Soft Shadow Computation using Precomputed Line Space Visibility Information

Kevin Keul
Department of Computer Graphics, Institute for Computational Visualistics, University of Koblenz-Landau, Koblenz, Germany
keul@uni-koblenz.de

Nicolas Klee
Department of Computer Graphics, Institute for Computational Visualistics, University of Koblenz-Landau, Koblenz, Germany
nklee@uni-koblenz.de

Stefan Müller
Department of Computer Graphics, Institute for Computational Visualistics, University of Koblenz-Landau, Koblenz, Germany
stefanm@uni-koblenz.de

ABSTRACT

Shadows are one of the most important effects to create realism in rendering. Most real-time applications use some sort of image based technique like shadow mapping. While these techniques are quite fast, they often struggle at rendering realistic and accurate shadows of area lights. To produce correct shadows it is therefore often necessary to use ray tracing with some sort of acceleration method, nowadays mostly GPU based BVH which have their downsides in real-time rendering. We present a novel approach in calculating approximated but fast shadows using the line space as precomputed data structure for visibility information. With that it is possible to skip intersection tests with scene geometry and completely rely on the line space data structure for the shadow computations of area lights. Our approach is therefore almost scene-independent and is able to produce accurate shadows with better performance in comparison to typical ray tracing data structures.

Keywords

Visualization, Computer Graphics, Ray Tracing, Data Structures, Visibility Algorithms

1 INTRODUCTION

Computing shadows is one of the most important ways to enhance realism in a scene. Shadows increase the spatial perception and with that the overall appearance of realistic rendering results. A simple way to compute shadows is to compute the distance of the foremost objects to a point light source first and store those in a shadow map. This map can then be used to differentiate occluded from lighted objects. This approach can be applied in combination with typical rendering and is therefore useful for exploitation of the parallel nature of the graphics processing unit (GPU). With that it is fast and gives realistic results for point light sources. But in most cases area lights are favoured because of better quality in realistic scenes and shadow mapping techniques fail to produce fast shadows of those with good quality. Therefore in most approaches some sort of ray tracing method is used to approximate the surface of the area light source with multiple samples. While the

Figure 1: Soft Shadow computation by using 49 Shadow rays for a static scene. The scene is rendered with typical forward rendering while the shadows are computed with our ray tracer using early line space termination. The usage of the line space grants better performance compared to an equivalent BVH-based ray tracer with similar quality.

results of these techniques have a good and physically plausible quality, the computation needs quite a lot of time even with good acceleration data structures.

We propose a novel approach to compute approximate shadows using the line space as acceleration data structure. By using the line space it is possible to precompute visibilities which are used in our method for blocker calculation. With this we do not need to test the actual scene geometry for intersection, but we use approximate occlusions based on the shaft informations of the line space. This increases the rendering performance and allows us to only store the line space data structure in GPU memory with no need of storing any geometry information at all. One downside of our method is that the produced shadows are not precise because of the approximations with shafts. By using the line space for area lights we are able to show that these inaccuracies are negligible. Our results demonstrate that the use of the line space leads to a faster method compared to other ray tracing data structures and better quality compared to typical image based techniques.

Our main contributions are:

- An approximate technique for shadow computation using the line space as termination criterion.
- An acceleration for rendering approximate soft shadows of static scenes in real-time on modern GPUs.
- An analysis and comparison of the benefits of our technique.

2 RELATED WORK

Shadow Methods Rendering of shadows is a well researched topic and we will only give a brief overview of recent and relevant work. For further information we refer the reader to [Eis11] and [Has03].

Many methods exist for the task of rendering shadows. Starting with the work by [Wil78] there have been many approaches to image based methods. There, the occluding objects are stored in a so called shadow map first. In a second pass it is possible to determine with only one texture lookup of the shadow map which objects are visible from the light source. Lighting therefore has to be computed for exactly these objects, while all non-visible objects from the light source have to be shaded. This standard process of shadow mapping is fast but tends to have visible aliasing artifacts if the resolution of the shadow map is not big enough. In this form, it is only possible to produce hard shadows, where the light source has no volume at all but is only represented by a single point in space.

Percentage closer filtering [Ree87] is one possibility to reduce the problem with aliasing through blurring of the shadow edges. It works by taking not only one but multiple nearby texture lookups of the shadow map and using this to calculate the percentage of visibility from the light source. With adjustments to this it is possible to approximate soft shadows from area light sources [Fer05]. There, the size of the filter kernel is adjusted according to the distance of the occluder. With this approximation the shadow is not physically accurate but the results are sufficient in many cases.

Other concepts to create shadows are geometry based methods based on the generation of shadow volumes that enclose the shadowed space [Cro77]. It is possible to create correct shadows for point lights with hard shadow edges but it also needs some adjustments to create soft shadows with this idea [Ass03] [Lai05]. In general, image based methods using some kind of shadow mapping algorithm are more popular in comparison to geometry based methods. This is due to performance reasons and a greater versatility and applicability of shadow mapping algorithms, but both approaches can benefit from rasterization and are therefore fast.

Ray Tracing Methods A different approach to compute shadows is usually done with some kind of ray tracing algorithm. For each point that has to be tested for lighting a ray is constructed starting in that point and ending in the light source. If the ray is not intersecting scene geometry on this path then the point is lit, otherwise it is shadowed. This approach is more versatile in comparison to the previous ideas but the calculation of the intersection between rays and scene geometry is rather slow. Among the most popular and effective acceleration data structures for this are bounding volume hierarchies (BVHs) because of good performance [Sti09][Ail09]. Recently, there have been approaches to efficiently build good BVHs on the GPU [Kar12][Kar13]. Construction can be parallelized for example with SAH binned methods [Wal07][Wal12] or by using linear BVHs [Lau09]. This way it is possible to produce interactive results for construction and traversal of the data structure on GPUs. Extensions like multi bounding volume hierarchies are further exploiting the parallelization to get better performance by storing more subnodes per node than usual [Ern08][Wal08][Áfr14].

While all previous approaches use the scene geometry to test for intersections, there are algorithms that avoid that. Among the most popular are sparse voxel octrees (SVOs) where far away nodes are used to compute shadow occlusions [Gob05]. Efficient SVOs where proposed that use contours in order to decide whether the subdivision of a node can stop [Lai11]. With this it is possible save memory. Compression methods for SVOs were introduced in [Käm13], which can be precomputed [Sin14]. It was shown that the construction and

the traversal speed are fast enough to be used in real time applications [Cra11].

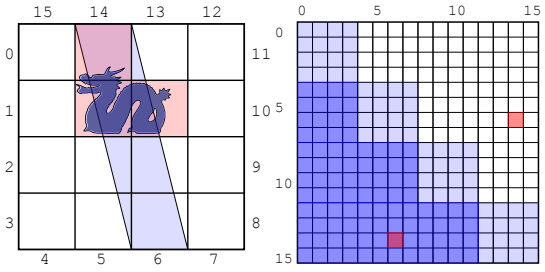
Visibility Precomputation Moreover, there have been approaches that precompute visibilities for example in radiosity calculations. Line space computations were used by [Dre97], where they compute shafts between two arbitrary surface elements. Those shafts represent all possible rays between the corresponding surface elements and thereby visibilities can be precomputed. This approach was recently applied to the N-Tree [Keu16], which is a variation of recursive grids [Jev88]. In this attempt every subdivided node has the line space information as well, which summarizes every possible shaft in this node and shows which shafts are empty or non-empty. This information is used during traversal. Precomputed visibilities for urban scenes were proposed by [Bit01] and [Ley03]. They also use the term "line space" but in a different meaning compared to approaches mentioned above.

3 LINE SPACE INFORMATION

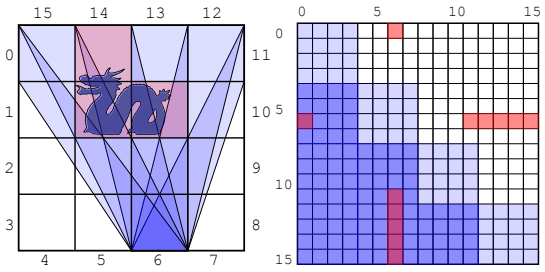
Our goal is to compute shadows without testing the scene geometry for intersection. For this, we create a data structure with the help of the scene geometry which does not need the geometry afterwards. In that our data structure is similar to sparse voxel octrees as they are for example used in voxel cone tracing [Cra11]. In contrast to octress we do not store leaf-nodes containing scene geometry at all. Instead we store a line space in every subdivided (= non-leaf) node and ignore the deepest level of the tree as proposed by [Keu16]. We therefore have approximated shadows comparable to those of sparse voxel octrees. By using the direction based data of the line space and its early termination criterion (which is explained later on) we are able to accelerate the shadow computation even further.

A line space contains the visibility information for every possible shaft within the corresponding node. A shaft is expressed through a given start and end side from the nodes surface. In the N-Tree every node is axis aligned and can be represented with an axis aligned bounding box (AABB). Every subdivided node has precisely $N \times N \times N$ subnodes, so the surface of each side of the AABB is subdivided in $N \times N$ subsides. Those subsides serve as the start and end sides for the shafts. A shaft therefore is able to group all rays starting and ending at specific subsides of the AABB of the node.

It can be precomputed which subnodes of the current node are intersected by a certain shaft. If all intersected subnodes are empty, so they do not contain any geometry of the scene geometry at all, it is possible to conclude that all possible rays within this shaft are unable to intersect scene geometry within this node and all potentially intersected subnodes. For this, a shaft only needs the information if all intersected subnodes



(a) Shaft starting from 6 and ending in 14



(b) All non-empty shafts starting from start side 6

Figure 2: Illustration of the the N-Tree and part of the according line space in 2D. In the upper images one non-empty shaft and the associated entry in the line space are shown. In the lower images all non-empty shafts starting from one specific start side are shown. The light blue entries in the line space represent shafts that start and end on the same sides of the bounding box, while the dark blue entries are symmetric information and therefore unnecessary. Note that every red node in the N-Tree may be subdivided as well and would therefore contain line spaces on their own.

are empty or if there is at least one non-empty subnode intersecting the shaft. This can be expressed in one bit of information. The line space contains this information for all possible shafts of one node and can be represented as a 2D array or texture where the first dimension stands for the start side and the second dimension for the end side of a shaft.

Figure 2 shows an example of the line space information, where it is shown that the entries of this array are symmetrical around the diagonal because of the inversion of start and end sides of the shafts. Shafts that start and end on the same side of the AABB do not contain any volume at all and are therefore always empty which is observable in the empty squares around the diagonal. Keeping this in mind, the necessary size of the line space can be reduced to less than a half.

As with other data structures finding the correct settings is important. The line space has two essential parameters: the maximum depth and the branching factor of the underlying N-Tree. The maximum depth limits the maximum number of nodes. Only subdivided nodes contain a line space. So in our case the deepest level in the tree is not needed after the initialization anymore. According to [Keu16] the branching factor

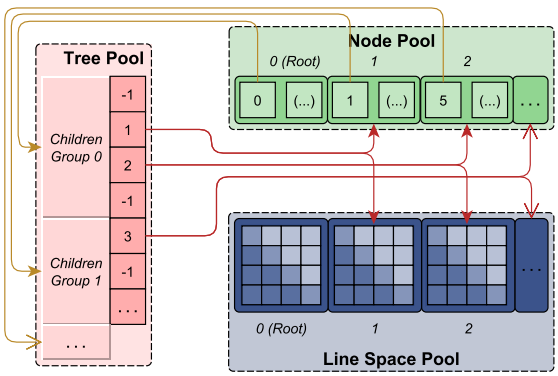


Figure 3: Management of the GPU data structure. The entries of the tree pool are clustered in children groups of nodes. Each entry of this pool is either set to a default value or refers to a subdivided node (stored in the node pool) and its corresponding line space (stored in the line space pool). Each node in the node pool contains the necessary data for its traversal and therefore a reference to its children group which can be traversed recursively.

N has a significant impact on the shape of the shafts and therefore on the traversal speed. A high branching factor leads to long and slim shafts which are good for skipping many subnodes during traversal at once. On the other hand the number of entries within the 3D line space is $15 \times N^4$, so a high value for the branching factor results in a huge memory consumption. For good traversal results it was stated that the optimal value for the branching factor is between 6 and 10 and for the depth is either 3 or 4.

3.1 GPU Data Structure

Adapting the idea of [Cra11] we implement our data structure in data pools. We use three different pools, which are stored in linear buffers on the GPU. In the first data pool (the tree pool) we store the tree information of the N-Tree where all node relations are ordered in groups of subnodes. In the second data pool (the node pool) the information for all subdivided nodes is stored. This information is used to compute the traversal of the subnodes of one node. The third data pool (the line space pool) is used for all line space information of the subdivided nodes. This concept is illustrated in figure 3. We implemented our approach with OpenGL Compute Shaders and therefore optimized all storage units for this.

The data structure is used in a way to only rely on subdivided nodes for traversal. Leaf nodes containing scene geometry are not needed and therefore not stored within the data pools. The tree pool consists of all possible pointers that are needed to represent the hierarchy. The order of the pointers is based on groups of children of subdivided nodes. All children of one node are clustered to one children group. They have a specific inter-

Figure 4: Illustration of the structure of one N-Tree node. It consists of different attributes which are used for the traversal of the tree. All subdivided nodes have a reference to the children group index of this node. The reference is used in combination with this nodes position and additional information like its resolution for the traversal of its subnodes. If it has no subdivided children then the reference is set to a default value, indicating that this node can be skipped during traversal.

nal order, which is the same for all children groups and dependant on the local position within the parent node. If a children node is subdivided then the pointer is set to the index of the children within the node pool. If the children is not subdivided then the pointer is set to a default value, indicating that the traversal can skip this node. Instead of using a default value it is also possible to use negative values with a special meaning. With this it is possible to also store references to geometry information if needed.

The node pool is also used for the traversal. During traversal of the line space leaf nodes are skipped completely. For efficient memory usage therefore only subdivided nodes are stored within the node pool. A node within the node pool consists of different attributes as illustrated in figure 4 which are used for the traversal. The main attribute of a node is a reference pointer to its children group within the tree pool. Other information needed for the traversal are the position of this node in world space and its size. It is possible to store additional information of a node like its resolution for the case that nodes can have a variable number of children nodes.

The data of the line space pool is used as termination criterion in the traversal. The single bit information whether a shaft is empty is stored in this pool. For better incorporation with GPU-memory, the information of multiple shafts is combined to an integer value. The partition of the pool correlates with the node pool, so the n-th node in the node pool is related to the n-th line space in the line space pool. With this a pointer of the tree pool simultaneously refers to the corresponding node and its line space as shown in figure 3. While a line space, as explained above and shown in the figure, is illustrated as a two-dimensional data set, it is in fact implemented as one-dimensional data within the buffer. It consists of a sequence of combined integer values and is therefore stored in an efficient way.

4 SHADOW COMPUTATION

Using the line space we have a data structure that is able to decide whether a ray is probably going to intersect scene geometry through a given shaft or if it is

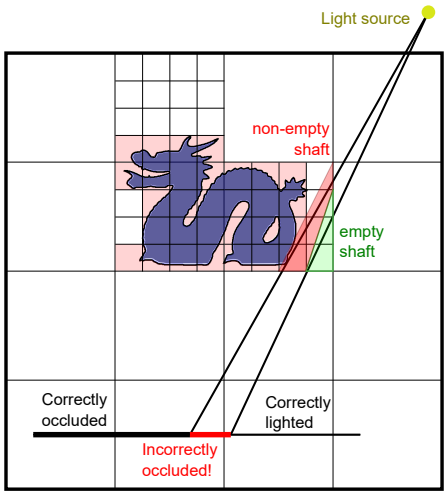


Figure 5: Illustration of the line space used for shadow computation. The object on the bottom is partially occluded by the dragon. While the occlusion on the left of the object is correct, the usage of the line space also results in incorrect occlusion. This inaccuracy is based on non-empty shafts in which the shadow rays would normally pass by the scene geometry without intersecting it, but are wrongly classified as occluded. Note that this inaccuracy can be greatly reduced by using a higher branching factor for the underlying N-Tree.

definitely not going to intersect anything. Visibility of a light source in this context is therefore merely an approximation for the pure possibility of visibility. If a shaft only intersects empty subnodes, it is called empty itself. An empty shaft does not change the possibility of visibility and therefore the light source counts as "visible". If a shaft intersects at least one non-empty subnode, then the shaft is called non-empty. A non-empty shaft may be able to block the visibility of a light source and as a result we classify this shaft as "occluding". Note that this is a rather conservative estimation of occlusion. A ray to a light source within a shaft may be declared as occluded even though it may pass by the scene geometry contained in the subnodes. An example to this is shown in figure 5. Although this approximation may result in places that are wrongly occluded, this technique allows for a quick shadow traversal without the need to test the actual scene geometry for intersection. The benefit is not only that the test for occlusion is faster than the typical occlusion test in ray tracing, but furthermore it is not necessary to store the scene geometry in the data structure at all. With this our data structure is mostly scene-independent.

4.1 Traversal

Normally the traversal for shadow computation traverses through the data structure until a node with scene geometry is found. This geometry is then tested for intersection. If an intersection is found within the node, then the traversal can end with a positive result. Otherwise it has to continue until an intersection is found or the data structures boundary is reached. Depending on the data structure the tests for intersection with scene geometry may be more costly than the traversal of the data structure itself.

In our case we try to use this fact in two ways. On the one hand we have an implicit intersection test of the scene geometry which is done with the help of the shaft information in the line space. This shaft information is precomputed during initialization of the line space, so no intersection tests with scene geometry have to be done during rendering. On the other hand we try to accelerate the traversal of the data structure by skipping the deepest level of the hierarchy by using the shafts as a summary of the underlying deepest level.

Algorithm 1 The recursive line space traversal algorithm for shadow computation

```
1:  procedure TRAVERSE(Ray r, Node n)
2:      if n has subnodes and LS(r, n) is true then
3:          if n is deep enough in hierarchy then
4:              return true
5:          else
6:              while subnodes left do
7:                  s ← next subnode in direction of r
8:                  if s is non-empty then
9:                      if TRAVERSE(r, s) then
10:                         return true
11:                     end if
12:                 end if
13:             end while
14:         end if
15:     end if
16:     return false
17: end procedure
```

The final traversal algorithm is shown in algorithm 1. All nodes in the data structure are either subdivided and contain a line space or are leaf nodes and contain scene geometry. The latter are not needed during traversal as explained above, so this is checked first. If the node is subdivided then the entry for the ray within the line space of this node is checked (line 2). If the entry is not set, it means that the according shaft of this ray is empty and therefore the ray is not able to hit anything within this node. Further inspection of this node can be skipped. If the entry is set, so the according shaft is non-empty, then the traversal of this node continues. Next it is tested whether the current node is deep enough in the tree hierarchy (line 3). A node is called deep enough, if it is on the second to last level in the hierarchy, so all subnodes of this node are leaf nodes. With this it may be possible that scene geometry is intersected by the ray and the ray gets accepted as occluded. Note that this is the part where the shadow is approximated. If

the current recursion depth is not yet deep enough then the subnodes intersecting the ray are being recursively tested for intersection (line 9).

A drawback using the line space without scene geometry is that it does not work if the ray starts within a shaft. The information stored within this shaft can not be applied to the ray because of the uncertainty how the objects within the shaft may be positioned in relation to the ray. Therefore the node where the ray starts has to be skipped in the process and the traversal needs to start with the next node. This leads to a loss in quality in detailed areas.

4.2 Soft Shadow Computation

Using the line space as termination criterion for the traversal has the benefits of a faster occlusion test and it may need less memory space. The downside to this is the loss in accuracy which comes from the approximation of the scene geometry with shafts as explained above. This effect is observable at the edges of the shadow regions where in general the line space produces more occluded areas as other approaches (see figure 5). Then again in most cases there are no point lights required but area lights, which do normally not produce hard shadow edges but soft transitions between occluded and non-occluded areas. Most approaches try to generate this effect by using multiple samples of the area light and calculating the percentage of non-occluded samples for lighting. By using this technique combined with the line space, the approximative nature of line space generated shadows become negligible (see figure 7).

Though the shadows generated by this are overly shadowed, the difference is almost not visible, especially when many samples are used. In addition the inaccuracy caused by the line space is especially less significant the bigger the area light sources become. This is due to the fact that less samples near the geometry edge are falsely occluded by the line space. An example of soft shadows generated by the line space is shown in figure 6. By only testing the visibility based on shafts and not on the actual scene geometry the traversal of the shadow rays is also more coherent and therefore better suitable for parallelization with the GPU.

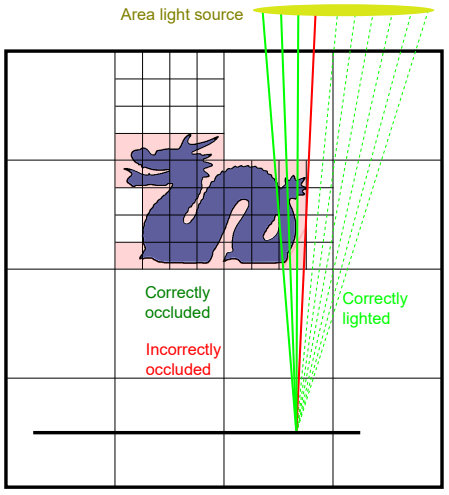


Figure 6: Illustration of the line space used for soft shadow computation. In this example 10 samples of the light source are used to calculate occlusion, while one of those samples is wrongly occluded. Note that as with normal shadows the accuracy of soft shadows based on the line space can be greatly reduced by using a higher branching factor for the underlying N-Tree.

5 RESULTS

We implemented our approach on a NVidia GeForce GTX 1080 system with an Intel i7-6800k 3.6 GHz CPU and used GLSL Compute Shaders for GPU computing. As test scenes we used typical test models with different numbers of triangles and different characteristics (Bunny 69k triangles, Dragon 871k triangles, Buddha 1087k triangles and Dragon and Buddha combined) on top of a simple plane. All scenes were rendered with a resolution of 1024×1024. While the geometry is rendered using typical forward rendering first, the shadows are applied using one of the techniques mentioned. We use different camera angles and take the average run time as result. We compared our method with the state of the art in BVH accelerated ray tracing using the GPU as proposed by Aila and Laine [Ail09].

The size of the line space and the build time varies significantly with the branching factor and the maximum tree depth used for the underlying N-Tree. As supposed by [Keu16] we use a branching factor between 4 and 8 with a maximum recursion depth of either 3 or 4. The line space is constructed on the GPU on top of the previously build N-Tree. As with most other complex ray tracing data structures, we do not achieve interactive initialization timings. The number of nodes containing a line space and the resulting size of the data structure with different parameter sets are given in table 1. There, the build timings for the line space on top of the N-Tree and the rendering timings are shown as well.

In comparison to BVH based ray tracing, the line space does not need any intersection test with scene geometry. With this, it has a substantially better performance in computing soft shadows. The results may differ significantly in scenes that do not fit in the GPU memory, but this needs to be further investigated and is beyond the scope of our work. It is visible that the rendering performance of the line space in comparison to the BVH is better in medium and big scenes, whereas the BVH achieves faster results in the small scene. Overall it is visible that the quality of the BVH in terms of performance is mainly influenced by the number of triangles

Scene		BVH	LS (4, 4)	LS (5, 4)	LS (6, 3)	LS (7, 3)	LS (8, 3)
BUNNY	size (MB)	4,3	9,4	92	12,3	45,6	115,2
(69k tris)	nodes (in 1000)	31,4	13	57,2	3,9	8,1	12,4
	init time (ms)	-	28	489	100	547	1779
	render (FPS)	95,3	58,7	52,5	69,9	62,2	59,6
DRAGON	size (MB)	35	9,9	97,7	13,1	48,7	122,2
(871k tris)	nodes (in 1000)	82,9	13,8	60,7	4,1	8,7	13,2
	init time (ms)	-	34	533	105	558	1788
	render (FPS)	22,4	23,7	21,1	33,6	32,2	29,8
BUDDHA	size (MB)	41,6	9,4	67,1	12,1	35,4	95,7
(1087k tris)	nodes (in 1000)	69,9	13,1	41,7	3,8	6,3	10,3
	init time (ms)	-	30	355	98	397	1432
	render (FPS)	16,4	47,5	42,6	60,8	57,3	19,7
BUDDHA &	size (MB)	76,6	12,7	89,4	15,8	46,2	127,6
DRAGON	nodes (in 1000)	152,7	17,7	55,6	5	8,2	13,7
(1959k tris)	init time (ms)	-	42	475	131	523	1928
	render (FPS)	12,7	23,6	22,4	26,6	24,9	24,3

Table 1: Comparison of the size in MB, number of subdivided nodes, build time in ms and rendering times in frames per second for different parameter sets of the line space and BVH based on the work by Aila and Laine [Ail09]. Every parameter set of the line space is given as (N, d), where N stands for the branching factor and d for the maximum depth of the used N-Tree. For the rendering we measured the time to compute soft shadows of one area light source with 25 shadow rays with an image resolution of 1024×1024. It is visible that the line space has better rendering performance in medium and big scenes, but not in scenes with only a small number of triangles.

used in the scene. In contrast, the performance results of the line space are as expected more stable with varying numbers of scene triangles, but are more influenced by the spatial structure of the scene. This is due to the fact that the line space does not store the scene geometry in any kind, but an approximation of the scene within the abstraction of the shafts.

In terms of memory consumption the BVH is mainly affected by the number of scene triangles which have to be stored in addition to the node information of the hierarchy. In contrast the line space does not need to store any triangle information at all and it can only rely on the node and line space data. Figure 8 shows the memory consumption for the mentioned scenes for the BVH and the line space with the tested parameter sets. The size of the line space significantly depends on the used parameter set, where a higher value of the branching factor N or the maximum tree depth d leads to a much higher memory consumption. As with the rendering performance, it is visible that the memory size of the line space is nearly scene independent and quite stable over all used scenes. With this it is possible to give an early estimation of the memory size for a given parameter set before actually building it.

As explained before, the usage of the line space for shadow generation is flawed with approximations due to the abstraction with the shafts. This is especially visible when only one shadow ray is used per pixel. In soft shadow computation, this problem is only barely noticeable and can be reduced by increasing the parameter set values used. Figure 7 illustrates this effect.

(a) 1 Shadow Ray to center of light

(b) 25 Shadow Samples to area light

Figure 7: Rendering results for the Dragon Scene using a single shadow ray (top) and 25 shadow samples (bottom). The shadows in the left images are computed with the line space with a low valued parameter set for illustration (a branching factor N of 5 and tree depth d of 3) while the shadows in the right images are computed with a BVH as ground truth data. Using the low values in the line space parameter set leads to a bad shadow silhouette in the case that 1 shadow ray is used. In soft shadow computation the difference to the ground truth data is nearly not visible.

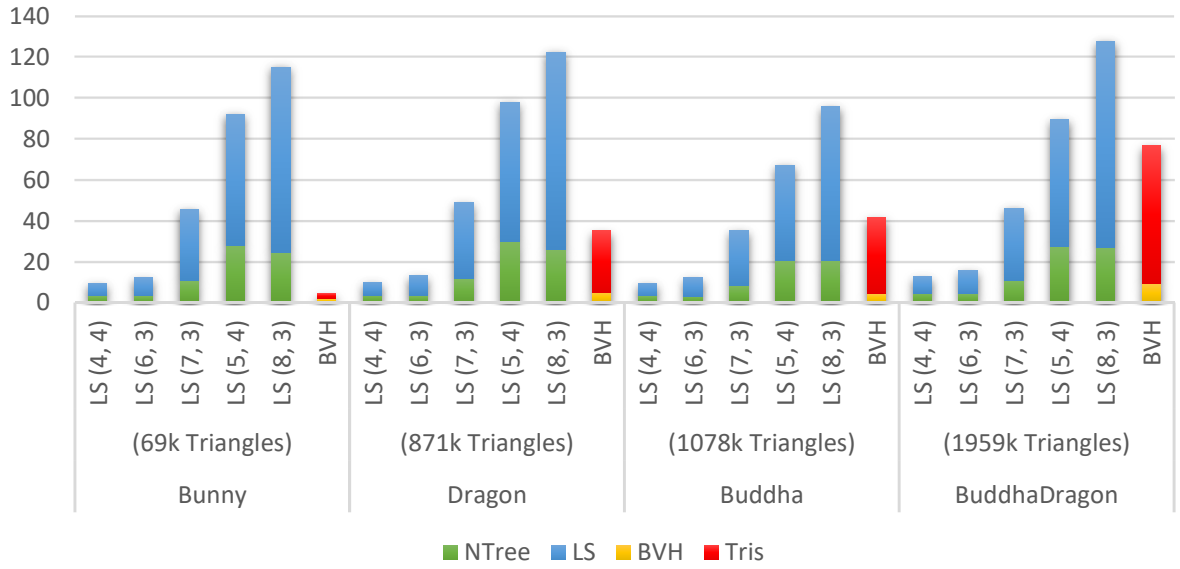


Figure 8: Overview of the memory usage (in MB) of the line space using different parameter sets and BVH for the test scenes. Although it is not necessary to store triangle information within the line space data structure, it is important in terms of memory usage to choose the correct parameter set. Using a parameter set with a big value of N or d leads to a high memory consumption, as shown by the parameter sets of LS (5, 4) and LS (8, 3). It is visible that the size of the used memory is nearly scene independent and mainly depends on the used parameters. The size of the BVH is rather small but needs the triangle information of the scene in addition.

The rendering results for soft shadow computation are shown in figure 9 on the last page. The usage of the line space leads to faster results in bigger scenes with only a minimal loss in accuracy. Nevertheless, the shadows are only approximated as explained above and so the quality of the line space results is not as precise compared to accurate computations using BVH accelerated ray tracing. The results show the mentioned difficulties of our approach. It is observable that different parameter sets of the line space lead to different results in the shadow generation, which can be explained with the varying orientation and size of the shafts within nodes with different resolutions. Furthermore it is visible, that shadows in detailed areas get lost with the line space and the shadows are in total slightly darker. But overall, these inaccuracies are only barely noticeable. The results are nearly similar to the precise results of the BVH traced method and are therefore quite acceptable.

6 CONCLUSION AND FUTURE WORK

We present a novel approach in calculating fast shadows with the GPU. Using the line space as a data structure for precomputed approximated occlusion values leads to a fast traversal of shadow rays. Though the produced shadows are not absolutely accurate, they are precise enough for soft shadows. Moreover, we showed that the results are faster in production than typical ray tracing methods. Furthermore, the data structure does not need information about the scene geometry for shadow calculation and it is therefore able to have a smaller memory consumption.

As future work we want to accelerate the initialization process by computing it in parallel on the GPU. With that it may be possible to work with dynamic scenes and the line space may then become an alternative for typical soft shadow techniques for dynamic scenes.

Apart from this we want to investigate the impact of storing not only binary information for shafts. For example it may be possible to precompute ambient occlusion values per shaft and store them within the data structure. This would accelerate the traversal step further and may therefore lead to better results. Another option would be to not store binary information in a shaft, whether it is intersected by scene geometry, but to count the number of objects intersecting the shaft. This may give the possibility for dynamic updates, so that it is not necessary to recompute the full line space once an object is moving.

Furthermore we want to identify the extents of the usefulness of the line space. In previous work it was shown that it is possible to speed up the general traversal in ray tracing. Additionally we showed that it is possible to use the line space visibility information for shadow computations. In the future we want to examine if other information for shafts may also grant the possibility to compute indirect or global illumination.

7 REFERENCES

[Áfr14] Áfra, A. T. and Szirmay-Kalos, L. Stackless multi-bvh traversal for cpu, mic and gpu ray tracing. In *Computer Graphics Forum*, volume 33, pp. 129–140. Wiley Online Library, 2014.

[Ail09] Aila, T. and Laine, S. Understanding the efficiency of ray traversal on gpus. In *Proceedings of the conference on high performance graphics 2009*, pp. 145–149. ACM, 2009.

[Ass03] Assarsson, U. and Akenine-Möller, T. A geometry-based soft shadow volume algorithm using graphics hardware. In *ACM Transactions on Graphics (TOG)*, volume 22, pp. 511–520. ACM, 2003.

[Bit01] Bittner, J., Wonka, P., and Wimmer, M. Visibility preprocessing for urban scenes using line space subdivision. In *Computer Graphics and Applications, 2001. Proceedings. Ninth Pacific Conference on*, pp. 276–284. IEEE, 2001.

[Cra11] Crassin, C., Neyret, F., Sainz, M., Green, S., and Eisemann, E. Interactive indirect illumination using voxel cone tracing. In *Computer Graphics Forum*, volume 30, pp. 1921–1930. Wiley Online Library, 2011.

[Cro77] Crow, F. C. Shadow algorithms for computer graphics. In *Acm siggraph computer graphics*, volume 11, pp. 242–248. ACM, 1977.

[Dre97] Drettakis, G. and Sillion, F. X. Interactive update of global illumination using a line-space hierarchy. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 57–64. ACM Press/Addison-Wesley Publishing Co., 1997.

[Eis11] Eisemann, E., Schwarz, M., Assarsson, U., and Wimmer, M. *Real-time shadows*. CRC Press, 2011.

[Ern08] Ernst, M. and Greiner, G. Multi bounding volume hierarchies. In *Interactive Ray Tracing, 2008. RT 2008. IEEE Symposium on*, pp. 35–40. IEEE, 2008.

[Fer05] Fernando, R. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches*, p. 35. ACM, 2005.

[Gob05] Gobbetti, E. and Marton, F. Far voxels: a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. *ACM Transactions on Graphics (TOG)*, 24(3):pp. 878–885, 2005.

[Has03] Hasenfratz, J.-M., Lapierre, M., Holzschuch, N., and Sillion, F. A survey of real-time soft shadows algorithms. In *Computer Graphics Forum*, volume 22, pp. 753–774. Wiley Online Library, 2003.

[Jev88] Jevans, D. and Wyvill, B. Adaptive voxel subdivision for ray tracing. 1988.

[Käm13] Kämpe, V., Sintorn, E., and Assarsson, U. High resolution sparse voxel dags. *ACM Transactions on Graphics (TOG)*, 32(4):p. 101, 2013.

[Kar12] Karras, T. Maximizing parallelism in the construction of bvhs, octrees, and k-d trees. In *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*, pp. 33–37. Eurographics Association, 2012.

[Kar13] Karras, T. and Aila, T. Fast parallel construction of high-quality bounding volume hierarchies. In *Proceedings of the 5th High-Performance Graphics Conference*, pp. 89–99. ACM, 2013.

[Keu16] Keul, K., Lemke, P., and Müller, S. Accelerating spatial data structures in ray tracing through precomputed line space visibility. In *Proceedings of the 24th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pp. 17–26. WSCG, 2016.

[Lai05] Laine, S., Aila, T., Assarsson, U., Lehtinen, J., and Akenine-Möller, T. Soft shadow volumes for ray tracing. *ACM Transactions on Graphics (TOG)*, 24(3):pp. 1156–1165, 2005.

[Lai11] Laine, S. and Karras, T. Efficient sparse voxel octrees. *IEEE Transactions on Visualization and Computer Graphics*, 17(8):pp. 1048–1059, 2011.

[Lau09] Lauterbach, C., Garland, M., Sengupta, S., Luebke, D., and Manocha, D. Fast bvh construction on gpus. In *Computer Graphics Forum*, volume 28, pp. 375–384. Wiley Online Library, 2009.

[Ley03] Leyvand, T., Sorkine, O., and Cohen-Or, D. *Ray space factorization for from-region visibility*, volume 22. ACM, 2003.

[Ree87] Reeves, W. T., Salesin, D. H., and Cook, R. L. Rendering antialiased shadows with depth maps. In *ACM Siggraph Computer Graphics*, volume 21, pp. 283–291. ACM, 1987.

[Sin14] Sintorn, E., Kämpe, V., Olsson, O., and Assarsson, U. Compact precomputed voxelized shadows. *ACM Transactions on Graphics (TOG)*, 33(4):p. 150, 2014.

[Sti09] Stich, M., Friedrich, H., and Dietrich, A. Spatial splits in bounding volume hierarchies. In *Proceedings of the Conference on High Performance Graphics 2009*, pp. 7–13. ACM, 2009.

[Wal07] Wald, I. On fast construction of sah-based bounding volume hierarchies. In *2007 IEEE Symposium on Interactive Ray Tracing*, pp. 33–40. IEEE, 2007.

[Wal08] Wald, I., Benthin, C., and Boulos, S. Getting rid of packets-efficient simd single-ray traversal using multi-branching bvhs. In *Interactive Ray Tracing, 2008. RT 2008. IEEE Symposium on*, pp. 49–57. IEEE, 2008.

[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):pp. 47–57, 2012.

[Wil78] Williams, L. Casting curved shadows on curved surfaces. In *ACM Siggraph Computer Graphics*, volume 12, pp. 270–274. ACM, 1978.

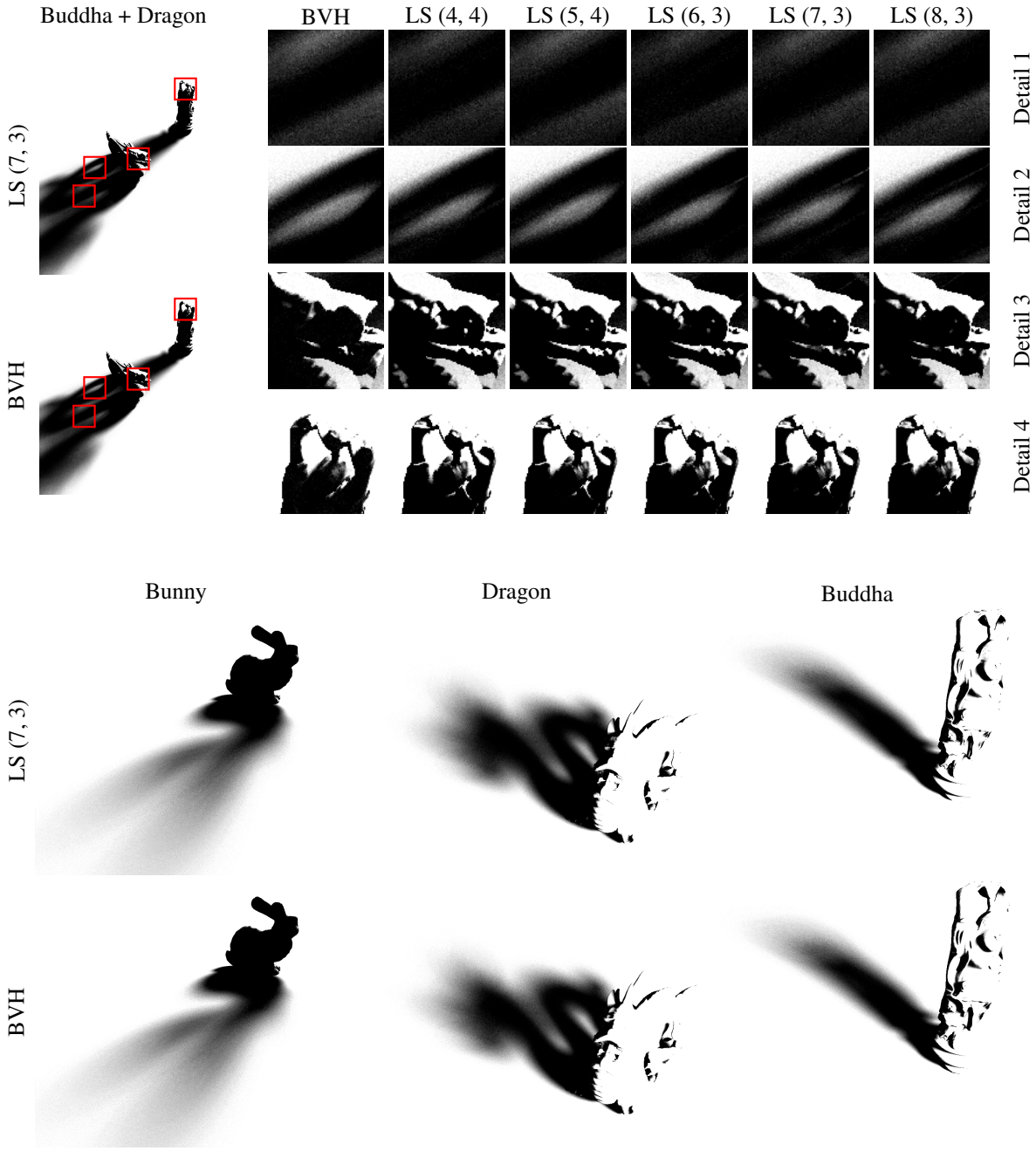


Figure 9: The test results with different test scenes. The geometry is rendered with typical forward rendering. Soft shadows are then computed with our technique using a medium sized parameter set of $(N,d) = (7,3)$. The results are compared to a BVH accelerated ray tracer based on the work of Aila and Laine [Ail09]. All images have resolutions of 1024×1024 and soft shadows were generated using one area light source and 25 samples. The size of the test scenes range from small (Bunny, 69k triangles) to rather big (Buddha + Dragon, 1959k triangles). Additionally two medium sized test scenes (Dragon, 871k triangles and Buddha, 1087k triangles) were used for evaluation. In the upper part the results for the big scene are shown. For the line space magnified images with varying parameter sets are illustrated to demonstrate the differences caused by the early ray termination of the line space. Using parameter sets with bigger parameter values lead to better results but have a much higher memory consumption as demonstrated in table 1. In the lower part a comparison to a BVH based ray tracer as ground truth is presented. The inaccuracies of the line space mentioned above are only barely noticeable, even with smaller parameter sets. The produced shadows are slightly darker and the skipping of the first node in the line space traversal leads to less occlusion in detailed areas. Overall, our results are similar to ground truth renderings, but have better performance and do not need to store the scene geometry, which grants less memory usage and different opportunities in the future.