EUROGRAPHICS 2011 / M. Chen and O. Deussen
(Guest Editors)

Volume 30 (2011), Number 2

RTSAH Traversal Order for Occlusion Rays

Thiago Ize and Charles Hansen

SCI Institute, University of Utah

Abstract

We accelerate the finding of occluders in tree based acceleration structures, such as a packetized BVH and a single ray kd-tree, by deriving the ray termination surface area heuristic (RTSAH) cost model for traversing an occlusion ray through a tree and then using the RTSAH to determine which child node a ray should traverse first instead of the traditional choice of traversing the near node before the far node. We further extend RTSAH to handle materials that attenuate light instead of fully occluding it, so that we can avoid superfluous intersections with partially transparent objects. For scenes with high occlusion, we substantially lower the number of traversal steps and intersection tests and achieve up to $2\times$ *speedups.*

1. Introduction

Occlusion rays can make up a significant contribution of the ray budget of complex scenes. PantaRay, used by Weta Digital, will cast up to a thousand occlusion rays when precomputing scene occlusion [PFHA10]. Boulos reports that for recent ray traced films from Sony Pictures Imageworks, the majority of the rays cast are shadow rays and more than half of these shadow rays, sometimes up to 90% of them, end up being occluded [Bou10]. Finding the occluders as efficiently as possible can thus significantly improve performance for scenes with many occluded rays.

Unlike radiance rays where we are interested in the closest hit object, for occlusion rays, we are interested in whether any hit occurs. We define occlusion rays to be rays that return only visibility and cannot change direction; thus, scattering, reflection, and refraction are not allowed and we consider those rays to be radiance rays. Since we are interested in any hit, instead of traversing a tree based acceleration structure in a front to back order, which is the optimal order for finding the closest hit, we can instead traverse it in any order and terminate traversal as soon as we find any object that fully occludes the ray. We use this to our advantage by choosing the traversal path through the tree that is most likely to quickly occlude the ray.

We accomplish this by developing a novel cost metric for occlusion-ray traversal through a tree, similar to the surface area heuristic (SAH), that gives the expected cost for terminating traversal if a ray enters that node. We can then use the cost metric to choose the traversal order of the child nodes, with the lowest cost child traversed first. We then extend this metric to handle partially occluding objects, such as transparent materials. We show that this method works for both kd-trees and BVHs as well as for single ray traversal and packet traversal. It has minimal preprocess and rendering overhead, no memory overhead, is simple to implement, and can provide significant performance improvements (we show up to $2\times$ improvements in our examples).

2. Background

One method for efficiently finding occluders is to use a shadow cache which keeps track of recent occluders that can be tested against instead of finding them by traversing through an acceleration structure [HG86]. This method works well when most rays intersect the same object, but suffers for most modern scenes which consist of finely tessellated primitives. Caching an entire node subtree can help to ameliorate this problem [PJ91]. Our method can be used alongside these and most other shadow acceleration techniques.

Some approaches for accelerating shadows in massive scenes will approximate the visibility of a group of small objects by a partially transparent bounding box [PFHA10] which could additionally have a directionally varying opacity [LBB*08]. These LOD approaches lower the amount of work performed and amount of scene data that must be touched for scenes with large ray differentials. RTSAH could likely be modified to work with these LOD methods and would of course directly handle the geometry that does not use the LOD optimization.

Djeu et al. showed how shadow rays can be accelerated when ray tracing a single watertight mesh by designating kd-tree nodes inside the mesh as volumetric occluders which can be cheaply intersected with the ray [DKH09]. In order to ensure that a ray can intersect the occluder before the mesh in front of it, they modified the traversal so that instead of the traditional ordered depth-first kd-tree traversal they used a breadth-first search to find the occluder which on average allowed them to reduce the number of traversal steps and triangle intersections performed. They further improved performance by testing all shadow rays against a cache of recently hit volumetric occluders. Our method could be used in place of their breadth-first traversal.

For radiance rays it is well understood that traversing nodes in a front to back order is ideal. For occlusion rays it is not as clear with researchers differing on the appropriate strategy. Smits claims that the front to back ordering did not matter for occlusion rays and to simply traverse it in a depth first order where the left subtree is always traversed first and then followed by the right subtree [Smi98]. Boulos on the other hand disagreed and claims that nodes should be traversed front to back just as with radiance rays [BH10].

2.1. Surface area heuristic

The expected cost of intersecting a ray with a scene using a tree based acceleration structure such as a general BSP tree (and by extension a kd-tree), or BVH can be approximately computed as the cost of all the primitive-ray intersection tests performed plus the cost of all the tree traversal steps performed by the ray. Given the assumption that rays are uniformly distributed and ray traversal will not terminate even if an intersection is found, the number of nodes and primitives seen by the ray is then given by the SAH [GS87, MB90] as follows. The probability of a node being pierced by a ray is

$$P(\text{child hit}|\text{parent hit}) = \frac{\text{SurfaceArea(child)}}{\text{SurfaceArea(parent)}} \quad (1)$$

For convenience, let us define $P_x = P(\text{node x hit}|\text{x's parent hit})$. If we assume fixed costs for performing an intersection test and a node traversal step, $T_{\text{intersection}}$ and T_{step} respectively, then the cost of intersecting a ray with the tree is the following recursive cost.

$$C_{\text{node}} = \begin{cases} P_l C_l + P_r C_r + T_{\text{step}} & \text{if an inner node,} \\ N_{\text{parent}} T_{\text{intersection}} + T_{\text{step}} & \text{if a leaf.} \end{cases} \quad (2)$$

Where N_x is the number of objects in the node x, and l and r correspond to the left and right child nodes. This allows us to compute an expected cost for traversing a ray through a node provided the traversal is not terminated. When used for building a tree based acceleration structure, in order to make the build tractable, a greedy approximation of the SAH is used where the child nodes, C_l and C_r, are assumed to always be leaf nodes [WH06]. Since we will be using the expected traversal costs for traversing an already built tree (likely built using a greedy SAH), we are not limited to relying on a greedy approximation for the traversal ordering.

The SAH is essential for creating high quality tree-based acceleration structures and many authors have attempted to improve upon the SAH. Perhaps the most common improvement to the standard SAH for kd-trees is to encourage large empty volumes to be created by lowering the cost of these splitting planes by a constant factor. This empty space bonus will usually result in slightly higher quality trees. Hunt modified the SAH to handle mailboxing in a kd-tree [Hun08]. His approach is simple to implement and reduces the number of object intersections, but also increases the number of traversal steps. The result was a small performance improvement of a few percent for the scenes he tested. Fabianowski et al. changed the assumption in the SAH that a ray originates outside of the scene to the ray origin being uniformly distributed inside the scene [FFD09]. For scenes that met this assumption, this often resulted in both a reduction in traversal steps as well as object intersections in their kd-tree based systems, with an overall improvement of a few percent.

In a BSP tree, when a ray hits an object, ray traversal is allowed to terminate without requiring further node traversals. A BVH might still require some more traversals because of overlapping nodes, but generally it too will be able to quickly terminate traversal. This early ray termination is a powerful optimization used in essentially all tree-based acceleration structures. Despite its importance, the SAH does not factor in early ray termination and instead assumes rays are never occluded. Havran attempted to remedy this by including early ray termination in the greedy SAH build; however, he found that doing this resulted in a complicated and long build that often hurt performance [Hav01]. This is likely due in part to how the hit probabilities were determined and that the modified SAH did not work well in a greedy build. Furthermore, his modified cost metric incorrectly computes the probability of a ray hitting just one child node and missing the other node (Equations 4.15 and 4.16 in [Hav01]). A simple example that shows this is as follows: assume a parent node has dimensions h, l, and $w \to 0$ (a node that is essentially flat) and it is split in the middle of its shortest dimension to form its two child nodes with dimensions h, l, and $w/2$. Since the two child nodes are practically occupying the same space, we would expect the probability of a ray hitting one node and missing the other to be zero. However, Havran's equation gives this probability as $\frac{\text{SurfaceArea(child)} - \frac{1}{2}\text{SurfaceArea(splitting plane)}}{\text{SurfaceArea(parent)}} = \frac{1}{2}$, where the splitting plane has a front and back side, so its surface area is $2hl$ and the child nodes and parent node in the limit all have the same surface areas of $2hl$.

3. RTSAH

We derive the Ray Termination SAH (RTSAH) by extending the SAH to assume that each node has a continuous visibility function, V. For simplicity, we assume that non-empty leaf nodes are completely opaque, $V = 0$, while empty leaf nodes

are fully transparent, $V = 1$, and an intersection in a node immediately terminates traversal in the tree. An interior node composed of some empty and some non-empty leaf nodes would have a visibility between zero and one. Since a ray can terminate, it is now possible for a ray to enter one child node, but not another either because it never would have pierced that node or because of early ray termination. We therefore need to distinguish between these two cases. P_{lr} is defined to be the probability of a ray piercing both the left and right child nodes. $P_{jl} = P_l - P_{lr}$ is defined to be the probability of the ray piercing just the left node and missing the right node, and similarly $P_{jr} = P_r - P_{lr}$ is the probability of the ray just piercing the right node and missing the left node. The probability of missing both child nodes and just traversing empty space is $P_e = 1 - (P_{jl} + P_{jr} + P_{lr})$.

We now have the terminology to solve for the improved SAH. We first solve for the BVH and then for the general BSP which is a special case of the BVH solution. There are several ways to implement a BVH traversal. In our packetized ranged traversal [WBS07], we first enter a node and then test to see if any rays actually pierce the node. In the event that all the rays terminate in that node's subtree, we then do not need to enter the sibling node and can thus perform one less bounding box intersection test. If the rays do not pierce the first child node or some do but not all terminate, then the other child node is entered. For simplicity, we treat ray packets as single rays, so the cost for entering the left node first is

$$leftFirst = T_{\text{step}} + P_l C_l + (P_{jr} + P_{lr} V_l)(T_{\text{step}} + C_r) + P_e T_{\text{step}} \tag{3}$$

Where the first term, T_{step}, is for intersecting the left bounding box, the second, $P_l C_l$, for the probabilistic cost of having to traverse into the left node, and the last term, $P_e T_{\text{step}}$ for the case where the ray missed the left node, tests the right bounding box, and misses that as well. The final term gives the cost of testing the right bounding box and traversing into it if either the left node was missed, P_{jr}, or it was entered but did not intersect any objects $P_{lr} V_l$.

And similarly the cost for entering the right node first is

$$rightFirst = T_{\text{step}} + P_r C_r + (P_{jl} + P_{lr} V_r)(T_{\text{step}} + C_l) + P_e T_{\text{step}} \tag{4}$$

For radiance rays we would want to enter the near child first. Since the order is ray dependent and the rays are assumed to be random, we have to assume that half of the time the left child might be first and the other half the right child is first. This means that the radiance rays cost for the node is the average of the two if the closer child is traversed first:

$$C_{\text{node}} = \frac{1}{2}(leftFirst + rightFirst) \tag{5}$$

For occlusion rays, we always want to traverse into the lower cost child, so we set the cost to be that of the lower cost child:

$$C_{\text{node}} = Min(leftFirst, rightFirst) \tag{6}$$

Leaf nodes have the same cost as before, as specified in Equation 2. The visibility probability remains the same regardless of the type of ray:

$$V_{\text{node}} = \begin{cases} P_{jl} V_l + P_{jr} V_r + P_{lr} V_l V_r + P_e & \text{if an inner node,} \\ 0 & \text{if nonempty leaf,} \\ 1 & \text{if empty leaf.} \end{cases} \tag{7}$$

P_{lr}/P_l can be viewed as a radiosity form factor expressing the fraction of energy (or rays) that leaves the left node and hits the right node [CW93]. Since we already know how to compute P_l (and P_r), by finding the form factor we can then solve to find P_{lr}, and with that we can easily find the remaining terms, $P_{jl} = P_l - P_{lr}$ for instance. The method we used to compute the form factor from the left node to the right node was to decompose the node bounding boxes into their rectangular faces and then analytically compute the form factors between all valid face pairs and for the overlapped region (see [Cho02] for a good survey of the relevant equations). Other methods should also work provided they do not break down when two objects are very close to each other, overlapping, or share an edge.

For BSPs, we can use the same equations as for BVHs, with the simplifications that only one traversal step computation is performed in order to determine which of the two child nodes are pierced by the ray, and that $P_e = 0$ since the two child nodes completely fill up all space in the parent node. Equation 6 for finding the C_{node} for occlusion rays can be substantially simplified to:

$$C_{\text{node}} = T_{\text{step}} + P_{jl} C_l + P_{jr} C_r + P_{lr} Min(C_l + V_l C_r, C_r + V_r C_l) \tag{8}$$

Furthermore, since $P_e = 0$, this lets us analytically solve for the remaining terms using just P_l and P_r so that $P_{jl} = 1 - P_r$, $P_{jr} = 1 - P_l$, and $P_{lr} = P_l + P_r - 1$. The form factor approach for computing probabilities is thus only needed for BVHs.

Approximate BVH RTSAH The BVH preprocess cost can be lowered at the expense of rendering performance by approximating the probabilities instead of finding them by sampling. We found that assuming that $P_e = 0$, which allows us to use the same probabilities as for a BSP, still offered a reasonable speedup, was simple to implement, and allowed for a very fast preprocess suitable for interactively rendering dynamic scenes.

4. Choosing Traversal Order of Occlusion Rays

Since we now have an expected cost for traversing the left child node or right child node first, we can lower the expected cost of occlusion rays by first entering the child node that gives a lower overall cost. We use the costs from Equations 3 and 4 to determine which to traverse first. Note that this can be simplified to only comparing the values in the *Min* statement of Equation 6, which in the case of the BSP tree, can be a much simpler comparison as seen in Equation 8.

Storage overhead These comparative costs can be computed as a preprocess and stored within the tree for use during traversal. Furthermore, since we are not interested in the actual costs, but only with which child has the smaller cost, we store only a single flag within each parent node to specify which child node should be traversed first in the event that both child nodes can be traversed. This Boolean results in an additional byte per node and can even be embedded in a bit field shared with some other node data which permits each node to not use any extra memory, albeit at a reduction in space for the other shared data. This bit packing is already a common optimization for packing kd-trees into 8 byte nodes.

4.1. Attenuated occlusion rays

Some materials do not fully occlude a ray and instead only attenuate it. In this case an intersection does not result in ray termination. We would therefore like to avoid intersecting objects with transparent materials during traversal in favor of intersecting with an object that fully occludes the ray and allows for ray termination. We do this by modifying Equation 7 so that leaf nodes that contain objects with attenuating materials will have $V = 1$ which signifies that any ray that enters that node will still have to continue traversal after leaving it.

While radiance rays require that the transparent materials be evaluated in front to back order, attenuating materials can be evaluated out of order for rays that only report occlusion, or a fraction of occlusion, such as shadow rays. This means that intersecting and shading transparent materials out of order will not result in incorrect rendering.

A standard front to back ordered high performance ray tracer should already handle transparent materials within the acceleration structure by continuing the traversal of a ray after it hits a transparent object instead of exiting the acceleration structure, shading, and then casting a new ray which must then reenter the acceleration structure and re-intersect all the objects in the node it was just in before continuing on. In that case, extending it to support out of order attenuating materials is fairly straight forward. If not already implemented, the main difficulty is for spatial partitioning structures, such as a BSP/kd-tree, since even with a front to back traversal order it is still possible to intersect objects out of order since there is no ordering within a node and objects might span multiple nodes causing them to get intersected multiple times. If not correctly handled, this can result in over shading of the transparent object which would produce an overly attenuated result. The simplest solution is to keep track of all intersections the ray has made and only shade when a new intersection is found. This is similar to mailboxing when the mailbox keeps track of all previously hit objects and not just the most recent ones. Since this calculation only occurs when an intersection is found and intersections do not normally happen very often, this does not noticeably impact performance.

Figure 1: *Mad Science (80K triangles) rendered with 5 samples per pixel, the 14 area lights are sampled 25 times per shading point, and 36 ambient occlusion rays are cast per shading point. Our method gives a* 2.02× *speedup for the single ray kd-tree and* 1.77× *for packetized BVH.*

5. Results

We demonstrate RTSAH in a packetized ranged traversal BVH and a single ray traversal kd-tree, although these results should transfer to single ray BVHs, packetized kd-trees, and single ray and packetized general BSPs [IWP08]. Both trees were built using a high quality SAH build and in the case of the kd-tree, triangle clipping (perfect splits) were used. Measurements were taken at 1024×1024 pixels on an 8 core Xeon X5550 running at 2.67GHz.

The Mad Science scene shown in Figure 1 uses complex lighting and materials and tests the attenuated shadows optimization of the RTSAH. Ambient occlusion, 14 area lights, transparent materials, dielectrics (which attenuate occlusion rays without bending the ray), and opaque materials are used. Both the kd-tree and the BVH show large speedups, 2.02× and 1.77× respectively, when using the RTSAH compared to the traditional method of tracing rays from the shading point to the light source in a front to back order. Table 1 shows that this is due to the RTSAH halving the number of triangle intersection tests and visiting roughly a third less nodes.

The Bedroom scene (Figure 2) shows an interactive scene with 11 point lights. There are no attenuating materials, so Table 2 does not differentiate between the RTSAH and RTSAH computed without attenuation. The Carnival scene (Figure 3) is shaded only with ambient occlusion and contains opaque, dielectric, and transparent materials.

Figure 2: *11 point lights placed behind louvers in Bedroom scene (361K triangles). Our method with a single ray kd-tree is* 1.38× *faster and packetized BVH is* 1.79× *faster.*

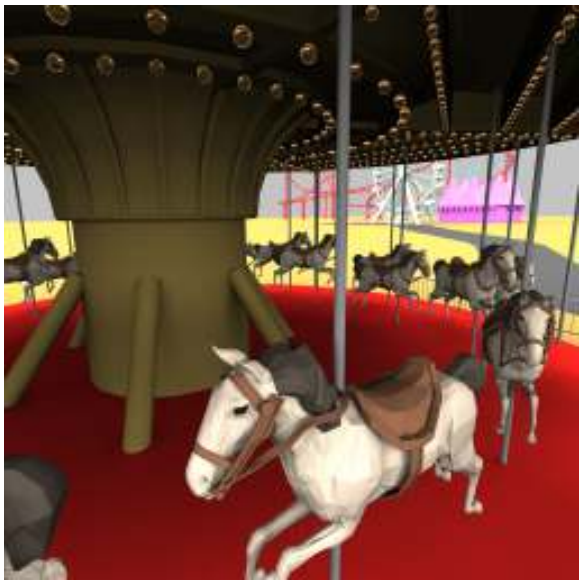

Figure 3: *The Carnival scene (446K triangles) rendered using 5 samples per pixel and 100 ambient occlusion rays per shading point. Our method gives a* 1.64× *speedup for the single ray kd-tree and* 1.88× *for packetized BVH.*

If we do not factor in attenuated occlusion when computing the RTSAH (*RTSAH traversal no-atten* in Tables 1 and 3), then as expected the performance decreases for the Mad Science scene which makes substantial use of attenuating materials. The Carnival scene, however, has many small spheres that attenuate light, but most occlusion rays do not hit these small spheres so the attenuated version of the RTSAH does not offer any improvement over the non-attenuated RTSAH.

Random order traversal Once a child node is entered, its subtree must be fully traversed before its sibling node may be entered; thus, a single wrong traversal choice near the top of the tree can profoundly impact performance. We can observe this by generating random node costs in the preprocess, measuring the resulting performance of RTSAH traversal using the fixed random costs and then swapping the sibling costs so that the cheap nodes are now the expensive nodes and measuring the performance again. Note that using one of these fixed random order traversals is essentially equivalent to Smits' traversal ordering of always traversing the left node first [Smi98], except that it further ensures the removal of any orderings created during tree construction; for instance, always choosing the left node first could easily resort to a front to back ordering depending on the direction of the ray. Several of our tests scenes show a dramatic difference in performance when the opposite paths are taken even though the costs were randomly generated. In fact, the bedroom scene rendered with a kd-tree has one of the random costs being faster than any of the other traversal orderings and its corresponding flipped random order is one of the slowest. If the traversal order is randomly chosen each time a node is visited instead of reusing the same fixed order generated during the preprocess, then the work performed ends up being an average of the two complementary preprocessed random paths. This true random ordering is always less efficient than the RTSAH traversal orderings, but usually more so than the front to back ordered traversal.

SAH order traversal If we ignore early ray termination when computing the RTSAH costs, this gives us the SAH cost of each node which we can then use to decide which node to first traverse. This SAH ordered traversal produces inconsistent results. For the Mad Science scene the SAH ordering is slower than the RTSAH ordering but faster than the front to back, while for the Bedroom scene the SAH ordering performs well for the kd-tree but performs poorly for the BVH and yet the Shadow Overlap scene (Figure 4a) has the opposite effect where SAH ordering is the slowest method for the kd-tree but the fastest for the BVH. The Carnival scene has the SAH being the fastest of all the orderings (even 6% faster than the RTSAH) for the kd-tree but slower than the RTSAH for the BVH. This suggests that the SAH is not a very reliable metric for traversal and in fact, it is possible that some of its very fast results could just be due to random chance since when the SAH does very well, one of the fixed random traversals also does correspondingly well. Just like the fixed random ordering performance could be due to just a few key sibling orderings, the SAH ordering could be randomly making use of the same ordering of those key siblings.

For the single ray kd-tree we can compare against the optimal traversal order found by exhaustively trying all traversal paths, which indicates that our method is still about 1.5× slower than optimal.

We measure the overhead of RTSAH traversal by rendering the Bedroom and Mad Science scenes with a light placed at the camera so that no shadow rays are occluded, and by sampling the light 10 times per shading point so that the occlusion rays and not the radiance rays are the dominant cost. With this setup, RTSAH traversal results in the same number of intersections and traversal steps as the front to back traversal and any performance differences are due solely to the overhead incurred by the RTSAH traversal. The single ray kd-tree ends up being about 3% slower, while the packetized BVH, which was already doing a memory lookup to determine which node is closer, shows no performance loss.

5.1. Preprocess time

Computing the probabilities is linear in the number of nodes and respectively takes roughly 500ms and 640ms per million triangles for the kd-tree and BVH. Though the kd-tree RTSAH computation is an order of magnitude faster per node than for the BVH, because the kd-tree often has an order of magnitude more nodes than a BVH, they end up with roughly the same preprocess time per triangle.

Our preprocess uses just a single thread; however, parallelizing the preprocess should be equivalent to a parallel BVH refitting and could even be done during the refitting. On our 8 core system this would likely allow for using the RTSAH for dynamic scenes. A further optimization is to trade RTSAH accuracy for lower preprocess time by using an approximate RTSAH cost. The approximate RTSAH that we used takes roughly 50ms per million triangles using just one core and is consistently faster than the front to back traversal.

Since an incorrect or outdated cost will still give a correct rendering and likely will not be any worse than a front to back ordering, a single thread could asynchronously compute updated RTSAH costs while the other threads simultaneously render using the partially updated costs, some of which might still be out of date. This is similar in principle to the asynchronous BVH rebuild of Wald et al. [WIP08] except that there is no need to keep two copies of the tree since updates can be made to the tree while it is being rendered. This should further facilitate the usage of RTSAH traversal when low preprocess times are important.

6. Limitations

To describe the limitations of the RTSAH traversal, we lined up four laser scanned models of varying triangle resolutions, placed a light behind all of them so that the shadows of all the models mostly overlap so as to sometimes give several valid traversal options, and rendered it with the kd-tree (see

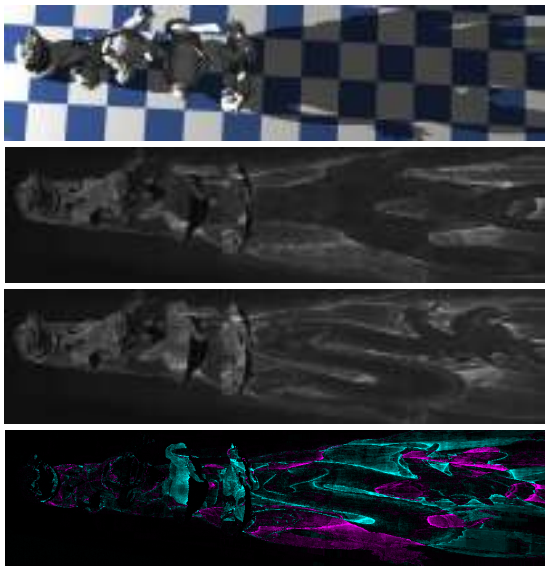

Figure 4: *From left to right, the Shadow Overlap scene (a) is composed of the 1087K tri Happy Buddha, the 30K tri Armadillo, the 69K tri Bunny, and the 871K tri Dragon and has the very small area light sampled 10 times so as to make the occlusion rays the dominant cost while keeping shadows sharp for visualizing purposes. (b) and (c) visualize the time required to render a pixel, with increasing rendering time corresponding with increasing pixel intensity. (b) uses the RTSAH traversal and (c) the front to back traversal. (d) Shows the difference of (b) and (c) with cyan corresponding to where RTSAH traversal was faster and magenta to where it was slower.*

Figure 4a). We expect that in locations where there is overlap that our method will pick the lower cost occluder (usually the Armadillo) and give a speedup, but when there is no overlap it will either result in the same traversal and give no speedup or will perform superfluous work going down wrong paths before finding the only occluder. In Figure 4b, which shows the amount of time required to render a pixel using the RTSAH traversal, we see that it did in fact traverse into the Armadillo model before trying the other models and Figure 4d, which shows which traversal method was faster, confirms that for rays that were occluded by the Armadillo model, the RTSAH traversal was almost always faster (cyan regions). However, for rays that are only occluded by the Dragon, traversing towards the Armadillo resulted in the expected superfluous work and slower performance as noticed by the magenta regions which tend to correspond with where only the Dragon was the occluder. It is possible for a single decision near the top of the tree to have a significant performance impact, so if that one decision is wrong for the set of rays being rendered, then performance can dramatically change. The rays that erroneously chose to traverse towards the Armadillo exhibit this

	kd-tree					BVH				
	render time(s)	tri int per ray	node trvs per ray	leaves per ray	preprocess time(ms)	render time(s)	tri int per ray	node trvs per ray	leaves per ray	preprocess time(ms)
front to back	339	8.64	47.0	9.3	0	276	24.0	125.2	13.9	0
RTSAH traversal	168	4.27	33.2	5.4	38	156	11.8	81.0	6.9	54
RTSAH traversal no-atten	207	5.20	34.1	5.6	38	169	12.1	82.4	7.1	52
approx RTSAH traversal	-	-	-	-	-	182	14.5	92.9	8.6	3.7
SAH traversal	229	5.43	37.0	7.1	34	221	15.3	104.6	10.4	1.5
fixed random	237	5.83	38.1	6.7	46	189	15.0	94.1	9.1	2.8
fixed random opp	222	5.34	37.7	6.7	46	256	20.6	123.0	12.3	2.8
optimal	-	2.94	23.3	3.2	-	-	-	-	-	-

Table 1: *The Mad Science scene (80K triangles). 73% of the 2596M occlusion rays are occluded.*

	kd-tree					BVH				
	render time(s)	tri int per ray	node trvs per ray	leaves per ray	preprocess time(ms)	render time(s)	tri int per ray	node trvs per ray	leaves per ray	preprocess time(ms)
front to back	0.870	10.4	43.5	8.8	0	0.413	13.1	101.4	7.0	0
RTSAH traversal	0.629	8.8	31.9	5.7	214	0.231	7.4	66.5	3.8	250
approx RTSAH traversal	-	-	-	-	-	0.226	6.9	65.3	3.4	9.1
SAH traversal	0.680	8.8	35.0	7.7	187	0.377	11.3	98.3	6.1	5.7
fixed random	0.840	9.5	45.2	8.0	220	0.398	12.6	96.1	6.7	7.5
fixed random opp	0.629	9.1	29.7	5.4	220	0.229	7.6	63.8	3.6	7.5
optimal	-	5.4	19.4	1.6	-	-	-	-	-	-

Table 2: *The Bedroom scene (361K triangles). 93% of the 11M occlusion rays are occluded.*

since they must traverse many nodes in that subtree before being able to leave and enter the correct sibling node containing the Dragon. This effect can also be seen with how the preprocessed fixed random traversal orders can sometimes have widely different performance. Any optimized traversal ordering can only accelerate completely occluded occlusion rays. For areas or scenes that are predominantly radiance rays or unoccluded occlusion rays, there is little work that can be accelerated and so of course our method will see only a minor speedup. All the lit surfaces in Figure 4a cannot be speedup for this reason and Figure 4d shows that there is in fact no noticeable performance difference. Finally, due to the logarithmic time complexity of ray tracing, though there might be an order of magnitude difference in number of triangles between the largest and smallest model in this scene, the cost savings of rendering the lower res model will be much smaller. For these reasons, the kd-tree RTSAH traversal still gives a speedup, but only a modest one of 1.15× (the BVH performed better at 1.45×).

The RTSAH is derived with the assumption of random ray distributions and for scenes that have a truly random ray distribution, it should always outperform any competing traversal method. However, useful renderings do not usually use completely random rays and so for a specific sets of rays, the lower cost child could end up requiring more work than its higher cost sibling node. For instance, inside the lower cost child node's subtree there could be a grandchild node with very low probability of being reached but with a very expensive cost if it does get traversed. If all the rays enter that expensive grandchild (perhaps a light source is in that node) then the lower cost child node will end up being more expensive than its sibling for those specific rays. This can lessen the performance gain of the RTSAH traversal or, albeit rare, even make it slower than the front to back traversal.

Another reason the RTSAH traversal might not be significantly faster than a front to back traversal is that the front to back traversal might already be doing the optimal traversal order. In this situation, RTSAH traversal clearly can not do any better and can only hope to match performance. In practice this does not usually occur for all rays and so our method will usually end up outperforming the front to back traversal, although sometimes it will only be slightly faster.

7. Future Work

The RTSAH assumes random rays that completely pierce the node. This of course is usually not true since the rays could have fixed sources (shading points) and destinations (light sources). Removing or lessening the assumption on rays being completely random could lead to further performance benefits. For instance, if we know a ray will not go past the light, then it would be nice to use a cost metric that ignores objects behind the light. Likewise, objects behind a shading point should be ignored when computing the cost.

	kd-tree					BVH				
	render time(s)	tri int per ray	node trvs per ray	leaves per ray	preprocess time(ms)	render time(s)	tri int per ray	node trvs per ray	leaves per ray	preprocess time(ms)
front to back	72.5	11.2	56.4	11	0	112.1	33.4	398.8	35.7	0
RTSAH traversal	44.1	4.8	38.2	6.7	181	59.6	18.1	214.2	20.5	364
RTSAH traversal no-atten	43.9	4.8	43.2	6.7	181	59.6	18.1	214.2	20.5	345
approx RTSAH traversal	-	-	-	-	-	60.6	17.2	220.4	20.7	33.9
SAH traversal	41.5	3.8	35.3	7.7	159	82.6	19.6	316.3	27.4	8.4
fixed random	58.1	8.4	45.8	8.6	186	128.9	32.4	463.5	41.2	11
fixed random opp	43.3	4.4	36.4	7.3	186	69.4	18.7	254.0	23.6	11
optimal	-	2.5	24.3	3.3	-	-	-	-	-	-

Table 3: *Carnival (446K triangles) shaded with only ambient occlusion. 81% of the 539M occlusion rays are occluded.*

	kd-tree					BVH				
	render time(s)	tri int per ray	node trvs per ray	leaves per ray	preprocess time(ms)	render time(s)	tri int per ray	node trvs per ray	leaves per ray	preprocess time(ms)
front to back	0.185	2.59	43.2	8.9	0	0.157	4.38	128.5	17.3	0
RTSAH traversal	0.161	2.57	39.0	7.3	382	0.108	3.85	86.6	10.1	1312
approx RTSAH traversal	-	-	-	-	-	0.115	3.97	93.5	11.7	78
SAH traversal	0.194	2.36	45.9	10.8	335	0.111	3.89	89.0	10.5	51
fixed random	0.191	2.50	44.2	9.2	393	0.163	4.46	133.1	17.5	66
fixed random opp	0.200	2.47	44.8	9.3	393	0.129	4.01	103.1	12.1	66
optimal	-	1.42	25.9	3.8	-	-	-	-	-	-

Table 4: *The Shadow Overlap Scene (2056K triangles). 56% of the 2.5M occlusion rays are occluded.*

The RTSAH is used in this paper for choosing the traversal order for occlusion rays; however, it can also be used for measuring tree quality for radiance rays (see equation 5). Building a tree that minimizes the RTSAH cost for radiance rays should result in a higher quality tree. The RTSAH can easily be used to measure the build quality of already built trees, but using it to guide the build is more challenging and an area of future research.

We demonstrated the RTSAH traversal for trees with two children; however, it could be extended to trees with an arbitrary number of children as used with the mBVH/QBVH varients [DHK08, EG08, WBB08]. Furthermore, with more child nodes to choose from, m-ary trees should benefit even more from RTSAH traversal.

8. Conclusion

We presented an improved version of the SAH, which we call the RTSAH, that takes into account ray termination and gives the expected traversal cost of radiance and occlusion rays through a tree. We then showed how the RTSAH can be used to guide the traversal of occlusion rays through a tree so that an intersection can be more efficiently located. The RTSAH traversal can try to avoid attenuating materials for a further improvement in traversal efficiency. The RTSAH can be computed faster than the tree can be built and there is practically no storage or rendering overhead for using it. For scenes that comprise mostly of occluded rays, the RTSAH traversal can give a substantial performance increase.

9. Acknowledgements

This publication is based on work supported by Award No. KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST), DOE VACET, NSF OCI-0906379, and NSF CRI-0551724. Pete Shirley wrote code to compute the BVH RTSAH using Monte Carlo sampling. We are extremely grateful for the anonymous reviewer's suggestion to use form factors to solve the BVH RTSAH. The Mad Science scene is by Dan Konieczka and Giorgio Luciano; the Carnival by Dan Konieczka, the Bedroom was modeled by David Vacek and designed by David Tousek; all three scenes are available from the 3dRender.com Lighting Challenges. The Happy Buddha, Bunny, Dragon, and Armadillo are courtesy of the Stanford Computer Graphics Laboratory.

References

[BH10] BOULOS S., HAINES E.: Sorted BVHs. *Ray Tracing News 23*, 1 (2010). 2

[Bou10] BOULOS S.: How often do shadow rays hit? *Ray Tracing News 23*, 1 (2010). 1

[Cho02] CHOI A.-S.: Practical applications of form factor computation in lighting calculations. *Building and Environment 37*, 11 (2002), 1107–1115. 3

[CW93] COHEN M. F., WALLACE J. R.: *Radiosity and Realistic Image Synthesis*. Morgan Kaufmann Publishers, 1993. 3

[DHK08] DAMMERTZ H., HANIKA J., KELLER A.: Shallow bounding volume hierarchies for fast SIMD ray tracing of incoherent rays. In *Eurographics Symposium on Rendering* (2008), pp. 1225–1234. 8

[DKH09] DJEU P., KEELY S., HUNT W.: Accelerating shadow rays using volumetric occluders and modified kd-tree traversal. In *High Performance Graphics* (2009), ACM, pp. 69–76. 2

[EG08] ERNST M., GREINER G.: Multi bounding volume hierarchies. In *Symposium on Interactive Ray Tracing* (2008), pp. 35–40. 8

[FFD09] FABIANOWSKI B., FOWLER C., DINGLIANA J.: A cost-metric for scene-interior ray origins. In *Eurographics Short Presentations* (2009). 2

[GS87] GOLDSMITH J., SALMON J.: Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications 7*, 5 (1987), 14–20. 2

[Hav01] HAVRAN V.: *Heuristic Ray Shooting Algorithms*. PhD thesis, Faculty of Electrical Engineering, Czech Technical University in Prague, 2001. 2

[HG86] HAINES E. A., GREENBERG D. P.: The light buffer: A ray tracer shadow testing accelerator. *IEEE CG&A 6*, 9 (Sept. 1986), 6–16. 1

[Hun08] HUNT W.: Corrections to the surface area metric with respect to mail-boxing. In *Symposium on Interactive Ray Tracing* (2008), pp. 77–80. 2

[IWP08] IZE T., WALD I., PARKER S. G.: Ray tracing with the BSP tree. In *Symposium on Interactive Ray Tracing* (2008), pp. 159–166. 4

[LBB*08] LACEWELL D., BURLEY B., BOULOS S., SHIRLEY P., STUDIOS W., BURBANK C.: Raytracing prefiltered occlusion for aggregate geometry. In *Symposium on Interactive Ray Tracing* (2008), pp. 19–26. 1

[MB90] MACDONALD J. D., BOOTH K. S.: Heuristics for ray tracing using space subdivision. *Visual Computer 6*, 6 (1990), 153–65. 2

[PFHA10] PANTALEONI J., FASCIONE L., HILL M., AILA T.: PantaRay: fast ray-traced occlusion caching of massive scenes. In *SIGGRAPH* (2010), ACM, pp. 1–10. 1

[PJ91] PEARCE A., JEVANS D.: Exploiting shadow coherence in ray tracing. In *Graphics Interface* (1991), pp. 109–116. 1

[Smi98] SMITS B.: Efficiency issues for ray tracing. *Journal of Graphics Tools 3*, 2 (1998), 1–14. 2, 5

[WBB08] WALD I., BENTHIN C., BOULOS S.: Getting rid of packets—efficient SIMD single-ray traversal using multi-branching BVHs. In *Symposium on Interactive Ray Tracing* (2008), pp. 49–58. 8

[WBS07] WALD I., BOULOS S., SHIRLEY P.: Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics 26*, 1 (2007), 1–18. 3

[WH06] WALD I., HAVRAN V.: On building fast kd-trees for ray tracing, and on doing that in $O(N \log N)$. In *Symposium on Interactive Ray Tracing* (2006), pp. 61–70. 2

[WIP08] WALD I., IZE T., PARKER S. G.: Fast, parallel, and asynchronous construction of BVHs for ray tracing animated scenes. *Computers & Graphics 32*, 1 (2008), 3–13. 6