case, the restricted candidate list is built following the cardinality restriction, consisting of the best k ranked subdivision points among all possible ones. The parameter k, along with the noise setting, can then be tuned to optimize the performance. Preliminary findings indicate that there is no apparent advantage of this approach. With a small k and a small noise setting, the performance is comparable to the previous version. Larger values of the parameters generally result in worse hierarchies. Furthermore, the computation resources required by this method is significantly greater than the previous one, making it an unattractive choice. It should be no surprise as the time complexity of this algorithm is k times that of the previous one. Figure 1 is a plot of costs vs number of trials for different values of k. It shows that the performance is optimal when k is 5.

4 Enhancing the Object Insertion Method with Evolutionary Approach

In [4] the authors proposed the use of evolutionary techniques for the automatic generation of near-optimal partitioning trees, which is another approach for speeding up rendering. The idea is that once partitioning planes are determined, the trees only differ in the order in which partitions are chosen; each tree can be specified by a permutation of the list of partitions. Populations of permutations can be created and mutation, crossover, and selection can be used, trying to optimize them.

Originally the idea adopted here was to use such an approach for choosing the split points for a top-down subdivision (Müller and Fellner's [10] is top-down, for example). Only the order of the split points matters, since all are chosen – however, that only applies to the one dimensional version of the problem. Eventually it was realized that there was no way to do this with three axes, since $n - 1$ split points have to be chosen out of $3(n - 1)$ possible over the three axes, and the

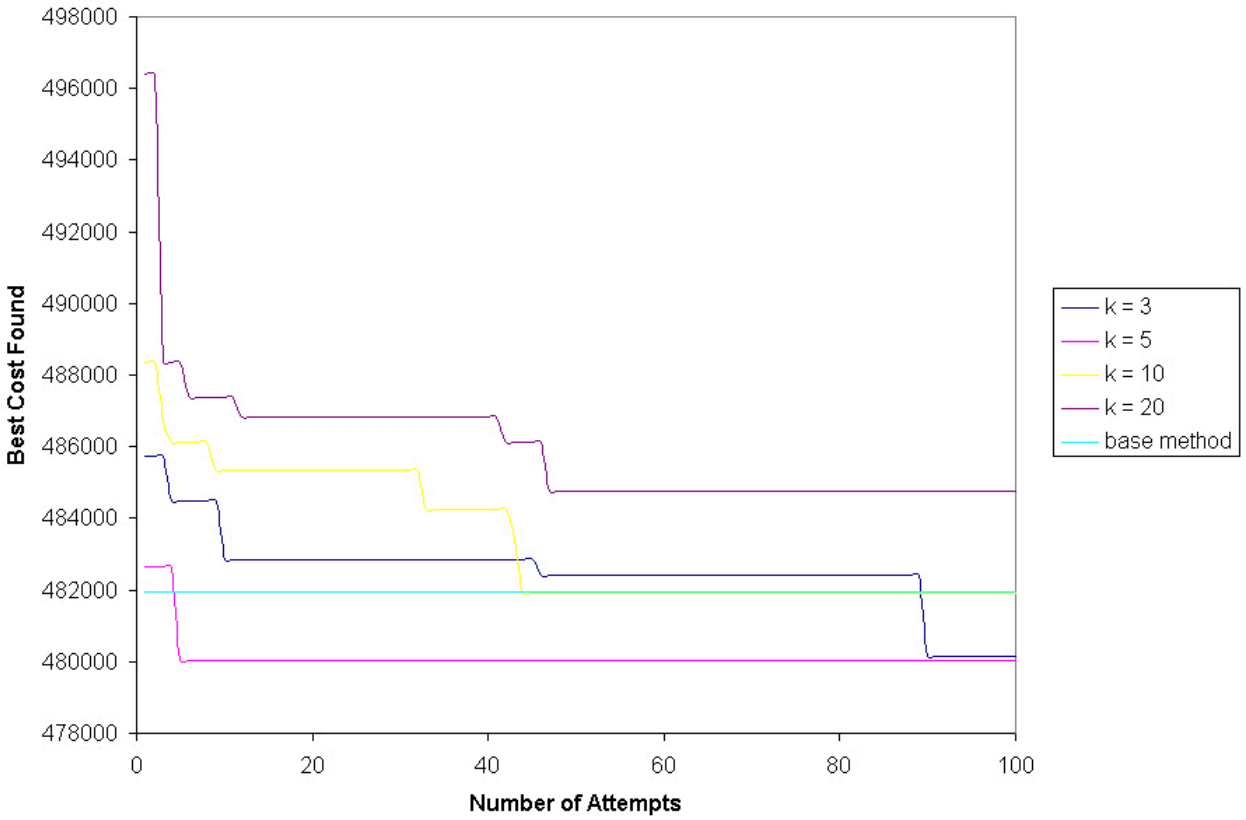

Figure 1: Costs vs number of trials on escher for various k.

choosing has to be done in such a way as to avoid invalid permutations (where two split points each on a different axis result in the exact same split between volumes; these cannot be detected since they change depending on the particular order). Most permutations generated using a genetic algorithm would be invalid, and it would be inefficient to test for that.

Next this idea was applied to Goldsmith and Salmon's algorithm, which builds the BVH by adding volumes one by one, and depends on the order of insertions. The following is a more detailed description of Goldsmith and Salmon's method. The first node to be inserted forms the root of the tree. The tree is then searched for an insertion location for the next node. It can either become a child or sibling of a node. The tree is searched top down without backtracking, every time choosing a subtree to search that will have a minimal increase in cost if the volume were to be added to that subtree. Cost increase depends on the surface area, increase in surface area in case of addition, and number of children, for the node and it's ancestors; for the exact algorithm, see [1], which includes the C++ source code, makefile, and one test scene (for gcc 3.2 or higher). This process takes approximately $log(n)$ time for each of the n volumes to be added.

Some discussion on testing is appropriate. The cost values that Goldsmith and Salmon's algorithm returned were different than those obtained by simulating actual ray intersections; however, this is to be expected, because here shadow rays were tested rather than primary rays (most rays in scenes are usually shadow rays), and depend on the distribution of the rays (here random rays with origins within the root scene volume were used; in actual ray tracing there is the additional restriction that shadow ray origins lie on either surfaces or lights, depending on implementation; only intersections along the direction of the ray after the ray origin, not behind it, are considered). The differences varied across scenes. However, the correlation between this heuristic cost function and the cost from simulations (for various orderings of input for the same scene) is very strong (almost proportional). Simulations of 10000 rays were used in each case; different simulations on the same bounding volume hierarchy gave average numbers of intersections that were the same to two significant digits. Unfortunately, using this many rays takes longer than building a hierarchy. The intention was to use actual simulation instead of the cost function for evaluating fitness, but using a small number of rays gave unstable evaluations and poor performance. Thus, the Goldsmith and Salmon cost function was used as a fitness measure, which is not much of a sacrifice due to the good correlation it has with the empirically determined costs.

The mutation, crossover, and selection methods are the same as those proposed by [4]. In a population of permutations, new permutations are created by mutation and crossover as follows. Mutation is simply a swap of two randomly chosen elements in a permutation. Due to the large number of objects in the scene (and thus length of the permutations), multiple mutations are possible at a time (a number proportional to the length). Crossover is accomplished by choosing two parents and a crossover point randomly; for the first offspring, elements are copied from the beginning of the first parent up to the crossover point, and then the rest from the second parent in a way that avoids copying duplicate elements and results in a valid permutation (see [1]). The second offspring is created in the same way, but starting with the other parent. The offspring replace the parents, and thus population size is constant. The selection is tournament style, whereby random pairs of permutations are chosen (as many times as is the size of the population), and the one with the lower cost (higher fitness) is kept for the next generation (additionally, the most fit is always kept; due to this selection, duplicates are possible). This approach was chosen here because it is simple, more memory efficient, and because it was used in [4] for a related problem. The fact that

in the case of high crossover probability most permutations will be replaced by their children even if they are less fit than the parents should not be a significant problem: the fittest permutation is always kept in the population and memorized as the best solution seen so far, and permutations with high fitness can be chosen more than once by the random tournament selection, resulting in duplicates.

In their experiments, [4] tested the algorithm with populations sized several times the length of the permutations, for several tens of thousands of generations, resulting in runtimes of hours. The justification of such long runtimes is that the acceleration data structure, once optimized, can be stored with the scene, and used to speed up many renderings, even though one rendering might only take minutes. However, in their case the calculation of the evaluation function can be done faster than here, where each permutation is evaluated by generating the BVH with a full run of the Goldsmith and Salmon algorithm. Most testing was done with two scenes obtained from the public domain of around 10000 objects (triangles) each; these can be found in pinecon2.zip and escher.zip from http://web.ukonline.co.uk/Members/bebop/meshes/meshes.htm. This algorithm appears to have a somewhat limited sensitivity to the ordering. Using subsets of the scenes of around 1000 objects, the evolutionary approach gave essentially no gains for runs with population sizes up to 100 and 200 generations. In the case of the whole scenes, due to the very large computation time for the evaluation of each permutation of length around 10000, small population sizes and number of generations were used (this is also why larger scenes were not tested). In figures 2 and 3 (on page 158) population sizes of 50 and 100 generations were used, taking several minutes each on the departmental Unix server. The value graphed is the Goldsmith and Salmon BVH optimality cost function returned by their algorithm for the most fit tree (see the description of that algorithm in section 1). The empirical costs from simulation of ray intersections for the most fit members of the last generation in each run were 33 and 27 average number of intersections per ray. It is interesting to note that these are similar numbers for two similarly sized scenes; Müller and Fellner's algorithm produces a value of 26 for the first scene, but a value of 12 for the second one. Clearly Müller and Fellner's approach has significantly better performance in optimization, and ran in less than a minute on the same machine.

It is worth considering why evolutionary techniques only resulted of improvements of about a third over the average Goldsmith and Salmon case, which was at least better than random picking of permutations (which was also tried), but not enough. Varying the mutation and crossover rates was attempted. Due to the long time for each run, a pick-and-test method was used for a number of values, rather than some exhaustive search. The aforementioned graphs are those for the best of the values tested, with a crossover rate of 0.7 and a mutation rate averaging around 1/500th of the length of a permutation. Graphs produced with other values, however, were very similar, with the worst case being no mutation at all and producing cost values about one third higher for runs of the same population size and number of generations. Doing five runs (only five due to the speed issue) with those parameters for each scene gave final costs within intervals of 30 and 38 for the first scene, and 26 and 30 for the second (the graphs in this paper show the median runs). It appears that the variance of the performance of the underlying Goldsmith and Salmon algorithm over the various possible orderings is quite low; i.e. Goldsmith and Salmon's method has only limited sensitivity to the ordering of objects in the majority of cases (this was also observed when testing random picking of permutations, where the majority of orderings had similar cost values, with some significantly worse, and few with higher ones). Another possible factor is that the specific

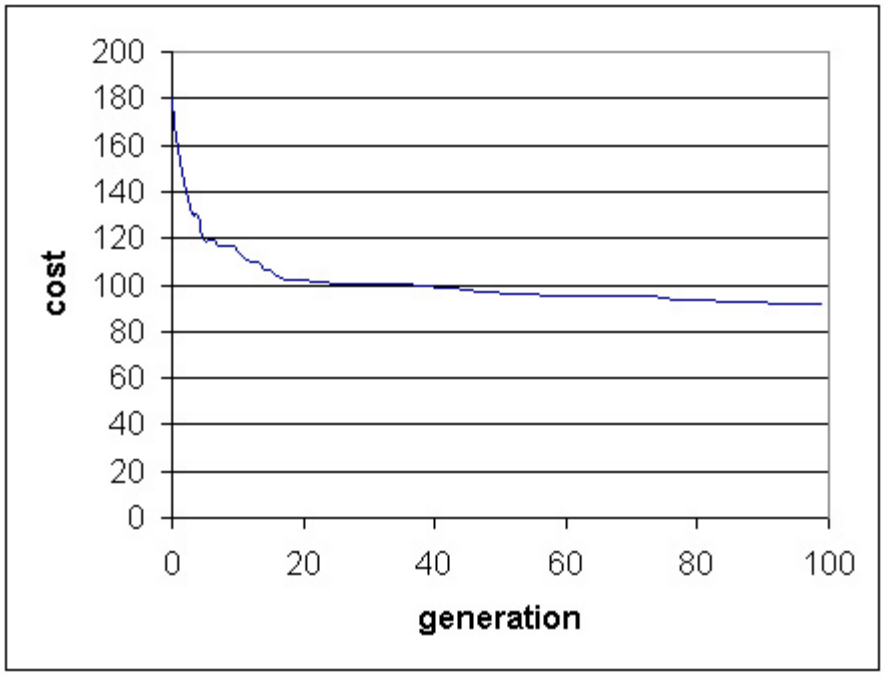

Figure 2: Cost vs generation (population=50) on pinecon2.

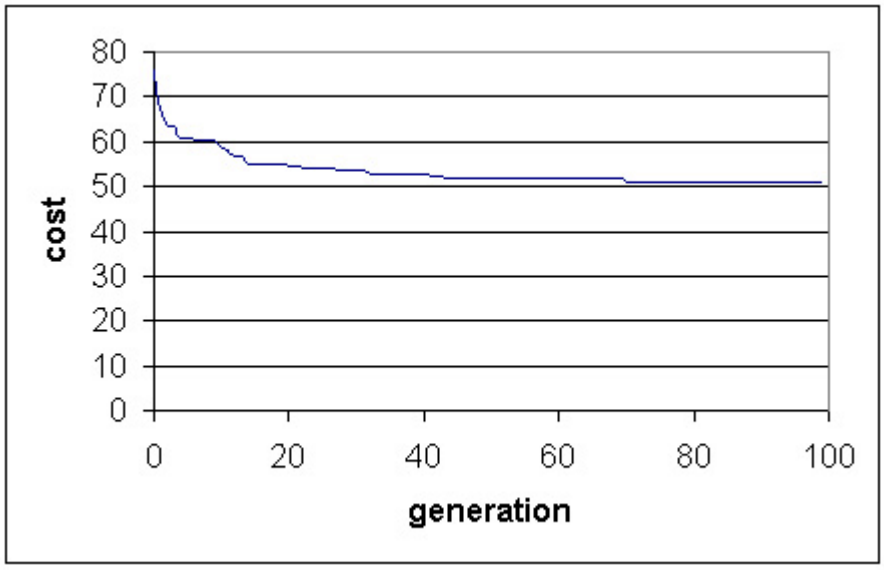

Figure 3: Cost vs generation (population=50) on escher.

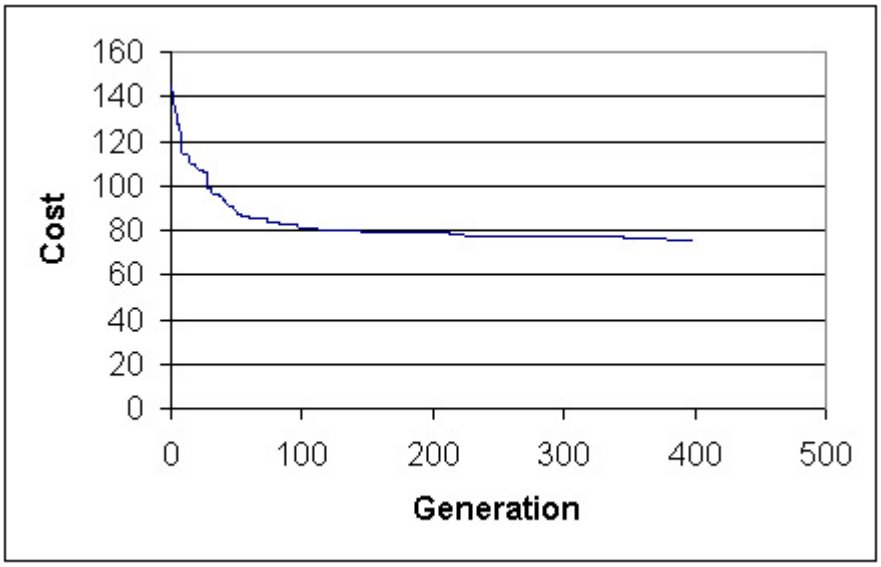

Figure 4: Cost vs generation (population=200, long run) on pinecon2.

crossover, mutation, and selection methods have too much difficulty getting out of local minima, as evidenced by the apparent stagnation seen in the graphs. Trying to increase diversification by having larger mutation and/or crossover rates than the ones used above, however, produced virtually identical graphs. Finally, a large test was ran with a population of 200 for 400 generations for the pinecon2 scene to see whether the algorithm continues to slowly improve the cost value or completely stagnates. This took about an hour on a dual Xeon machine in the β lab, with results shown in figure 4. Some improvement can be seen in the cost function as compared to the shorter runs, but in the empirical testing with random rays the value was actually worse (40). This fits with the observation that the Goldsmith and Salmon cost function is just an approximation; nevertheless, using the ray testing as a cost function instead is impractical, as getting stable values (no variance in the first few significant digits of the values returned by several tests of the same BVH) required the use of hundreds of thousands of rays; this would increase the computation time many times, as the ray tracing evaluation would take much longer than the actual Goldsmith and Salmon procedure, the current bottleneck.

5 Future Work

All the results from the randomized constructive search experience in section 3 seems to indicate that a good local improvement or perturbative local search phase could potentially be the missing piece of the puzzle. However, there is no indication of how to meaningfully encode the hierarchy and select a good neighbourhood that will produce good results when local improvement techniques are applied. One may wish to view that a hierarchy as a permutation of the subdivision points. Unfortunately, not all subdivision points are needed due to the existence of three axes.

Furthermore, even in the one axis case, there could be different distinct permutations that generate identical hierarchies.

Another suggestion is to re-examine the hierarchies after they have been built. If at some node it appears that a wrong decision has been made so that the two children are highly imbalance and one is contributing significantly to the overall cost of the hierarchy, a different selection point can be made and this part of the hierarchy resembling a subtree can be rebuilt. This approach, however, requires more information to be stored at each node, and it is not a local search step strictly speaking. Despite that, it is still a worthwhile attempt in the future.

6 Conclusion

Good data structures for storing and organizing scene objects are crucial in the ray-tracing based rendering. We have attempted to apply stochastic search techniques to enhance existing heuristic methods that are generally agreed to produce good quality bounding volume hierarchies. A approach based on Greedy Randomized Adaptive Search Procedures (GRASP) is applied to improve upon Müller and Feller's object division construction method. An evolutionary algorithm is employed in the attempt to obtain a good object ordering for Goldsmith and Salmon's object insertion-based construction method. The tested approaches could only provide marginal improvements over the original algorithms at the expense of increased computation time. Furthermore investigation can be performed to study other possible ways to improve upon existing results found in this project.

References

[1] Borislav Trifonov.: Source code for evolutionary approach. http://www.cs.ubc.ca/ trifonov/gsg.tar.gz.

[2] Müller, G. and Ng, Kelvin.: Personal electronic mail correspondences, dated Mar 19-24, 2003. http://www.cs.ubc.ca/ kng/mueller-email.txt.

[3] Chang, A. Y. A survey of geometric data structures for ray tracing. Technical Report TR-CIS-2001-06, CIS Department, Polytechnic University, 2001.

[4] Cassen, T., Michalewicz, Z., and Subramanian, K. R.: Near-optimal construction of partitioning trees by evolutionary techniques. Graphics Interface, 1995.

[5] R. L. Cook, T. Porter, and L. Carpenter.: Distributed ray racing, Computer Graphics, vol. 18, no. 3, pp 137-145, 1984.

[6] Feo, T.A. and Resende, M.G.C.: A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters, 8:67-71, 1989.

[7] Feo, T.A. and Resende, M.G.C.: Greedy randomized adaptive search procedures. Journal of Global Optimization, 6:109-133, 1995.

[8] Goldsmith, J. and Salmon, J.: Automatic creation of object hierarchies for ray tracing. IEEE Computer Graphics and Applications, 7(5):14-20, May 1987.

[9] Kay, T.L. and Kajiya, J. T. Ray tracing complex scenes. Computer Graphics (SIGGRAPH '86 Proceedings), 20(4):269-78, Aug. 1986.

[10] Müller, G. and Fellner, D. W.: Hybrid scene structuring with application to ray tracing. Proceedings of the International Conference on Visual Computing (ICVC'99), 19-26, February 1999.

[11] Hoos, H. H. and Stützle, T.: Stochastic local search: foundations and applications. Morgan Kaufmann Publishers, to appear in 2003.

[12] Smits, Brian. Efficiency issues for ray tracing. Journal of Graphics Tools, 3(2):1-14, 1998.

[13] Whitted, T. An improved illumination model for shaded display. Communications of the ACM, 23(6):343-349, June 1980.