

T-SAH: Animation Optimized Bounding Volume Hierarchies

J. Bittner and D. Meister

Czech Technical University in Prague, Faculty of Electrical Engineering



Figure 1: Average speedups of ray tracing for several tested animations when using our single T-SAH optimized BVH with refitting compared to SAH with refitting and per frame rebuilding using HLBVH algorithm. From left to right: Fairy Forest, Dragon Bunny, San Miguel + Running Horses, Breaking Lion.

Abstract

We propose a method for creating a bounding volume hierarchy (BVH) that is optimized for all frames of a given animated scene. The method is based on a novel extension of surface area heuristic to temporal domain (T-SAH). We perform iterative BVH optimization using T-SAH and create a single BVH accounting for scene geometry distribution at different frames of the animation. Having a single optimized BVH for the whole animation makes our method extremely easy to integrate to any application using BVHs, limiting the per-frame overhead only to refitting the bounding volumes. We evaluated the T-SAH optimized BVHs in the scope of real-time GPU ray tracing. We demonstrate, that our method can handle even highly complex inputs with large deformations and significant topology changes. The results show, that in a vast majority of tested scenes our method provides significantly better run-time performance than traditional SAH and also better performance than GPU based per-frame BVH rebuild.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Raytracing—I.3.5 [Computer Graphics]: Object Hierarchies—I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—

1. Introduction

Ray tracing stands at the core of many rendering algorithms. With the advances of computational power it is possible to use ray tracing for interactive applications, including those involving dynamic or animated scenes. Bounding Volume Hierarchies became a very popular data structure for ray tracing. They exhibit a predictable memory budget, fast construction, very good ray tracing performance, and also allow easy updates for handling dynamic scenes.

A number of efficient techniques have been proposed for bounding volume hierarchies. For example methods for construction of high quality BVHs [WBKP08, BHH13, KA13, GHFB13] are particularly suitable for offline applications as they provide hierarchies leading to the best possible ray tracing performance. Other class of techniques uses very fast BVH construction targeting at interactive and real-time applications with dynamic and animated scenes [GPM11, Kar12]. These methods lead to real-time or interactive performance for scenes of low to moderate complexity.

In many cases animated or deformable models exhibit large amount of coherence and just refitting the BVH and keeping its topology provides a very good overall performance. However, in some cases this approach breaks due to topology changes or large deformations [WBS07]. Several methods have been proposed to handle this issue by dynamically updating the BVH at runtime [LYTM06, KIS^{*}12]. We propose to use a novel approach based on temporal extension of the surface area heuristic (T-SAH). We describe an algorithm that uses T-SAH to construct a single BVH optimized for a given animation sequence. In the most of the test cases the T-SAH based BVH provides surprisingly stable results even for scenes involving very complex topology changes (see the accompanying paper video or snapshots in Figures 5 and 6 for the range of topology changes we aim to handle). Apart from animated scenes the method can also handle deformable and/or articulated objects such as animated characters.

Since the BVH can be optimized for the whole animation sequence in the preprocess, there is no need to rebuild the BVH at runtime. Our tests indicate, that for interactive rendering of complex scenes with known animations this leads to significantly better performance than rebuilding the BVH at runtime. The savings are not only due to faster update time, but the measured ray tracing performance is also higher for most tested scenes, since the preprocess allows to construct BVH of better quality than the methods working at runtime.

2. Related Work

SAH Bounding volume hierarchies have a long tradition in rendering and collision detection. Kay and Kajiya [KK86] designed one of the first BVH construction algorithms using spatial median splits. Goldsmith and Salmon [GS87] proposed the measure currently known as the *surface area heuristic* (SAH), which predicts the efficiency of the hierarchy during the BVH construction. The vast majority of currently used methods for BVH construction use a top-down approach based on SAH. A particularly popular method is the fast approximate SAH evaluation using binning proposed by Havran et al. [HHS06] and Wald et al. [Wal07].

SAH extensions Several extensions of the basic SAH have been proposed in the past. Hunt [Hun08] proposed corrections of SAH with respect to mailboxing. Fabianowski et al. [FFD09] proposed SAH modification for handling scene interior ray origins. Bittner and Havran proposed to modify SAH by including the actual ray distribution [BH09], Feltman et al. [FLF12] extended this idea to shadow rays. Corrections of the SAH based BVH quality metrics have been recently proposed by Aila et al. [AKL13].

Parallel BVH construction Another class of methods target fast parallel BVH construction. Lauterbach et al. [LGS^{*}09] proposed a GPU BVH construction algorithm

using a 3D space-filling curve. Wald [Wal12] studied the possibility of fast rebuilds from scratch on an Intel architecture with many cores. Pantaleoni and Luebke [PL10], Garanzha et al. [GPM11], and Karras [Kar12] proposed GPU based methods for efficient parallel BVH construction. These techniques achieve impressive performance, but generally construct a BVH of lower quality than the full SAH builders. We use the HLBVH method [GPM11] as one of the references for our comparisons.

High quality BVH Recently more interest has been devoted to methods, which are not limited to the top-down BVH construction. Walter et al. [WBKP08] proposed to use bottom-up agglomerative clustering for constructing a high quality BVH. Gu et al. [GHFB13] proposed a parallel approximative agglomerative clustering for accelerating the bottom-up BVH construction. Kensler [Ken08], Bittner et al. [BHH13], and Karras and Aila [KA13] proposed to optimize the BVH by performing topological modifications of the existing tree. These approaches allow to decrease the expected cost of a BVH beyond the cost achieved by the traditional top down approach. Our method uses an extension of the insertion based optimization algorithm proposed by Bittner et al. [BHH13].

Dynamic and animated scenes Wald et al. [WBS03] suggested to decompose dynamic scenes into multiple objects with their local transformations and use a top-level hierarchy for organizing these objects. This concept has recently been used within the Embree ray tracer [WWB^{*}14]. Lauterbach et al. [LYTM06] proposed to rebuild subtrees if they exhibit significant SAH cost changes. Gunther et al. [GFW^{*}06] subdivided the model into clusters of coherent movement and used fuzzy kd-trees to describe the residual geometry changes. Olsson [Ols07] studied 4D kd-trees focusing on optimizing their performance for rendering motion blur. Wald et al. [WBS07] proposed to use BVH refitting combined with packet based traversal for rendering dynamic scenes. Ize et al. [IWP07] proposed asynchronous BVH reconstruction and refitting for increasing runtime performance. Updating BVH at runtime using rotations has been proposed by Yoon et al. [YCM07] and more recently by Kopta et al. [KIS^{*}12]. Garanzha [Gar09] precomputes coherent triangle clusters using motion prediction in order to accelerate the BVH construction at runtime. Mora [Mor11] proposed to perform ray tracing without an explicit data structure using a divide and conquer algorithm. Ray tracing animated scenes was surveyed by Wald et al. [WMG^{*}09]. Unlike the previous methods, our method is the first to create a single ordinary BVH optimized for the whole animation, requiring no topology modifications at runtime.

The paper is further structured as follows: Section 3 presents an overview of the proposed method, Section 4 introduces the T-SAH cost model, Section 5 describes the BVH construction using T-SAH, Section 6 presents the results and their discussion, and Section 7 concludes the paper.

3. Algorithm Overview

Our algorithm starts by resampling the input animation sequence into a small number of representative frames. The representative frames provide information about the geometry distribution during the animation. We construct a BVH for the first representative frame using a standard sweep based algorithm based on SAH. The BVH is then iteratively optimized using the cost model given by the temporal extension of the surface area heuristic (T-SAH). T-SAH simultaneously considers costs of all representative frames, and thus optimizing according to T-SAH will improve the overall performance of the BVH for the input animation sequence.

The actual BVH optimization algorithm is a modification of the method proposed by Bittner et al. [BHH13] that uses the new T-SAH cost model. The method performs a series of remove and insert operations on subtrees of the BVH in order to minimize the T-SAH cost of the BVH. Since the estimated ray tracing performance is known during the BVH optimization procedure, the method can put more effort to optimize difficult representative frames. An illustration of the optimization and its impact on a scene involving significant topology changes is shown in Figure 2.

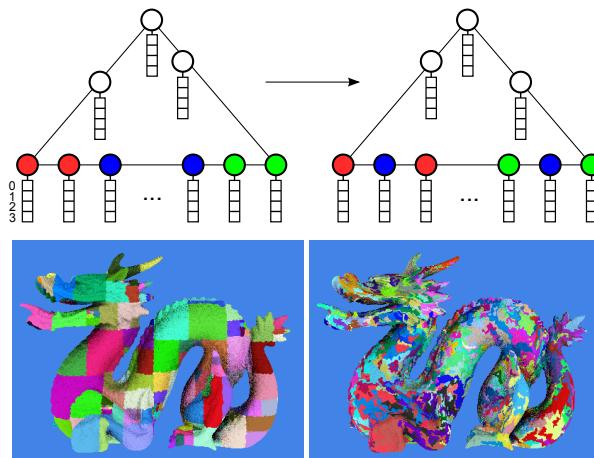


Figure 2: Illustration of the BVH optimization using T-SAH. During the optimization we keep an array of bounding boxes for each BVH node, which correspond to bounding volumes for different representative frames. The optimization regroups nodes at all tree levels to yield clusters as compact as possible considering all frames. The bottom row shows visualization of a BVH constructed for the Dragon Bunny animation using SAH (left) and T-SAH (right) (a cut in the BVH involving nodes with less than 5.000 polygons is shown). Note how T-SAH automatically identifies groups of polygons forming fragments of the broken Dragon that appear in the later stages of the animation.

4. T-SAH for Animations

The SAH cost is based on expressing the probability of hitting a bounding volume with a random unoccluded ray. Once the ray hits a bounding volume of area S we express the probability of hitting its child bounding volume S' as the ratio of surface areas S'/S . The SAH cost of the whole BVH can be expressed as a function of time as:

$$C(t) = \frac{c_T}{S(t)} \sum_{N \in \text{inner}} S_N(t) + \frac{c_I}{S(t)} \sum_{N \in \text{leaves}} S_N(t) j_N, \quad (1)$$

where $S_N(t)$ is the surface area of node N at time t , $S(t)$ is the surface area of the root of the BVH, j_N is the number of triangles in leaf N , and c_T, c_I are costs for ray traversal and ray intersection computations.

We resample the animation sequence or the pose space of a character into a set of n representative frames at corresponding times t_i for which we obtain a discrete set of n BVH costs $C_i = C(t_i)$, $0 \leq i < n$. Using the representative frames we define the T-SAH cost \tilde{C} , which is a temporal extension of the SAH cost function. T-SAH cost \tilde{C} expresses the cost of the whole animation sequence as a weighted average of the frame costs:

$$\tilde{C} = \frac{1}{\sum_i w_i} \sum_i w_i C_i \quad 0 \leq i < n \quad (2)$$

where w_i is the weight of the representative frame i .

As we do not assume a priori knowledge of importance of the representative animation frames, we use sampling at uniform time intervals and we set the weights w_i to depend on the actual cost of the frame: $w_i = C_i^k$. The exponent k is the parameter of the method influencing the relative contribution of the cost for the given frame to the T-SAH cost function. The T-SAH cost function is then given as:

$$\tilde{C}^k = \frac{1}{\sum_i C_i^k} \sum_i C_i^{k+1} \quad 0 \leq i < n \quad (3)$$

Note that for the special case of $k = 0$, all the frames are weighted equally and the T-SAH cost function \tilde{C}^0 corresponds to the average cost of the sampled animation frames:

$$\tilde{C}^0 = \frac{1}{n} \sum_i C_i \quad 0 \leq i < n \quad (4)$$

If the exponent k becomes larger (e.g. $k \geq 2$) significantly more weight is given to the frames with high values of the cost function. Thus when minimizing the T-SAH cost function these frames will have higher importance and the algorithm should aim to optimize the BVH by reducing the cost

of these frames with higher priority. In other words when minimizing the T-SAH cost function using higher exponents k we should eliminate peaks of the sampled cost function even at the price that the average cost \bar{C}^0 will increase. This is useful for example in the scenario when we have a strict time budget for rendering a frame and we want that the BVH provides performance fulfilling this budget for the whole animation sequence.

The above described definition expresses the T-SAH cost of the whole BVH. When evaluating a cost of a subtree using Eq. 2 we use the global weights w_i corresponding to the whole tree as they give a better overview of the importance of different representative frames for the animation.

5. Building BVH using T-SAH

We decided to use T-SAH with the insertion based BVH optimization method proposed by Bittner et al. [BHH13]. This method works by optimizing an already existing BVH, and so it can take into account all the components of the T-SAH cost model including the calculation of weights w_i based on the costs of the representative frames. This section describes the BVH construction algorithm focusing on the modifications required for supporting the T-SAH cost model.

5.1. Insertion Based Optimization

The elementary step of the insertion based optimization method is removing a node from the BVH and reinserting it back at a position reducing the global tree cost. Here we do not reproduce the basics of the method and we refer the interested reader to the work of Bittner et al. [BHH13]. The removal of a node and its insertion back into the BVH requires only simple local operations on the tree once the optimal insertion position is found.

The crucial part of the algorithm is searching for the optimal position to re-insert the node into the BVH while accounting for the T-SAH cost model. Similarly to the standard version of the insertion based optimization the algorithm uses branch-and-bound search. This search tracks the cost increase caused by expanding bounding volumes of nodes above the currently examined insertion position (induced cost). The other component of the total cost increase is given by inserting a new node connecting the re-inserted subtree and the node which was identified as the best insertion point (direct cost). In the algorithm by Bittner et al. [BHH13] the induced cost corresponds to the surface area increase of nodes on the path to the root and the direct cost to the surface area of the union of bounding boxes of the inserted node and the box of the node at the insertion point.

The modification for T-SAH involves computing vectors of costs instead of scalar variables, where the components of these vectors correspond to different representative frames. For example the components of the vector of induced costs

correspond to surface area increase at different representative frames above the node that is currently visited by the searching algorithm. The actual costs are then evaluated from the cost vectors using the T-SAH model (Eq. 2).

To accelerate the search procedure we use an approximation to T-SAH cost model, that only updates weights w_i after a batch of reinsertion operations has been performed. We used a batch size equal to 2% of the number of nodes in the BVH. Note that when optimizing the average cost \bar{C}^0 the weights are constant and need not be re-evaluated in the whole optimization process.

5.2. Selecting Nodes for Update

One choice to apply the above described optimization is to select nodes to be updated in random order. However, using some prioritization scheme can accelerate the optimization process [BHH13]. As a priority for updating a given node we use its contribution Δ_N^k to the cost function \bar{C}^k . This contribution corresponds to a weighted sum of surface areas of the node over all representative frames, where the weights correspond to frame weights w_i :

$$\Delta_N^k = \sum_i w_i S_N(t_i) = \sum_i C_i^k S_N(t_i)$$

For \bar{C}^0 this priority simplifies to a sum of surface areas of the node bounding box over all representative frames.

We use an independent chain Metropolis-Hastings sampling in order to quickly select nodes for updates. We want the target distribution of selected nodes to be proportional to Δ_N^k . We pick up a node X_{i+1} using a uniform proposal distribution. This node is either accepted or rejected based on the ratio of the cost contributions of the node and the previously accepted node $\Delta_{i+1}^k / \Delta_i^k$. This process introduces a correlation, i.e. some nodes would be drawn a number of times in a sequence. In our application updating the same node in a sequence does not make sense as the deterministic search would always find the same optimal position for this node in the BVH. We therefore continue the sampling until we accept a different node, which is then updated. The actual distribution of selected nodes does not strictly follow Δ_N^k , but nevertheless it is able to significantly accelerate the convergence of the BVH cost (about 1.5 to 4x compared to random sampling). We also observed that for complex animations, the described stochastic method converges faster than the deterministising batch sorting proposed for optimizing a single BVH [BHH13], as the later one primarily focuses only on small percentage of nodes with high costs. On the contrary for simpler animations the batch sorting based selection provided the best convergence rate (see Figure 3).

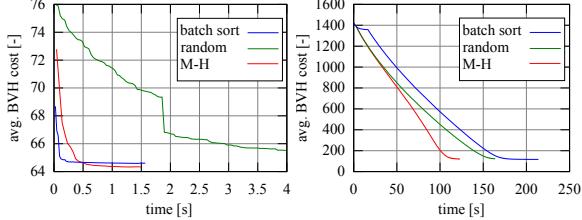


Figure 3: Convergence rates of BVH optimization for different node selection methods (left: BART robots, right: BART museum). Note that on a very complex animation sequence like BART Museum batch sorting was not very successful and the cost was reduced only after switching to random sampling after couple of initial optimization passes (at $t=20s$).

5.3. Terminating the Optimization

We terminate the BVH optimization if there was no reduction of the BVH cost in the given number of update batches. We observed, that terminating after three batches when the cost was not reduced was a good overall setting for the tested scenes.

5.4. Adaptive Leaf Size

For obtaining an optimized BVH with the highest possible quality it is important to initially construct a deep BVH with a single triangle per leaf. This allows the T-SAH based optimization algorithm to identify fine grained groups of triangles which exhibit coherent movement or deformations. Having such a deep BVH is however not desirable for the actual rendering. An optimal leaf size depends on the constants of the ray tracer implementation, the type of animation, and ray distribution.

To finalize the BVH we compact the tree by collapsing the subtrees into larger leaf nodes as proposed by Bittner et al. [BHH13]. In the case of animated scenes this does not only account for particular constants of the ray tracer (ratio of c_T/c_I), but it also efficiently handles some potential deformations of the model. The algorithm evaluates the cost of the subtree C_N^k and compresses it when the cost of the subtree becoming a leaf is smaller.

6. Results

We evaluated the proposed method on nine animated scenes. Eight scenes represent standard benchmarks for animated ray tracing, one scene is a composition of a static model of a moderate complexity (7.8M triangles), and an animation representing several horses running through the scene. The overview of tested animated scenes is given in Table 1.

6.1. BVH Optimization

First, we compared the costs and the build times of BVHs constructed offline using different methods. For the comparison we used BVH constructed for every frame using the full sweep SAH (SAH-rebuild), the best-over-time BVH [WBS07] with refitting (SAH*), BVH constructed for the first frame with refitting (SAH⁰), and the proposed T-SAH with refitting using three different settings (C^0 , C^2 , \widetilde{C}^4). For SAH-rebuild and SAH* we also employed the insertion based optimization [BHH13]. For SAH⁰ this optimization was not used as we observed that for some complex animations, like Museum or Dragon-Bunny, optimizing the BVH for the first frame slightly increased the average BVH cost for the whole animation. The measurements were performed within a single threaded C++ implementation on a PC with Intel Xeon E5-1620, 3.6GHz, with 32GB RAM. The results are summarized in Table 1. For T-SAH we used 5 uniformly sampled representative frames. The evaluation was performed using 50 interpolated animation frames for all tests.

The table shows that the BVH construction time for T-SAH varies between 1.9s for the smallest reported scene (BART robots) and 328s for the largest one (San Miguel + Horses). The actual T-SAH optimization time corresponds to the difference between the build times of the particular T-SAH method and SAH⁰, as we always applied the T-SAH based optimization on the BVH constructed for the first frame. Comparing to SAH⁰ we can observe, that for simpler animations the reduction of the average BVH cost is not very dramatic (in order of 10-20%), which indicates that even a single SAH optimized BVH is quite good to handle these animations. However for scenes with extreme deformations such as Dragon Bunny, Breaking Lion, or BART museum our method reduces the cost by more than an order of magnitude. We can see that in these cases the SAH* corresponding to the best over time BVH also provides much better results than SAH⁰, but the BVH cost is still significantly larger than for T-SAH methods. As expected the SAH-rebuild reference method provides the best results in all tests, but in the simpler animations the SAH* and T-SAH methods are very close to this reference solution. We can also observe, that in scenes with simple geometric changes different exponents of the T-SAH cost model lead to very similar results. However in scenes with pronounced peaks of the cost function such as Dragon-Bunny or Museum, the T-SAH with higher exponents provides lower maximum BVH costs at the expense of a higher average BVH cost. In other words the cost function gets more smoothed for the whole animation sequence (see Figure 4-left). This is important for time critical applications, like for example ray traced augmented reality with animated objects, in which we want to ray trace the pre-processed animation at stable real-time rates. Figure 4-right shows the progress of the optimization for different settings of the T-SAH cost model.

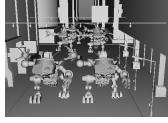
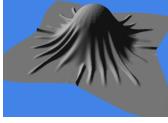
	BART Robots # triangles 71.5k # key frames 300		BART Museum # triangles 75.8k # key frames 300		Cloth Ball # triangles 92.2k # key frames 94						
test	time [s]	cost	max. cost	test	time [s]	cost	max. cost	test	time [s]	cost	max. cost
SAH-rebuild	24.7	56	57	SAH-rebuild	39.7	40	148	SAH-rebuild	22.9	73	91
SAH*	18.5	57	58	SAH*	18.6	364	1376	SAH*	20.8	97	156
SAH ⁰	0.4	69	71	SAH ⁰	0.4	1558	2335	SAH ⁰	0.4	97	156
T-SAH $\widetilde{C^0}$	1.9	57	57	T-SAH $\widetilde{C^0}$	123	119	365	T-SAH $\widetilde{C^0}$	9.0	90	139
T-SAH $\widetilde{C^2}$	1.9	57	57	T-SAH $\widetilde{C^2}$	155	136	282	T-SAH $\widetilde{C^2}$	9.7	90	129
T-SAH $\widetilde{C^4}$	2.0	57	57	T-SAH $\widetilde{C^4}$	161	149	266	T-SAH $\widetilde{C^4}$	0.6	98	147
	BART Kitchen # triangles 110.5k # key frames 300		24 Cell # triangles 122.8k # key frames 128		Fairy Forest # triangles 174.1k # key frames 21						
test	time [s]	cost	max. cost	test	time [s]	cost	max. cost	test	time [s]	cost	max. cost
SAH-rebuild	41.6	26	26	SAH-rebuild	49.2	111	120	SAH-rebuild	68.5	76	77
SAH*	32.6	26	26	SAH*	28.6	161	181	SAH*	22.7	76	77
SAH ⁰	0.7	37	38	SAH ⁰	0.5	195	250	SAH ⁰	1.0	87	92
T-SAH $\widetilde{C^0}$	3.6	26	26	T-SAH $\widetilde{C^0}$	13.1	131	142	T-SAH $\widetilde{C^0}$	7.5	76	77
T-SAH $\widetilde{C^2}$	3.7	26	26	T-SAH $\widetilde{C^2}$	13.5	131	142	T-SAH $\widetilde{C^2}$	6.2	76	77
T-SAH $\widetilde{C^4}$	3.7	26	26	T-SAH $\widetilde{C^4}$	13.5	131	141	T-SAH $\widetilde{C^4}$	5.3	76	77
	Dragon Bunny # triangles 252.5k # key frames 15		Breaking Lion # triangles 1604.0k # key frames 34		Horses # triangles 8048.9k # key frames 31						
test	time [s]	cost	max. cost	test	time [s]	cost	max. cost	test	time [s]	cost	max. cost
SAH-rebuild	112	57	134	SAH-rebuild	912	555	644	SAH-rebuild	6603	130	131
SAH*	25.8	474	1441	SAH*	460	747	907	SAH*	2906	159	182
SAH ⁰	1.4	1542	2865	SAH ⁰	12.2	1530	3801	SAH ⁰	89	214	281
T-SAH $\widetilde{C^0}$	13.2	137	365	T-SAH $\widetilde{C^0}$	116	571	658	T-SAH $\widetilde{C^0}$	328	133	133
T-SAH $\widetilde{C^2}$	34.0	102	220	T-SAH $\widetilde{C^2}$	116	567	647	T-SAH $\widetilde{C^2}$	319	133	133
T-SAH $\widetilde{C^4}$	28.7	103	190	T-SAH $\widetilde{C^4}$	120	566	640	T-SAH $\widetilde{C^4}$	319	133	133

Table 1: Summary results for the high quality BVH construction algorithms. The table shows the BVH optimization time, and the average and maximum BVH costs for the given animation sequence. For computing the BVH cost, we used $c_T = 3$, $c_I = 2$.

We also evaluated the dependence of the method on the number of representative frames. When increasing the number of representative frames from 5 to 10 the BVH optimization time almost doubled, the maximum BVH cost was slightly reduced, but the resulting average BVH costs stayed practically the same for most tested scenes. The most notable differences we observed for the Museum and Breaking Lion scenes where the average BVH cost was reduced by 1% and the maximum BVH cost by about 2%. This indicates that a relatively small number of representative frames is sufficient to optimize even for complex animation sequences.

6.2. Performance Comparisons

We evaluated the performance of the optimized BVH for ray tracing using CUDA ray tracing framework by Aila

et al. [AL09]. We extended the framework by implementing new kernels for interpolating triangle vertices, refitting bounding volumes, recomputing Woop matrices, and implementing the HLBVH algorithm. The actual ray tracing kernel was used without modifications. All computations thus run on the GPU, the CPU is used only to execute CUDA kernels. For all tests we used NVIDIA GeForce GTX TITAN Black GPU with 6GB RAM.

We used five reference methods for comparisons. The first one uses precomputed high quality BVH (SAH-rebuild), the second one uses best over time BVH with refitting (SAH*), the third one uses BVH build for the first frame with refitting (SAH⁰). The forth reference method uses the fast GPU build using the HLBVH algorithm [GPM11] limited to spatial median splits with 30-bit Morton codes (HLBVH-med).

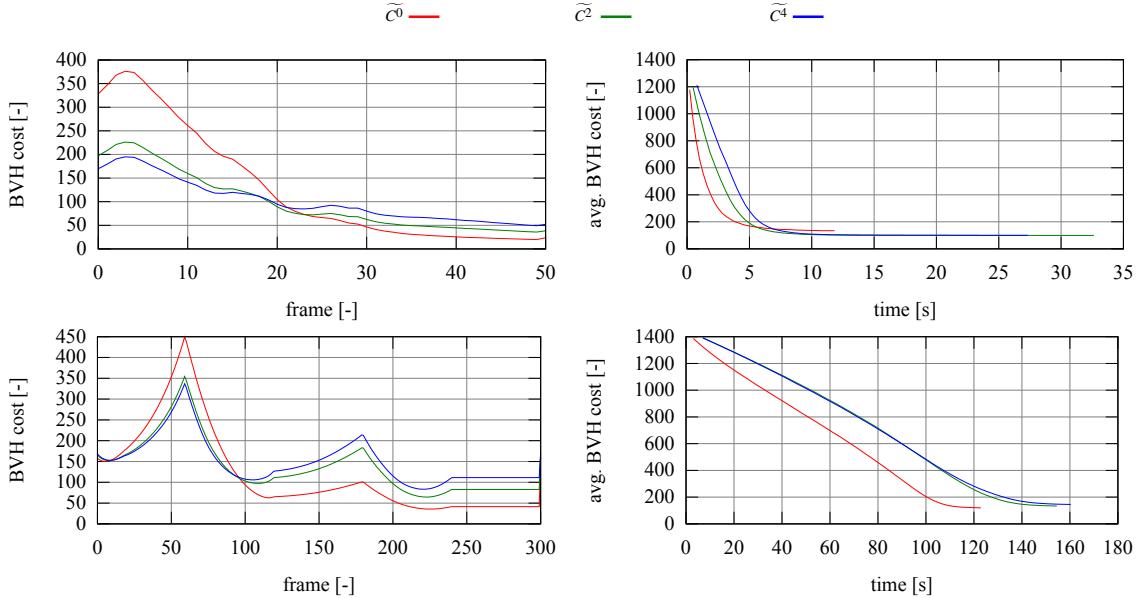


Figure 4: Plots showing the costs for the BVH optimized using T-SAH with different exponents \widetilde{C}^0 , \widetilde{C}^2 , \widetilde{C}^4 . (left) Costs of the optimized BVH for different frames of the animation. (right) Average BVH cost (\widetilde{C}^0) in the progress of the BVH optimization. The upper figures correspond to the Dragon Bunny scene, the lower to the Museum scene.

The fifth reference uses HLBVH including the surface area heuristic (HLBVH-SA) with 30-bit Morton codes (15-bits SAH, 15-bits spatial median). Both HLBVH algorithms use maximum 8 triangles per leaf as a termination criterion. For all tests we used 1024x768 resolution, 1 primary ray per pixel, 8 ambient occlusion rays, and 2 shadow rays. We measured the time needed to update the BVH for the given frame (using either refit or rebuild), the ray tracing speed expressed in MRays/s, and the total frame time.

The results are summarized in Table 2. We can observe that in the majority of cases T-SAH C^0 provides the best total frame times for the tested animations. In the case of relatively simple animations (BART Robots, BART Kitchen, Fairy Forest) the results of SAH⁰, SAH*, and T-SAH are very close and the timing differences are also influenced by the actual match of the BVH topology and the view point. For scenes which involve complex animations with deformations or topological changes, the SAH⁰ method fails completely (BART Museum, Dragon Bunny, Breaking Lion), whereas T-SAH \widetilde{C}^0 with refit successfully compensates for the dynamic changes of the scene. The HLBVH methods provide a stable performance as the BVH is always constructed from scratch. However, with increasing size of the scene the overhead of BVH rebuild becomes more and more important and the method provides significantly lower frame times than T-SAH. Interestingly for most tests the T-SAH provides not only lower BVH update times, but also

higher ray tracing speed even comparing to the HLBVH-SA method. This is due to the fact, that the method uses a more complex BVH optimization than the approximate SAH algorithm used in HLBVH. A notable exception is the BART Museum scene, in which both HLBVH methods provide smaller total times. This scene exhibits extreme topological changes and although T-SAH provides more than 10x better performance than SAH⁰, it is not able to compensate for all these changes. Nevertheless when splitting the animation into three parts and constructing a BVH using T-SAH for each of these parts, we can observe that the overall frame time for T-SAH gets very close to rebuilding the BVH from scratch at each frame (see Table 2 - BART Museum*). The influence of using multiple BVHs is also shown in Figure 5. While more elaborate techniques for splitting the animation sequence could be used such as adaptive splitting and/or finding the minimal number of BVHs fulfilling the given performance limit, we leave these extensions as topics for future work. Note that the differences in refit times are due to the adaptive leaf size discussed in Section 5.4, which for the case of using the T-SAH cost model usually lead to slightly deeper tree.

The frame times and the ray tracing speed for two selected animated scenes are shown in Figure 6. We can observe, that for the Dragon Bunny scene T-SAH is able to eliminate the large total time increase of the SAH method caused by the explosion of the geometry of the dragon. In the Breaking Lion scene we can observe that T-SAH successfully com-

penses the part of the animation, in which the lion is broken into a large number of disconnected pieces. This scene has a higher polygon count and therefore we can observe a constant overhead of both HLBVH methods in total time, which is caused by a longer time needed to rebuild the BVH in each frame.

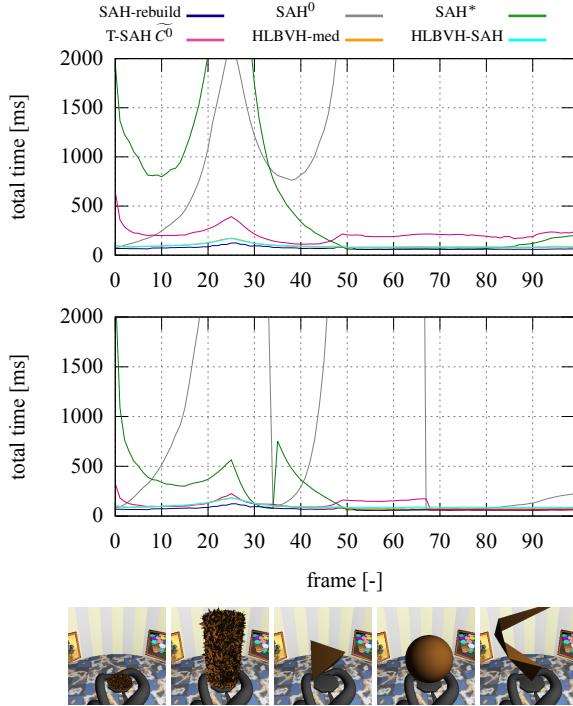


Figure 5: Plots of the total frame time for BART Museum scene. (top) Single BVH optimized using T-SAH. (bottom) Using 3 BVHs optimized for frames 0-32, 33-65 and 66-99 using T-SAH.

6.3. Discussion and Limitations

Maximizing ray tracing performance In terms of ray tracing performance an optimal solution for a known animation sequence would precompute a high quality BVH for every frame that is rendered as we did using SAH-rebuild method. However this method has important practical drawbacks compared to T-SAH. First, for scenes with large number of key frames the SAH-rebuild becomes very memory intensive. Second, the implementation has to maintain large number of BVHs, which requires more implementation effort and complicates the usage with existing frameworks. The first issue can actually be handled by using T-SAH optimized BVH with a range of key frames (as we suggest for the Museum scene). Given the same memory budget T-SAH optimized BVHs should provide better ray tracing performance than the same number of SAH optimized ones.

Dynamic scenes In fully dynamic scenes new objects can appear, or existing objects can undergo some movement,

which cannot be predicted. However we believe, that T-SAH could still be used to optimize parts of such scenes for which the movement is well predictable. We also expect that by feeding T-SAH optimization algorithm with a set of expected positions of dynamic objects would cause an automatic separation of dynamic and static content. Those objects which are expected to behave in a similar way would get clustered together in appropriate levels of the BVH. T-SAH could also be used in combination with the decomposition to local subtrees proposed by Wald et al. [WBS03] to handle dynamic scenes with unpredictable object movement.

Optimization time We implemented the BVH optimization using a simple single threaded application that takes significantly more time than fast GPU builders. However, this phase is only done once in preprocessing and at runtime the final BVH can be used just as an ordinary BVH, on which only refit is applied.

Using T-SAH with other algorithms We expect that bottom up clustering could easily handle T-SAH as the T-SAH could serve as a basis for evaluating the distance between clusters. We believe that T-SAH could be successfully exploited also in combination with a GPU based BVH optimization using rotations [KIS^{*}12], or with bottom-up BVH construction algorithms [GHFB13]. Using these methods might lead to lower optimization times for applications where this is required.

7. Conclusion and Future Work

We proposed an extension of the surface area heuristic to temporal domain (T-SAH). We exploit T-SAH to construct a single bounding volume hierarchy optimized for all frames of an animated sequence. The algorithm uses a small set of representative frames and performs an iterative optimization of the BVH in order to minimize the T-SAH cost. Our method uses an extension of a recently proposed insertion based BVH optimization algorithm. We have shown that the algorithm is able to handle a broad class of animated scenes including those involving complex deformations and topology changes. We evaluated the performance of T-SAH optimized BVH for ray tracing. The results show that T-SAH performs significantly better than SAH and in a vast majority of cases it outperforms even the fast per frame rebuilds using the HLBVH algorithm. The proposed method requires practically no modifications to existing rendering packages and thus it can be used as an optional optimization step for handling animated scenes in the context of real-time ray tracing or collision detection.

We believe that our extension of SAH to animations gives a new insight on how hierarchical object partitioning can organize the scene by exploring locality in both spatial and temporal domains. In the future we want to look at adaptively subdividing the input animation sequence in order to provide a minimal set of BVHs within a given memory budget, which yield the optimal rendering performance. We also

	update time [ms]	trace speed [MRays/s]	total time [ms]									
BART Robots												
BART Museum												
BART Museum*												
SAH-rebuild	0.0	179.8	64.8	0.0	184.9	70.9	0.0	184.9	70.9	0.0	87.5	49.1
SAH*	7.4	183.1	70.7	2.3	19.5	674.5	2.4	56.0	234.1	3.0	78.8	57.0
SAH ⁰	3.8	190.1	66.3	2.3	6.3	2082.4	2.1	6.7	1958.0	2.5	81.2	55.5
T-SAH $\widetilde{C^0}$	7.1	190.7	68.2	2.7	62.7	212.0	2.1	118.3	110.8	3.0	82.2	55.0
HLBVH-med	18.1	117.0	119.7	10.3	161.3	91.7	10.3	161.3	91.7	12.1	79.6	66.7
HLBVH-SAHA	49.3	129.4	139.0	22.6	177.9	96.5	22.6	177.9	96.5	31.2	80.3	84.6
Cloth Ball												
BART Kitchen												
24 Cell												
Fairy Forest												
Dragon Bunny												
Breaking Lion												
Horses												

Table 2: Performance comparison of tested methods. The reported numbers represent the average values for the whole animation sequence. The times were measured using CPU side timers and thus they include the CUDA kernel execution and setup times. For SAH*, SAH⁰, and T-SAH method a single BVH is used for the whole animation, SAH-rebuild, HLBVH-med, and HLBVH-SAHA perform BVH rebuild in each frame. Note that for BART Museum* we used three BVHs optimized for different parts of the animation. The results with the best trace speed and total time for a given scene are highlighted in bold (note that this does not include the SAH-rebuild reference method).

want to study the possibility of taking into account the expected ray distribution by creating BVHs optimized not only for the scene geometry changes, but also for the expected camera positions. Finally, we want to exploit T-SAH in the context of other applications such as collision detection or run time BVH optimization using temporal prediction of object movement.

Acknowledgements

This research was supported by the Czech Science Foundation under research program P202/12/2413 (Opalis) and the Grant Agency of the Czech Technical University in Prague, grant No. SGS13/214/OHK3/3T/13.

References

- [AKL13] AILA T., KARRAS T., LAINE S.: On Quality Metrics of Bounding Volume Hierarchies. In *Proceedings of High Performance Graphics* (2013), ACM, pp. 101–108. [2](#)
- [AL09] AILA T., LAINE S.: Understanding the Efficiency of Ray Traversal on GPUs. In *Proceedings of HPG* (2009), pp. 145–149. [5](#)
- [BH09] BITTNER J., HAVRAN V.: RDH: Ray Distribution Heuristics for Construction of Spatial Data Structures. In *Proceedings of SCCG* (2009), ACM, pp. 61–67. [2](#)
- [BHH13] BITTNER J., HAPALA M., HAVRAN V.: Fast Insertion-Based Optimization of Bounding Volume Hierarchies. *Computer Graphics Forum* 32, 1 (2013), 85–100. [1, 2, 3, 4, 5](#)
- [FFD09] FABIANOWSKI B., FOWLER C., DINGLIANA J.: A cost metric for scene-interior ray origins. *Eurographics, Short Papers* (2009), 49–52. [2](#)
- [FLF12] FELTMAN N., LEE M., FATAHALIAN K.: SRDH: Specializing BVH Construction and Traversal Order Using Representative Shadow Ray Sets. In *Proceedings of HPG* (2012), pp. 49–55. [2](#)
- [Gar09] GARANZHA K.: The use of precomputed triangle clusters for accelerated ray tracing in dynamic scenes. *Comput. Graph. Forum* 28, 4 (2009), 1199–1206. [2](#)
- [GFW*06] GUNTHER J., FRIEDRICH H., WALD I., SEIDEL H.-P., SLUSALLEK P.: Ray tracing animated scenes using motion decomposition. *Computer Graphics Forum* 25, 3 (2006), 517–525. [2](#)
- [GHFB13] GU Y., HE Y., FATAHALIAN K., BLELLOCH G. E.: Efficient BVH Construction via Approximate Agglomerative Clustering. In *Proceedings of High Performance Graphics* (2013), ACM, pp. 81–88. [1, 2, 7](#)
- [GPM11] GARANZHA K., PANTALEONI J., MCALLISTER D.: Simpler and Faster HLBVH with Work Queues. In *Proceedings of posium on High Performance Graphics* (2011), pp. 59–64. [2, 5](#)
- [GS87] GOLDSMITH J., SALMON J.: Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics and Applications* 7, 5 (May 1987), 14–20. [2](#)
- [HHS06] HAVRAN V., HERZOG R., SEIDEL H.-P.: On the Fast Construction of Spatial Data Structures for Ray Tracing. In *Proceedings of IEEE Symposium on Interactive Ray Tracing 2006* (Sept 2006), pp. 71–80. [2](#)
- [Hun08] HUNT W.: Corrections to the surface area metric with respect to mail-boxing. In *Interactive Ray Tracing, 2008. RT 2008. IEEE Symposium on* (Aug 2008), pp. 77–80. [2](#)
- [IWP07] IZE T., WALD I., PARKER S. G.: Asynchronous BVH Construction for Ray Tracing Dynamic Scenes on Parallel Multi-Core Architectures. In *Proceedings of Symposium on Parallel Graphics and Visualization '07* (2007), pp. 101–108. [2](#)
- [KA13] KARRAS T., AILA T.: Fast Parallel Construction of

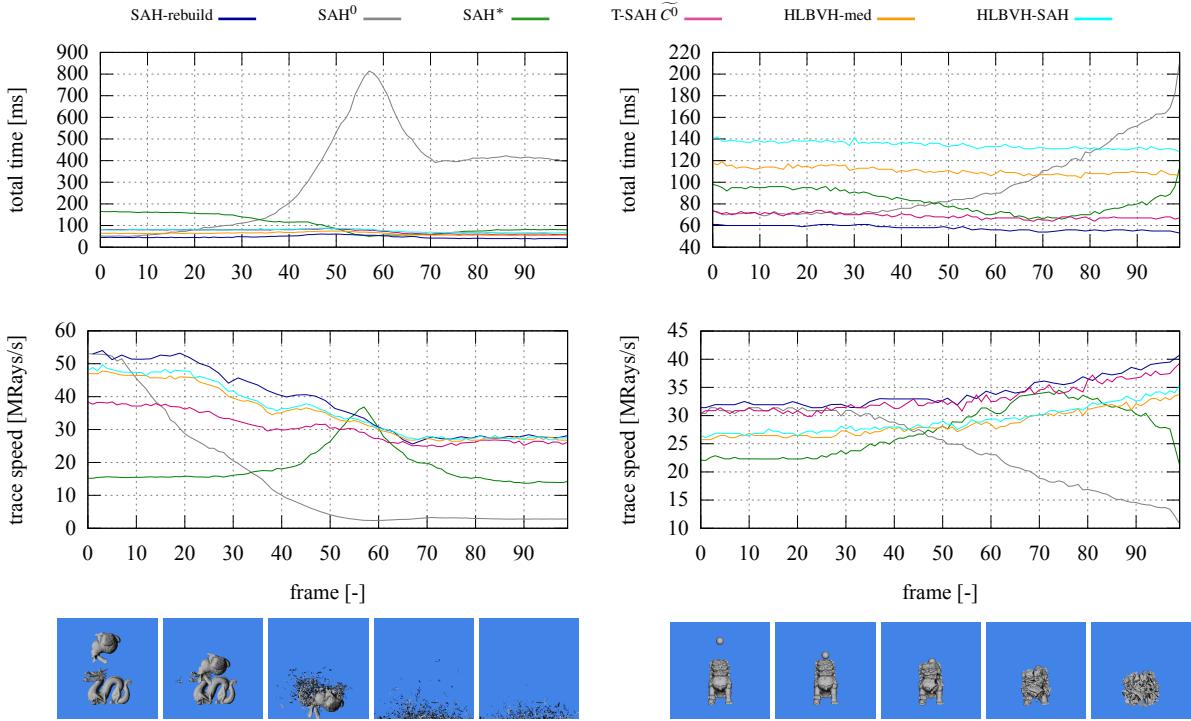


Figure 6: Plots of the total frame times and ray tracing speed for the Dragon-Bunny scene (left) and Breaking Lion scene (right).

- High-Quality Bounding Volume Hierarchies. In *Proceedings of High Performance Graphics* (2013), ACM, pp. 89–100. [1](#), [2](#)
- [Kar12] KARRAS T.: Maximizing Parallelism in the Construction of BVHs, Octrees, and k-d Trees. In *Proceedings of High Performance Graphics* (2012), pp. 33–37. [2](#)
- [Ken08] KENSLER A.: Tree Rotations for Improving Bounding Volume Hierarchies. In *Proceedings of the 2008 IEEE Symposium on Interactive Ray Tracing* (Aug 2008), pp. 73–76. [2](#)
- [KIS*12] KOPTA D., IZE T., SPJUT J., BRUNVAND E., DAVIS A., KENSLER A.: Fast, effective bvh updates for animated scenes. In *Proceedings of Symposium on Interactive 3D Graphics and Games* (2012), pp. 197–204. [2](#), [7](#)
- [KK86] KAY T. L., KAJIYA J. T.: Ray Tracing Complex Scenes. *Computer Graphics (SIGGRAPH '86 Proceedings)* 20, 4 (1986), 269–278. [2](#)
- [LGS*09] LAUTERBACH C., GARLAND M., SENGUPTA S., LUEBKE D., MANOCHA D.: Fast BVH Construction on GPUs. *Comput. Graph. Forum* 28, 2 (2009), 375–384. [2](#)
- [LYTM06] LAUTERBACH C., YOON S.-E., TUFT D., MANOCHA D.: RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs. In *IEEE Symposium on Interactive Ray Tracing (RT'06)* (Sept 2006), pp. 39–46. [2](#)
- [Mor11] MORA B.: Naive ray-tracing: A divide-and-conquer approach. *ACM Trans. Graph.* 30, 5 (2011), 117:1–117:12. [2](#)
- [Ols07] OLSSON J.: *Ray Tracing Animations Using 4D Kd-Trees*. Master's thesis, Lund University, 2007. [2](#)
- [PL10] PANTALEONI J., LUEBKE D.: HLBVH: Hierarchical LBVH Construction for Real-Time Ray Tracing of Dynamic Geometry. In *Proceedings of High Performance Graphics '10* (2010), pp. 87–95. [2](#)
- [Wal07] WALD I.: On fast Construction of SAH based Bounding Volume Hierarchies. In *Proceedings of the Symposium on Interactive Ray Tracing* (2007), pp. 33–40. [2](#)
- [Wal12] WALD I.: Fast Construction of SAH BVHs on the Intel Many Integrated Core (MIC) Architecture. *IEEE Transactions on Visualization and Computer Graphics* 18, 1 (2012), 47–57. [2](#)
- [WBKP08] WALTER B., BALA K., KULKARNI M., PINGALI K.: Fast Agglomerative Clustering for Rendering. In *IEEE Symposium on Interactive Ray Tracing* (2008), pp. 81–86. [1](#), [2](#)
- [WBS03] WALD I., BENTHIN C., SLUSALLEK P.: Distributed interactive ray tracing of dynamic scenes. In *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (2003), pp. 77–86. [2](#), [7](#)
- [WBS07] WALD I., BOULOS S., SHIRLEY P.: Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Trans. Graph.* 26, 1 (Jan. 2007). [2](#), [5](#)
- [WMG*09] WALD I., MARK W. R., GÜNTHER J., BOULOS S., IZE T., HUNT W. A., PARKER S. G., SHIRLEY P.: State of the art in ray tracing animated scenes. *Comput. Graph. Forum* 28, 6 (2009), 1691–1722. [2](#)
- [WWB*14] WALD I., WOOP S., BENTHIN C., JOHNSON G. S., ERNST M.: Embree: A kernel framework for efficient CPU ray tracing. *ACM Transactions on Graphics* 33 (2014). [2](#)
- [YCM07] YOON S.-E., CURTIS S., MANOCHA D.: Ray Tracing Dynamic Scenes using Selective Restructuring. In *Proceedings of Eurographics Symposium on Rendering* (2007), pp. 73–84. [2](#)