Tree Rotations for Improving Bounding Volume Hierarchies

Andrew Kensler

SCI Institute, University of Utah

ABSTRACT

Current top-down algorithms for constructing bounding volume hierarchies (BVHs) using the surface area heuristic (SAH) rely on an estimate of the cost of the potential subtrees to determine how to partition the primitives. After a tree has been fully built, however, the true cost value at each node can be computed. We present two related algorithms that use this information to reduce the tree's total cost through a series of local adjustments (tree rotations) to its structure. The first algorithm uses a fast and simple hill climbing method and the second uses simulated annealing to obtain greater improvements by avoiding local minima. Both algorithms are easy to add to existing BVH implementations and are suitable for pre-processing static geometry for interactive ray tracing.

Keywords: Ray tracing, acceleration structures, bounding volume hierarchies, tree rotations.

1 INTRODUCTION

Algorithms based on the surface area heuristic (SAH) [6] currently represent the state of the art in the construction of kd-trees [7] and bounding volume hierarchies (BVHs) [11, 9] for accelerating ray tracing. This heuristic provides a cost model for the average time that a ray will take to traverse down the tree and intersect candidate primitives at the leaf notes. In its basic form as used for binary tree BVHs constructed over triangle meshes, it says that the cost, C, of a node is

$$C = \begin{cases} C_t + C_i N & \text{for leaf nodes} \\ C_t + \frac{S_L C_L + S_R C_R}{S} & \text{for interior nodes,} \end{cases}$$

where C_t and C_i, respectively, represent the relative costs for a traversal step and for a primitive intersection step in the rendering implementation and N is the number of primitives at a leaf node. S_L, S_R, S give the respective surface areas of the bounding volumes of the left child, right child and current node. Similarly, C_L and C_R give the computed costs for the left and right children.

The SAH algorithm for constructing BVHs proceeds in top-down fashion. At each node it considers a set of candidate partitionings of the primitives and greedily chooses the one that minimizes its estimate of the node's cost. Typically, "exact" SAH algorithms form these candidate partitionings by using the planes from each of the six sides of each primitive's axis-aligned bounding box. These partitionings appoint each primitive to a group based on which side of the plane the primitives' centroid falls on. Once the best partition has been found, these groups become the current node's children and then the process continues recursively down until the number of primitives assigned to a node falls below some threshold at which point that node becomes a leaf.

Approximate SAH construction algorithms for BVHs [15] work similarly, except that they use binning to sample a smaller number of potential partitions—often by simply choosing axis-aligned planes at regular intervals rather than the sides of the primitives'

bounding boxes. From these, they fit a simple curve to the cost estimates and use the curve's minimum to choose the best plane along which to partition the primitives. Thus, the approximate algorithms attempt to build a tree with a quality close to those of the exact algorithms' but in a fraction of the time.

One notable characteristic of all top-down construction algorithms based on the SAH is that they are not minimizing the true value of the C but only an estimate because C_L and C_R can only be computed with certainty after their corresponding subtrees have been fully constructed. Instead, they estimate C_L and C_R either linearly or logarithmically from the number of primitives potentially assigned to each child.

After fully constructing the tree, however, one can compute the true cost value both for the tree as a whole and for each node. With this additional information it may be possible to produce an improved tree with a lower cost. Ng and Trifonov [12] partially explored stochastic algorithms for this by repeatedly building BVH trees with jittered splitting plane locations, and then preserving the tree that yields the lowest true cost. This method failed to produce better trees than the standard greedy algorithm. Inspired by the use of genetic algorithms for constructing BSP trees in [5], they also explored extending Goldsmith and Salmon's [6] bottom-up BVH construction method with a genetic algorithm to optimize the order of primitive insertion. Despite the improvements yielded by the genetic algorithm, they found that the standard top-down greedy build algorithm still produced superior trees.

In this paper we explore an alternate method based on starting with an already existing BVH such as one produced by the usual top-down greedy algorithm and then using local tree rotations—similar to those used for balancing binary search trees—and hill climbing to improve it. Then we show how to combine this with stochastic global optimization via a simulated annealing algorithm [8, 14] to attempt to avoid local minima. We conclude with a description of our implementation and some remarks on the results obtained from it.

2 TREE ROTATIONS

Self-balancing binary search trees such as AVL [1] and red-black [3] trees use tree rotations as the fundamental operation for re-balancing the tree after an insertion or deletion. Splay trees [13] also use tree rotations, but apply them after a lookup in order to improve future access times when it needs that element again soon.

Tree rotations are local operations involving the root of a subtree and its immediate children and grandchildren and come in two symmetric forms: left-rotations and right-rotations. By altering the linkage of nodes within the tree, one of the children moves up to take the place of the root of the subtree, while the original root descends to become a child of the new root. This is shown in Figure 1. Which child ascends and which descends depends on whether a left-rotation or a right-rotation is being applied. In either case, a rotation will undo the effects of its opposite. Furthermore, rotations always preserve the search ordering of the original tree and it is possible to transform any two binary search trees with the same nodes into the each other through a sequence of rotations.

We can apply a similar idea to modifying BVHs. Because the BVH has no order property and the interior nodes carry no inherent information of their own—only the links to their children and
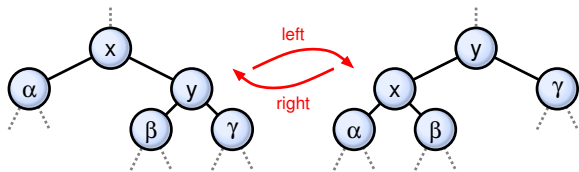
Figure 1: Left- and right-rotations on ordered binary trees

the union of their children's bounding volumes—the rotation operations are slightly different, however. The rotations reduce to simply swapping a child with a grandchild from the other side. Figure 2 shows the four possible pairs to exchange. As with the rotations for binary search trees these result in elevating one subtree, rooted at the grandchild, while demoting another, rooted at the child.
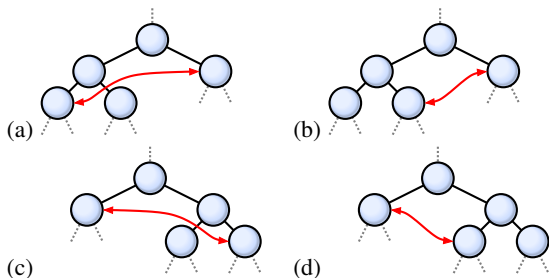

Figure 2: Nodes to exchange for each possible rotation

Note that these exchanges do not affect the bounding volume at the subtree's root—only its children's bounding volumes will change. Because of this, an exchange will not affect the bounding volume of any node in the tree above the subtree's root and so the denominators in the computations of the SAH costs for these upper nodes will remain unchanged. Consequently, any rotation that increases or decreases the subtree root's cost must produce a corresponding, monotonic change to the global cost of the BVH for the entire scene. Applying a rotation that improves the subtree's cost will always improve the scene's global cost and choosing the rotation that produces the greatest reduction to the local cost will produce the greatest reduction to the global cost. Moreover, computing the local effect of a rotation is a constant-time operation if the costs and surface areas of the children and grandchildren are available.

3 HILL CLIMBING

This leads to a very straightforward, efficient, hill climbing algorithm for improving the cost of a BVH after it has been constructed: recurse over the tree and visit each node. If the node is a leaf node, simply compute its cost and return. If the node is an interior node, visit its children first and then recompute its current cost. Next, determine which rotations the node is eligible to be the root for and compute the new costs that the node would have if the rotations were applied. If any of them yield an improvement over the current cost, then apply the rotation that produces the greatest improvement, update the node's cost and return. Otherwise, leave the node unchanged and return to the parent. Repeat these passes through the tree until no new beneficial rotations can be found.

One improvement that we have found beneficial is to allow for direct swaps between the grandchildren of a node. While not true rotations, the procedure for updating the tree and computing the effect of these swaps on the node's cost is nearly identical to that for true rotations. Though a sequence of rotations can produce the same effect, the intermediate steps may temporarily increase the tree's cost and thus the hill climbing algorithm would overlook them.

Figure 3 demonstrates this algorithm applied to the standard conference room scene. Starting with the initial global SAH cost after the tree's construction, each pass of our algorithm progressively lowers the cost until it cannot reduce it any further.
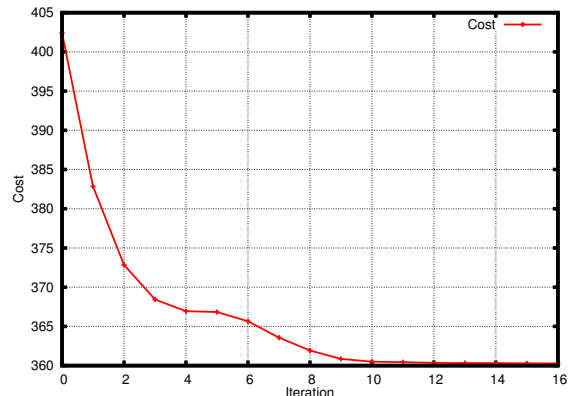

Figure 3: Progressive lowering of SAH cost for conference room

4 SIMULATED ANNEALING

While the algorithm described above will never increase the global SAH cost of a BVH it tends to quickly settle into local minima. To remedy this we used the simulated annealing algorithm [8, 14], a variant of the Metropolis algorithm [10]. Inspired by the annealing methods for growing large crystals, simulated annealing uses the concept of a global temperature to control moves that worsen the solution but also allow it to escape the minima.

The algorithm requires several pieces: an energy function to compute the energy of a particular state, a way to propose candidate transitions to neighboring states, an acceptance probability function, and an annealing (cooling) schedule that defines how the temperature changes over time. The algorithm works by repeatedly proposing changes that the energy function measures the effect of. It always accepts changes that decrease the energy but for changes that increase the energy, the acceptance probability function uses the measured difference in energy and the current temperature from the annealing schedule to compute a probability. It accepts the proposed change if a random number in $[0, 1)$ is less than this probability and rejects them otherwise. After each iteration, it adjusts the current temperature according to the annealing schedule. Finally, there is often a "quench" phase after the annealing schedule ends. This phase moves to the local minimum by only accepting changes that reduce the systems energy.

Mapping this to the BVH tree rotation algorithm is straightforward. The SAH metric provides the energy function, while the set of rotations possible at each node provides the proposed state transitions. For the acceptance probability function, we use the standard Boltzmann factor used by most simulated annealing implementations,

$$P(\Delta e, T) = \begin{cases} \min(e^{-\Delta e/T}, 1) & T > 0 \\ 0 & T = 0, \end{cases}$$

where Δe is the change in energy produced by the proposed change and T is the current temperature. Through experimentation we have found that a clamped and linearly ramped sine function makes a reasonably good annealing schedule. We used $T(i) = \max(0, -\sin(i2\pi/f))(N-i)h/N$, where i is the current iteration over the tree, f is the sine function's frequency, h is the hottest temperature allowed, and N is the schedule's length, defined as the total number of iterations to run. In our tests, we used $N = 1250$, $f = 50$ and $0.8 \leq h \leq 2.2$, depending on the scene.

To implement this, we start with the previous hill climbing algorithm that recurses over the tree and at each node test each possible rotation and swap to determine which one lowers the local cost of the node the most. However while trying to find the minimum if a proposed exchange would increase the node's local cost relative to the currently best found exchange then instead of outright rejecting it as before, we pick a random number ξ in $[0,1)$ and test if $P(\Delta e, T) < \xi$ and use the difference in local cost as Δe. If this test succeeds then we accept this as the currently best found exchange anyway. After all valid rotations and swaps around a node have been tested this way, we apply the best one, if any. After the recursion has finished processing all the nodes this way, we update the temperature for the next pass and begin walking the tree again. Because we define $P(\Delta e, T)$ to always be zero when $T = 0$, the simulated annealing algorithm reduces to the original hill climbing algorithm for the phases of the annealing schedule when $T = 0$. At the end of the schedule, we quench the system by leaving $T = 0$ and continue to make passes until the global cost of the BVH tree converges to a local minimum. Optionally, we can retain a copy of the best tree found as the optimization progresses and replace it whenever a pass ends with an improvement to it, then return that tree as the simulated annealing algorithm's result. If the annealing schedule starts out with $T = 0$ for enough iterations with this option then in the worst case the simulated annealing algorithm will at least match the hill climbing algorithm.

Figure 4 shows how the simulated annealing algorithm reduces the global SAH cost for the Soda Hall scene. The annealing schedule begins with a phase where the temperature is zero and so it will initially lower the global cost for the scene as with the hill climbing algorithm before. After a few iterations, the temperature rises and the algorithm makes changes to the tree that result in increasing the tree's cost. By cycling through these heating and cooling phases while gradually diminishing the strength of the heating phases, the BVH for the scene is brought to a lower global SAH cost than would be possible with the hill climbing algorithm alone (left plateau on the graph.)
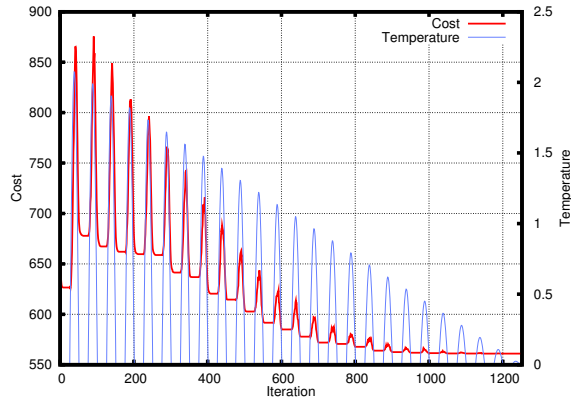

Figure 4: Simulated annealing applied to the Soda Hall model

Note that for any two trees with the same leaves and the same number of interior nodes, there is a sequence of rotations that will transform one into the other. Because the number of leaf nodes determines the number of interior nodes, the best possible BVH (as defined by global SAH cost) for a given set of leaves is always reachable from any other state. With this property, the probability of the simulated annealing algorithm coming across the best possible BVH approaches one as the annealing schedule grows [2]. While this will require an infeasible amount of computation in practice, the algorithm is nonetheless theoretically sound.

5 RESULTS

We have implemented our optimization algorithm in the Manta interactive ray tracer [4]. The BVH code in Manta evolved from a port of the code from the DynBVH system [16]. We have kept the existing BVH construction and traversal code otherwise unmodified and simply hooked in our tree optimization to run after the BVH's initial construction. For all tests, we used the exact SAH build algorithm rather than the approximate binning algorithm.

For interior nodes, the BVH traversal code tries to improve the chances of early ray termination by choosing which child of an interior node to process first based on the split axis recorded for that node and the ray direction's sign along that axis. Because the optimization process effectively destroys this information, we have added a final pass over the BVH tree to try recreate this. At each interior node, this checks the bounding box of the children and determines which axis provides the greatest separation between the end of the one child's extent and the beginning of the other child's extent. It records this as the node's split axis and the orders the children accordingly. If there is no such axis which separates the bounding boxes' extents then it uses the axis with the least overlap of the extents instead.

We benchmarked the performance impact of the changes on rendering with a single core on a 2.8GHz Core 2 Duo machine so as to avoid jitter and overhead from threading synchronization. For our tests, we used two different rendering modes: first, we tested with pure ray casting (primary rays only), traced in packets of 64 rays each and with the SSE code paths and culling via interval arithmetic culling enabled. This represents state of the art SIMD packet traversal with highly coherent rays. To test the opposite end of the spectrum, we also measured the performance on two-bounce path tracing with the SSE code paths turned off, packets of a single ray each and 1024 samples per pixel. This test emulates classic single ray rendering with rays diverging to traverse the scene from many directions.

Table 1 shows the results of these test on several common models along with the times required by each algorithm to optimize them. The rows showing ray cast and path trace times list the average time in seconds needed to render a single frame. In both cases, the percentages indicate the time needed to render with optimization relative to the time to render without optimization. Similarly, the percentages given in the rows for the SAH cost compare the global SAH cost for the scene after optimization to the cost before. Thus for all rows lower percentages indicate greater benefits from the optimization. The table also provides the time each method spent on optimizing the trees.

As shown in Table 1, the simulated annealing algorithm required significantly more time to process the BVH tree than the hill climbing algorithm but achieved a better reduction in the scenes' SAH costs. The number of iterations to run the simulated annealing is a user controllable parameter, however, and thus it is possible to control the trade-off between the processing time and the degree of improvement to the tree.

6 DISCUSSION

Several things are noteworthy about these results. First, scenes with densely tessellated models such as the Happy Buddha, the Dragon, and even to some extent the Fairy Forest, present the greatest difficulty with respect to optimization. While it is still possible for the optimizer to find small improvements that lower the SAH cost of these scenes, the improvements made tend to be quite small and may even be slightly detrimental to the rendering times. The majority of the iterations of the simulated annealing algorithm for these scenes ended with the computed SAH cost of the scene greater than it was initially.

Interestingly, this suggests that the top-down greedy construction algorithm succeeds at producing good trees for these scenes despite

Scene	Happy Buddha	Dragon	Powerplant §9	Powerplant §16	Fairy Forest	Sponza Atrium	Conference	Soda Hall
Triangles	1,087,474	871,306	121,862	365,970	174,117	76,065	282,664	2,169,132
No Optimization								
SAH Cost	552.6	486.0	154.3	278.8	294.9	713.5	402.4	685.5
Ray Cast Time	0.262	0.496	0.165	0.365	0.372	0.294	0.229	0.326
Path Tr. Time	153.8	197.9	149.2	572.1	370.1	565.5	504.4	1436.8
Hill Climbing Optimization								
SAH Cost	549.6 (99.5%)	484.3 (99.7%)	143.2 (92.8%)	260.4 (93.4%)	290.2 (98.4%)	647.6 (90.8%)	360.3 (89.5%)	626.6 (91.4%)
Ray Cast Time	0.263 (100.2%)	0.496 (100.0%)	0.162 (98.5%)	0.363 (99.4%)	0.373 (100.1%)	0.307 (104.5%)	0.224 (97.8%)	0.320 (98.3%)
Path Tr. Time	154.8 (100.7%)	199.5 (100.8%)	141.2 (94.6%)	567.9 (99.3%)	376.2 (101.6%)	561.4 (99.3%)	446.8 (88.6%)	1417.1 (98.6%)
Time to Opt.	1.71	1.23	0.18	0.53	0.24	0.13	0.68	4.93
Simulated Annealing Optimization								
SAH Cost	549.6 (99.5%)	484.3 (99.7%)	136.1 (88.2%)	247.3 (88.7%)	281.6 (95.5%)	600.8 (84.2%)	340.2 (84.5%)	561.0 (81.8%)
Ray Cast Time	0.263 (100.2%)	0.496 (100.0%)	0.160 (97.1%)	0.354 (97.2%)	0.374 (100.4%)	0.315 (107.3%)	0.216 (94.4%)	0.306 (93.9%)
Path Tr. Time	154.8 (100.7%)	199.5 (100.8%)	127.5 (85.4%)	524.2 (91.6%)	368.6 (99.6%)	547.0 (96.7%)	415.7 (82.4%)	1403.4 (97.7%)
Time to Opt.	528.26	426.48	44.44	114.08	74.11	28.51	119.26	857.57

Table 1: Results of the algorithm applied to various scenes. All times are in seconds and all percentages are relative to the non-optimized values.

using only an approximation to the true cost of the subtrees. At the very least, these trees seem to be close to strong local optima if not the global optimum. This confirms that the greedy approach and the approximations to the subtree costs are valid approaches for these models.

For scenes with triangles of heterogeneous sizes such as the architectural scenes, the optimization seems to fair much better. Here, the simulated annealing algorithm is able to achieve a 15.5% reduction to the cost of the Conference Room scene and a corresponding 17.6% improvement to the path trace rendering time. For these types of scenes, the top-down greedy build does not come as close to building trees near the local optima as it does for the finely tessellated scenes and thus the tree optimizations succeed better at lowering the cost.

Also notable is the degree of correlation between the changes in the SAH cost and the changes in the rendering times. In particular, the correlation between the cost value and the time for packetized ray casting appears to be much weaker than it is between the cost value and the time to path trace with single rays. While this is not surprising, as the surface area heuristic was developed based on the assumption of tracing single rays with well distributed directions, this does suggest the need for an improved heuristic that takes into account the amortization afforded by ray packets with culling. We believe that this would be a valuable area of future work.

7 CONCLUSION

In this paper, we have presented two novel algorithms for improving the SAH cost of an existing BVH tree. Both algorithms are easy to add to existing BVH implementations and are suitable for preprocessing the static content of scenes before rendering, with architectural environments seeing the best improvement. The hill climbing algorithm can yield modest improvements to a scene with millions of primitives in a few seconds, while the simulated annealing algorithm can produce greater improvements given more time.

REFERENCES

[1] G. M. Adelson-Velsky and Y. M. Landis. An algorithm for the organization of information. *Soviet Mathematics Doklady*, 3:1259–1262, 1962.

[2] S. Anily and A. Federgruen. Simulated annealing methods with general acceptance probabilities. *Journal of Applied Probability*, 24(3):657–667, September 1987.

[3] R. Bayer. Symmetric binary b-trees: Data structure and maintenance algorithms. *Acta Informatica*, 1:290–306, December 1972.

[4] J. Bigler, A. Stephens, and S. G. Parker. Design for parallel interactive ray tracing systems. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 187–196, September 2006.

[5] T. Cassen, K. R. Subramanian, and Z. Michalewicz. Near-optimal construction of partitioning trees by evolutionary techniques. In *Graphics Interface '95*, pages 263–271, May 1995.

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

[7] V. Havran. *Heuristic Ray Shooting Algorithms*. PhD thesis, Czech Technical University in Prague, November 2000.

[8] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[9] J. A. Mahovsky. *Ray tracing with reduced-precision bounding volume hierarchies*. PhD thesis, University of Calgary, August 2005.

[10] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, June 1953.

[11] G. Müller and D. W. Fellner. Hybrid scene structuring with application to ray tracing. In *Proceedings of the 1999 International Conference on Visual Computing*, pages 19–26, February 1999.

[12] K. Ng and B. Trifonov. Automatic bounding volume hierarchy generation using stochastic search methods. In *CPSC532D Mini-Workshop "Stochastic Search Algorithms"*, pages 147–161, April 2003.

[13] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, July 1985.

[14] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, January 1985.

[15] I. Wald. On fast construction of sah-based bounding volume hierarchies. In *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, pages 33–40, September 2007.

[16] I. Wald, S. Boulos, and P. Shirley. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics*, 26(1), January 2007.