STAG: Smart Tools & Apps for Graphics (2015)
Silvia Biasotti and Marco Tarini and Andrea Giachetti (Editors)

Faster Ray-Traced Shadows for Hybrid Rendering of Fully Dynamic Scenes by Pre-BVH Culling

K. Selgrad J. Müller M. Stamminger

University of Erlangen-Nuremberg

Abstract

With ever increasing ray traversal and hierarchy construction performance the application of ray tracing to problems often tackled by rasterization-based algorithms is becoming a viable alternative. This is especially desirable as the ground truth for these algorithms is often determined by using ray tracing and thus directly applying it is the simplest way to generate images satisfying the reference. In this paper we propose a very efficient pre-process to speed up the construction and traversal of sub-optimal, but fast-to-build hierarchies used for interactive ray tracing and show how it can be applied to shadow rays in a hybrid environment, where ray tracing is used to sample area lights for scene positions found and shaded via rasterization.

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

1. Introduction

The fast computation of shadows in computer generated images has been one of the core problems of rendering research for decades, and even the efficient rendering of high quality *hard shadows*, i.e. shadows from simple point light sources, is still an active area of investigation [WHL15]. The task of computing *soft shadows*, i.e. shadows cast by arbitrarily shaped area light sources, is an especially challenging problem in that domain. Many approaches targeting this effect have been presented and applied successfully. However, fast methods are usually based on approximations (e.g. based on an average occluder depth [Fer05]) only solve some restricted form of the problem [GBP06] or suffer from quality issues [SFY13]. The reference for all those methods is soft shadows computed by ray tracing.

Ray tracing performance has steadily increased and it has become ever more relevant for interactive applications [ALK12, KKW*13]. For interactive rendering, the raw traversal speed itself has become secondary [LGS*09]. The reason for this is that interactive applications call for frequent reconstruction of acceleration structures to manage dynamic scenes. Accordingly, rendering time is considered to encompass both, hierarchy construction time and ray traversal time. These demands can lead to choosing inferior hierarchies as the increase in traversal time can be made up by considerable savings during hierarchy construction.

In this paper we present a novel method to increase the quality of such hierarchies by introducing a fast pre-process that culls potentially large parts of the scene prior to hierarchy construction. Our method is applicable when the source and destination of the rays to be traced are known, such as for computing soft shadows, and the additional computation required is often amortized with even a single sample per pixel, much more so if higher quality shadows are computed.

The contribution of our work is an extension to hierarchy construction that improves performance of ray-traced shadows and yields results that are exactly the same as ground truth. This is achieved by computing a conservative, but non-convex estimate of the part of the scene that will be traversed by shadow rays, thus significantly reducing the number of triangles that have to be considered for hierarchy construction and thereby the size of the hierarchy used for traversal. We also provide an analysis of the window where our approach is beneficial in contrast to directly computing high quality structures.

The remainder of this paper is structured as follows. Section 2 lists related work, both regarding soft shadow approaches for interactive rendering, as well as methods targeting interactive ray tracing. We then describe our BVH culling technique in Section 3, provide an evaluation of its applicability in Section 4 and conclude with Section 5.

2. Related Work

Producing plausible shadows for arbitrary scene configurations is one of the most deeply researched problems in the field of rendering. For hard shadows, real-time applications rely predominantly on shadow mapping [Wil78] and shadow volumes [Cro77]. Even though much research has been done on approximations for computing soft shadows more quickly in interactive contexts, with the ever increasing performance of ray tracing approaches (both hierarchy construction and ray traversal performance) such methods start to become a viable alternative to faster, but less general approximations. The remainder of this section will therefore focus on interactive techniques for high-quality soft shadow approximations and list advances in the field of ray traced shadows that converge towards real-time rendering.

Soft Shadow Volumes. Shadow volumes [Cro77] employ a polygonal representations (potentially created on-the-fly) to determine shadowed areas by rasterization. Based on this scheme, penumbra wedges [AAM03] can be added to objects' silhouettes to capture the whole penumbra region. Forest et al. [FBP06] tackle the overestimation of this approach by deriving blending heuristics.

Backprojection. A popular approach to approximate soft shadows is backprojecting occluders stored in a shadow map onto the light source, reducing the light's contribution accordingly [GBP06]. Guennebaud et al. [GBP07] extend this method to address overestimation and light leaks. Bavoil et al. [BCS08] manage overlapping occluders by considering multiple layers and bitmask soft shadows [SS07] keeps track of occluded parts of the light source. Refer to the books by Eisemann et al. [ESAW11] and Woo and Poulin [WP12] for further variants of the backprojection scheme.

Soft Shadow Mapping. The fundamental restriction of shadow mapping to soft shadow computations lies in the fact that shadow maps cannot be filtered on their own (i.e. prefiltered). Filtering can only be applied during shadow lookup where both the query depth and the shadow map's values are available. Accordingly, percentage closer soft shadows [Fer05] compute an approximate filter size based on the average occluder depth during shadow lookup.

Summed-area variance shadow maps [LM07] (SAVSM) and variance soft shadow mapping [YDF*10] (VSSM) are soft-shadow extensions of variance shadow mapping [DL06] (VSM). The shadow map representation of VSM allows to filter shadow map data by using its moments and applying Chebyshev's inequality. SAVSM achieves varying penumbra size with constant lookup time using a summed area table of the VSM data and trade shadow overestimation for a reduction of more objectionable light leaks. VSSM speeds up the computation of the average blocker distance and apply filter-kernel partitioning to avoid light leaking.

Convolution shadow maps [AMB*07] (CSM) approximate the shadow test as a weighted summation of basis terms. Exponential shadow maps [AMS*08] (ESM) build upon this using a more compact and well behaved exponential basis. These approaches have been extended for soft shadows by Annen et al. [ADM*08] and Yang et al. [SFY13], respectively.

A common limitation of soft shadow mapping approaches is that filtering over strong depth variations is prone to introducing light leaks or shadow overestimation. Also, occluder fusion cannot be properly addressed without layered extensions, and even then remains a challenging problem.

Multi-Layer Approaches. The drawbacks of methods based on shadow maps can be addressed by using multi-layer shadow maps. With this extension depth discontinuities can be managed better, and handling of overlapping occluders can be incorporated. Layered variance shadow maps [LM08] provide such an extension to VSM. In addition to tackling overlapping occluders and complicated configurations, multi-layer transparent shadow maps [XTP07] and multi-layer filtering approaches [SDMS14] also manage transparent occluders.

These approaches are capable to achieve much better shadow quality for complicated geometric configurations, but are still based on approximations, and as such may exhibit artifacts when compared to ray tracing of the actual scene geometry.

Interactive Ray Tracing. The reference solution for real-time algorithms is usually computed by ray tracing [Whi80]. Improving the performance of ray traversal has been the focus of active research for decades [KKW*13]. Initially, research strived for ultimate ray traversal speed by finding faster intersection algorithms, closely modelling algorithms to hardware, e.g. by accounting for SIMD instruction sets [DHK08, EG08, WWB*14] or the underlying execution model [ALK12]. The optimal construction of highly effective acceleration structures [MB90, Wal07, SFD09] contributed significantly to increasing traversal speed.

However, with the trend towards interactive ray tracing, it has been realized that for fully dynamic scenes traversal time is only part of the problem. Especially highly optimized algorithms for hierarchy construction can consume a considerable amount of time, and when only few samples are taken this can easily outweigh any gains in traversal time. Therefore, approaches that consider the *time to image*, e.g. construction and setup time in addition to traversal time, have become very popular for interactive applications [LGS*09, PL10, Kar12, KA13]. In contrast to approaches that modify existing acceleration structures [WBS07, KIS*12] these methods provide more reliable performance.

Note that our proposed culling scheme is applicable to all hierarchical structures, but provides greater benefit to fast approaches that generate sub-optimal structures.

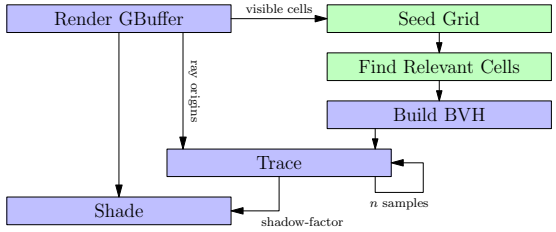

Figure 1: *Overview of our method. The blue nodes are part of a regular hybrid rendering pipeline, whereas the green nodes are introduced by our algorithm. The input of the dynamic scene geometry is omitted, but present in both cases.*

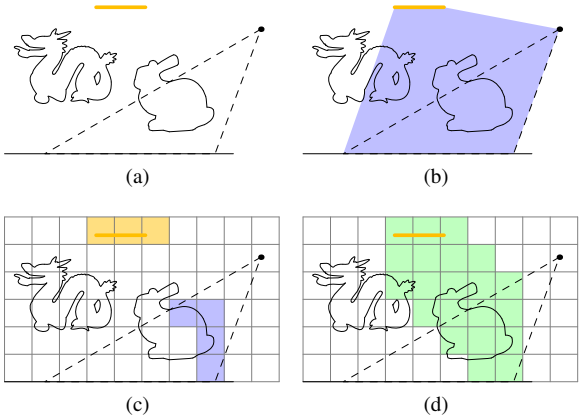

Figure 2: *The different steps involved in pre-BVH culling. (a) Basic scene setup, most fragments visible to the camera are part of the bunny. (b) Simply merging light and camera view is overly conservative. (c) Instead we introduce a uniform grid and tag all visible (blue) and light source (orange) cells. (d) Tagging the cells between all pairs of visible and light cells yields a conservative, but still much reduced estimate of the part of the scene relevant for tracing shadow rays (green).*

3. Pre-BVH Culling

In the following we describe our approach to culling irrelevant scene parts prior to the construction of a sub-optimal hierarchy. As a pre-process, this method not only increases ray traversal performance, but also provides for faster hierarchy construction. The outline of our algorithm is:

1. A very coarse uniform grid is introduced and cells containing directly visible scene geometry and light sources are tagged (see Section 3.1).
2. The relevant subset of the scene is determined by traversing this grid and tagging the encountered cells (see Section 3.2).
3. The hierarchy is built using only primitives of tagged cells (see Section 3.3).

Figure 1 shows how these steps can be added to a hybrid rendering system. In addition to taking the (dynamic) scene geometry as input, our method also requires the visible part of the scene that is usually available from an early-Z pass or GBuffer data. The result of soft shadows computed by ray tracing is then provided by a buffer holding the occlusion mask and is easily incorporated into the shading stage.

The remainder of this section provides details on the three steps shown above (which are also illustrated in Figure 2).

3.1. Coarse Uniform Grid

We base the selection of relevant scene geometry on a very coarse grid that encompasses the whole scene, or, for larger setups, the part of the scene that is to be considered. The key to this structure is that as long as the resolution is low, traversal of the grid will be very fast. We usually employ a resolution of 32^3 to 64^3.

We use this grid to track which part of the scene is considered relevant. If a grid cell is tagged relevant, then all geometry overlapping it will be considered relevant. Relevant, in our context, means that geometry cannot be removed as it may block shadow rays. Therefore our algorithm must generate a conservative estimate to ensure that no blocking geometry is removed. Note that the coarse resolution is not used to *approximate* geometry, but to *classify* it. Thus, coarser grid resolution will result in more conservative estimates of the relevant scene parts, while finer resolution will result in longer traversal times.

As seed of our algorithm we tag all cells containing geometry visible from the camera (e.g. by considering early-Z pass (EZP) data or a GBuffer), and denote them 'source cells.' We also tag all cells that overlap the light source, and denote them 'destination cells.' See Figure 2 (c) for an illustration. Initial tagging from EZP or GBuffer data and light extents is executed using CUDA and, as shown in Section 4, the required time is negligible.

3.2. Relevant Subset Selection

Based on the seeded grid, we traverse a 3D line from each source cell to each destination cell using a simple DDA [AW87], also implemented in CUDA. All cells encountered during this phase are marked as relevant, and to ensure a conservative estimate, we also tag the one-ring of each encountered cell. Figure 2 (d) shows the marked cells as seeded by Figure 2 (c). Note that by this we select a (non-convex) region of the grid, and the tagged cells encompass the subset of the scene that will not be left for any shadow ray.

Traversing the grid on the GPU is very fast (see Section 4) which is a direct result of using a very coarse grid. Larger grid resolutions result in much longer traversal times and exhibit the 'teapot in a stadium' problem. The default grid size

used in all our tests is 32^3; with this resolution the aforementioned effect is strongly limited and such sizes impose a very low upper limit on execution time of this step (see Section 4).

In the same vein, memory requirements are very low, allowing us to not pack the data tightly (we require 128 KB for a 32^3 grid using an unsigned integer per cell). The advantage of this format, in contrast to encoding the flags with one bit, is that no locking is required to set a single cell. Using a tightly packed structure (as would be required for higher resolution grids) would require complicated and expensive management to ensure no flags are overwritten by concurring threads. With this simpler layout (and while staying well below 1 MB) we can simply overwrite each cell that is encountered. As all changes to cells will be from 'irrelevant' to 'relevant' the execution order of those changes is inconsequential as it is guaranteed that one change will be executed cleanly in the end [NVI15].

3.3. Hierarchy Construction

After the relevant part of the scene has been established by tagging cells in the grid we invoke a kernel for each geometric primitive that checks the cells overlapping it for relevance. If one cell overlapping a primitive is tagged, then the primitive itself is tagged. Even though, due to the grid's low resolution, each primitive usually only overlaps very few cells, this is the most time consuming step of our algorithm (see Section 4).

We then scan over the primitive indices based on the tags and forward the selected triangles to the hierarchy construction step. In Section 4 we show how the construction and traversal performance of LBVH [LGS*09] and treelet-optimized LBVH [KA13] is affected by this scheme.

4. Results

To evaluate the impact of our BVH culling we compare how it affects the time-to-image for two recent construction algorithms targeting interactive rendering, namely LBVH [LGS*09], and LBVH optimized by treelet rotations [KA13]. LBVH allows to rapidly construct BVHs, however with inferior performance, while treelet rotations (a post process) require more time during construction, but achieve better traversal rates. These relationships can be seen in Table 1 and 2, in the sections LBVH and TL-LBVH, together with times when these structures are applied on preculled primitive lists. We show detailed times for the different culling-steps as described in Section 3 and only repeat the overall culling time when comparing to LBVH and TL-LBVH. We also show construction and render times for the CPU-based binned SAH construction after Wald [Wal07], labeled BSAH. This approach takes considerably more time during construction, but consistently outperforms the other

Figure 3: *The Sponza atrium illuminated by two area light sources, a large and subtle light from above and a smaller, stronger light from behind, (all images sampled at 15 spp). For this view our approach reduces the scene to spatially compact 45%. Times are listed in Table 1.*

Figure 4: *The same scene as above, for this view reduced to 60%, see also Table 1. (All images at* 1366×768*)*

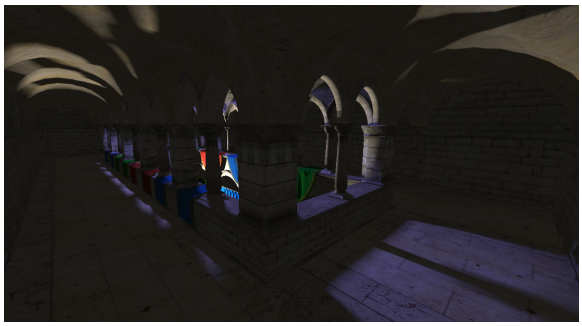

Figure 5: *Upper level of the atrium, reduced to 60%. Detailed times are shown in Table 2.*

	Figure 3	Figure 4
General:		
Render Gbuffer	2.158 ms	2.897 ms
Final blit	0.734 ms	0.710 ms
Culling details:		
Gbuffer → Grid	0.387 ms	0.710 ms
Mark cells (DDA)	0.122 ms	0.128 ms
Determine tri usage	0.638 ms	0.600 ms
Scan tri indices	0.231 ms	0.232 ms
Render with LBVH:		
Rebuild LBVH	3.520 ms	3.539 ms
Integrate 1spp	26.836 ms	15.872 ms
Render with LBVH culling:		
Sum culling	1.399 ms	1.405 ms
Rebuild culled LBVH	2.095 ms	2.428 ms
Integrate 1spp	20.955 ms	12.636 ms
Render with TL-LBVH:		
Rebuild TL-LBVH	23.316 ms	23.408 ms
Integrate 1spp	20.776 ms	11.862 ms
Render with TL-LBVH culling:		
Sum culling	1.399 ms	1.405 ms
Rebuild culled TL-LBVH	12.065 ms	15.185 ms
Integrate 1spp	17.653 ms	9.834 ms
Render with BSAH BVH:		
Rebuild SAH	278.500 ms	278.500 ms
Integrate 1spp	14.279 ms	8.428 ms

Table 1: *Detailed times for rendering the scenes shown in Figure 3 and 4. We compare two approaches for interactive BVH construction [LGS*09,KA13] with and without our pre-process and show the offline reference [Wal07]. Only one sample per pixel was taken for the traversal time, see Table 3 for how these values relate when more samples are taken.*

	Figure 5	Figure 6
General:		
Render Gbuffer	3.035 ms	4.554 ms
Final blit	1.539 ms	1.623 ms
Culling details:		
Gbuffer → Grid	0.344 ms	0.403 ms
Mark cells (DDA)	0.133 ms	0.126 ms
Determine tri usage	0.562 ms	1.570 ms
Scan tri indices	0.220 ms	0.251 ms
Render with LBVH:		
Rebuild LBVH	3.525 ms	9.902 ms
Integrate 1spp	12.684 ms	18.408 ms
Render with LBVH culling:		
Sum culling	1.288 ms	2.374 ms
Rebuild culled LBVH	2.687 ms	1.121 ms
Integrate 1spp	11.004 ms	14.160 ms
Render with TL-LBVH:		
Rebuild LBVH	23.293 ms	68.993 ms
Integrate 1spp	10.349 ms	10.540 ms
Render with TL-LBVH culling:		
Sum culling	1.288 ms	2.374 ms
Rebuild culled LBVH	16.678 ms	4.775 ms
Integrate 1spp	9.741 ms	9.329 ms
Render with BSAH BVH:		
Rebuild SAH	278.500 ms	879.400 ms
Integrate 1spp	8.372 ms	9.041 ms

Table 2: *This table shows the same analysis as presented in Table 1, but for the scenes shown in Figure 5 and 6.*

Figure 6: *A view into a larger cellar scene, illuminated by an area light from the left. Only 4% of the scene are used during hierarchy construction and tracing, see Table 2 for times.*

methods. However, it is also visible that applying treelet rotations to an LBVH greatly reduces the gap, especially when considering the much shorter build time.

From these tables it can further be seen that both, hierarchy construction and traversal benefit from culling, moving the LBVH traversal performance closer to TL-LBVH while not increasing the build time and pushing the TL-LBVH performance even closer to the BSAH reference, while even reducing the hierarchy construction time. Note that for all measurements we completely rebuild the respective structures.

Based on this data Table 3 shows how the different approaches scale with increasing sample count. The left part sums up Tables 1 and 2, followed by times for taking 10 and 100 samples per pixel. We compare the render times to that of using LBVH, a very popular structure for interactive tracing. Note that for most cases even after 100 samples per pixel the time-to-image using a standard high quality BVH builder (BSAH) has not caught up with approaches using our pre-process (especially in combination with treelet rotations). Figure 7 shows the same data as normalized to BSAH, and also gives the break-even points for the different interactive methods.

The impact of using different grid sizes is demonstrated in Table 4. It shows detailed times for culling, hierarchy construction and ray traversal averaged over all frames of the sequence shown in our accompanying video (where the scene consists of 3.5 million triangles). It can be seen that traversal times consistently decrease with larger grid sizes, but with dimishing gains towards the end. Furthermore, Table 4 shows that hierarchy construction benefits much more, reflecting the algorithm's much stronger sensitivity to input-size. However, with larger grid sizes the overhead introduced by culling, especially by the traversal of the grid, outweighs any gains in construction and traversal speed as at some resolution the well known 'teapot in a stadium' problem becomes clearly visible. Still, even at very low grid sizes our approach yields a net gain even when only a small number of samples are taken.

5. Conclusion

In this paper we introduced a novel pre-process to cull scenes primitives before hierarchy construction when using ray tracing to compute shadows in an interactive context. We showed how our method improves rendering performance even when only using very few (down to one) rays per pixel, as well as that it pays off for sample counts well beyond interactive rendering. The major limitation of our approach is that source and destination of each ray must be known in advance and thus the hierarchy cannot be used for generating, e.g., primary rays. For this reason, we construct the hierarchy after primary visibility has been resolved. In a purely ray tracing based pipeline this would incur the cost of an additional hierarchy. However, our focus is on hybrid rendering for which we have shown that our approach is well suited, and also seamlessly works together with other optimizations such as treelet rotations. This combination yields the best results of the interactive approaches we have investigated in this paper. We also believe that further investigation of pre-BVH culling, and pre- or post-processing of interactively built BVHs in general, is a promising research direction.

Acknowledgments

This work was supported by the German Research Foundation (GRK 1773).

References

[AAM03] ASSARSSON U., AKENINE-MÖLLER T.: A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Trans. Graph. 22*, 3 (July 2003), 511–520. 2

[ADM*08] ANNEN T., DONG Z., MERTENS T., BEKAERT P., SEIDEL H.-P., KAUTZ J.: Real-time, all-frequency shadows in dynamic scenes. *ACM Transactions on Graphics (Proc. SIGGRAPH) 27*, 3 (2008), 34:1–34:8. 2

[ALK12] AILA T., LAINE S., KARRAS T.: *Understanding the Efficiency of Ray Traversal on GPUs – Kepler and Fermi Addendum*. NVIDIA Technical Report NVR-2012-02, NVIDIA Corporation, June 2012. 1, 2

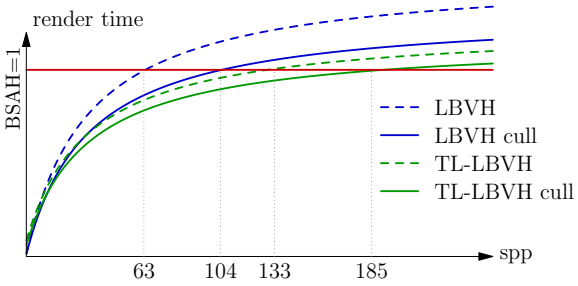

Figure 7: *Render times for increasing number of shadow samples per pixel normalized to ray tracing using BSAH. With large enough sample count the impact of hierarchy construction diminishes. However, while staying in the realm of interactive rendering sub-optimal hierarchies perform better regarding time-to-image.*

[AMB*07] ANNEN T., MERTENS T., BEKAERT P., SEIDEL H.-P., KAUTZ J.: Convolution shadow maps. In *Proc. EG Symposium on Rendering* (2007), pp. 51–60. 2

[AMS*08] ANNEN T., MERTENS T., SEIDEL H.-P., FLERACKERS E., KAUTZ J.: Exponential shadow maps. In *Proc. Graphics Interface* (2008), pp. 155–161. 2

[AW87] AMANATIDES J., WOO A.: A Fast Voxel Traversal Algorithm for Ray Tracing. In *In Eurographics âĂŹ87* (1987), pp. 3–10. 3

[BCS08] BAVOIL L., CALLAHAN S. P., SILVA C. T.: Robust soft shadow mapping with backprojection and depth peeling. *Journal of Graphics, GPU, and Game Tools 13*, 1 (2008), 19–30. 2

[Cro77] CROW F. C.: Shadow algorithms for computer graphics. *Computer Graphics (Proc. of SIGGRAPH) 11*, 2 (1977), 242–248. 2

[DHK08] DAMMERTZ H., HANIKA J., KELLER A.: Shallow bounding volume hierarchies for fast simd ray tracing of incoherent rays. *Comput. Graph. Forum 27*, 4 (2008), 1225–1233. 2

[DL06] DONNELLY W., LAURITZEN A.: Variance shadow maps. In *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2006), pp. 161–165. 2

[EG08] ERNST M., GREINER G.: Multi Bounding Volume Hierarchies. In *IEEE Symposium on Interactive Ray Tracing* (Aug. 2008), pp. 35–40. 2

[ESAW11] EISEMANN E., SCHWARZ M., ASSARSSON U., WIMMER M.: *Real-Time Shadows*. A.K. Peters, 2011. 2

[FBP06] FOREST V., BARTHE L., PAULIN M.: Realistic Soft Shadows by Penumbra-Wedges Blending. In *Graphics Hardware* (2006), Olano M., Slusallek P., (Eds.), The Eurographics Association. 2

[Fer05] FERNANDO R.: Percentage-closer soft shadows. In *ACM SIGGRAPH Sketches* (2005). 1, 2

[GBP06] GUENNEBAUD G., BARTHE L., PAULIN M.: Real-time soft shadow mapping by backprojection. In *Proc. EG Symposium on Rendering* (2006), pp. 227–234. 1, 2

[GBP07] GUENNEBAUD G., BARTHE L., PAULIN M.: High-quality adaptive soft shadow mapping. *Computer Graphics Forum (Proc. of Eurographics) 26*, 3 (2007), 525–534. 2

[KA13] KARRAS T., AILA T.: Fast parallel construction of high-quality bounding volume hierarchies. In *Proc. High-Performance Graphics* (2013). 2, 4, 5

	Cull	Build	Trace	Σ	to LBVH	Σ (10spp)	to LBVH	Σ (100spp)	to LBVH
Figure 3									
LBVH	–	3.520	26.836	30.356	100.0%	271.880	100.0%	2687.120	100.0%
LBVH culled	1.399	2.095	20.955	24.449	80.5%	213.044	78.3%	2098.994	78.1%
TL-LBVH	–	23.316	20.776	44.092	145.2%	231.076	84.9%	2100.916	78.1%
TL-LBVH culled	1.399	12.065	17.653	31.117	102.5%	189.994	69.8%	1778.764	66.1%
BSAH	–	278.500	14.279	292.779	964.4%	421.290	154.9%	1706.400	63.5%
Figure 4									
LBVH	–	3.539	15.872	19.411	100.0%	162.259	100.0%	1590.739	100.0%
LBVH culled	1.405	2.428	12.636	16.469	84.8%	130.193	80.2%	1267.433	79.6%
TL-LBVH	–	23.408	11.862	35.270	181.7%	142.028	87.5%	1209.608	76.0%
TL-LBVH culled	1.405	15.185	9.834	26.424	136.1%	114.930	70.8%	999.990	62.8%
BSAH	–	278.500	8.428	286.928	1478.1%	362.780	223.5%	1121.300	70.4%
Figure 5									
LBVH	–	3.525	12.684	16.209	100.0%	130.365	100.0%	1271.925	100.0%
LBVH culled	1.228	2.687	11.004	14.919	92.0%	113.955	87.4%	1104.315	86.8%
TL-LBVH	–	23.293	10.349	33.642	207.5%	126.783	97.2%	1058.193	83.1%
TL-LBVH culled	1.228	16.678	9.741	27.647	170.5%	115.316	88.4%	992.006	77.9%
BSAH	–	278.500	8.372	286.872	1769.8%	362.220	277.8%	1115.700	87.7%
Figure 6									
LBVH	–	9.902	18.408	28.310	100.0%	193.982	100.0%	1850.702	100.0%
LBVH culled	2.374	1.121	14.160	17.655	62.3%	145.095	74.7%	1419.495	76.7%
TL-LBVH	–	68.993	10.540	79.533	280.9%	174.393	89.9%	1122.993	60.6%
TL-LBVH culled	2.374	4.775	9.329	16.478	58.2%	100.439	51.7%	940.049	50.7%
BSAH	–	879.400	9.041	888.441	3138.2%	969.810	499.9%	1783.500	96.3%

Table 3: *With increasing number of samples investing more time in constructing better hierarchies helps improve time-to-image. At 100 spp offline BVH construction has caught up with interactive approaches for almost all our examples, except when using culling together with treelet rotations to optimize LBVH performance. Note that our culling pre-process pays off even at 1 spp in all cases.*

	16^3	32^3	64^3	128^3	256^3
Culling details:					
Gbuffer → Grid	0.319 ms	0.463 ms	0.658 ms	0.904 ms	2.995 ms
Mark cells (DDA)	0.074 ms	0.133 ms	0.704 ms	11.880 ms	534.021 ms
Determine tri usage	4.323 ms	4.810 ms	5.884 ms	8.515 ms	16.388 ms
Remaining triangles	34 %	24 %	19 %	17 %	16 %
Render with LBVH:					
Rebuild LBVH	42.161 ms				
Integrate 1spp	17.020 ms				
Render with LBVH culling:					
Sum culling	4.716 ms	5.406 ms	7.246 ms	21.299	553.404 ms
Build culled LBVH	15.464 ms	11.450 ms	9.809 ms	8.989 ms	8.458 ms
Integrate 1spp	15.430 ms	13.915 ms	13.307 ms	12.889 ms	12.538 ms
Render with TL-LBVH:					
Build TL-LBVH	261.186 ms				
Integrate 1spp	12.167 ms				
Render with TL-LBVH culling:					
Sum culling	4.716 ms	5.406 ms	7.246 ms	21.299	553.404 ms
Build culled TL-LBVH	95.601 ms	67.056 ms	56.057 ms	49.942 ms	46.145 ms
Integrate 1spp	11.496 ms	10.660 ms	10.354 ms	10.095 ms	9.874 ms

Table 4: *Detailed culling, BVH construction and rendering times with different grid sizes for the larger scene shown in the accompanying video, averaged over all frames. For readability the times of the non-culled versions are not repeated for all grid sizes.*

[Kar12] KARRAS T.: Maximizing parallelism in the construction of bvhs, octrees, and k-d trees. In *Proceedings of the EUROGRAPHICS Conference on High Performance Graphics 2012, Paris, France, June 25-27, 2012* (2012), pp. 33–37. 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 the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2012), I3D '12, ACM, pp. 197–204. 2

[KKW*13] KELLER A., KARRAS T., WALD I., AILA T., LAINE S., BIKKER J., GRIBBLE C. P., LEE W., MCCOMBE J.: Ray tracing is the future and ever will be.. In *International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2013, Anaheim, CA, USA, July 21-25, 2013, Courses* (2013), pp. 9:1–9:7. 1, 2

[LGS*09] LAUTERBACH C., GARLAND M., SENGUPTA S., LUEBKE D., MANOCHA D.: Fast BVH Construction on GPUs. *Computer Graphics Forum* (2009). 1, 2, 4, 5

[LM07] LAURITZEN A., MCCOOL M.: Summed-area variance shadow maps. *GPU Gems* (2007), 157–182. 2

[LM08] LAURITZEN A., MCCOOL M.: Layered variance shadow maps. In *Proc. Graphics Interface* (2008), pp. 139–146. 2

[MB90] MACDONALD D. J., BOOTH K. S.: Heuristics for ray tracing using space subdivision. *Vis. Comput. 6*, 3 (May 1990), 153–166. 2

[NVI15] NVIDIA CORPORATION: NVIDIA CUDA C programming guide, 2015. Version PG-02829-001_v7.0. 4

[PL10] PANTALEONI J., LUEBKE D.: Hlbvh: Hierarchical lbvh construction for real-time ray tracing of dynamic geometry. In *Proceedings of the Conference on High Performance Graphics* (Aire-la-Ville, Switzerland, Switzerland, 2010), HPG '10, Eurographics Association, pp. 87–95. 2

[SDMS14] SELGRAD K., DACHSBACHER C., MEYER Q., STAMMINGER M.: Filtering multi-layer shadow maps for accurate soft shadows. *Computer Graphics Forum* (2014), 205–215. 2

[SFD09] STICH M., FRIEDRICH H., DIETRICH A.: Spatial splits in bounding volume hierarchies. In *Proceedings of the Conference on High Performance Graphics 2009* (New York, NY, USA, 2009), HPG '09, ACM, pp. 7–13. 2

[SFY13] SHEN L., FENG J., YANG B.: Exponential soft shadow mapping. *Computer Graphics Forum (Proc. EG Symposium on Rendering) 32*, 4 (2013), 107–116. 1, 2

[SS07] SCHWARZ M., STAMMINGER M.: Bitmask soft shadows. *Computer Graphics Forum (Proc. Eurographics) 26*, 3 (2007), 515–524. 2

[Wal07] WALD I.: On fast construction of sah-based bounding volume hierarchies. In *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing* (Washington, DC, USA, 2007), RT '07, IEEE Computer Society, pp. 33–40. 2, 4, 5

[WBS07] WALD I., BOULOS S., SHIRLEY P.: Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Trans. Graph. 26*, 1 (Jan. 2007). 2

[Whi80] WHITTED T.: An improved illumination model for shaded display. *Commun. ACM 23*, 6 (June 1980), 343–349. 2

[WHL15] WYMAN C., HOETZLEIN R., LEFOHN A.: Frustum-traced raster shadows: Revisiting irregular z-buffers. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2015), i3D '15, ACM, pp. 15–23. 1

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *Computer Graphics (Proc. of SIGGRAPH) 12*, 3 (1978), 270–274. 2

[WP12] WOO A., POULIN P.: *Shadow Algorithms Data Miner*. A K Peter/CRC Press, 2012. 2

[WWB*14] WALD I., WOOP S., BENTHIN C., JOHNSON G. S., ERNST M.: Embree: A kernel framework for efficient cpu ray tracing. *ACM Trans. Graph. 33*, 4 (July 2014), 143:1–143:8. 2

[XTP07] XIE F., TABELLION E., PEARCE A.: Soft shadows by ray tracing multilayer transparent shadow maps. In *Proc. EG Symposium on Rendering* (2007), pp. 265–276. 2

[YDF*10] YANG B., DONG Z., FENG J., SEIDEL H.-P., KAUTZ J.: Variance soft shadow mapping. *Computer Graphics Forum 29*, 7 (2010), 2127–2134. 2