ISSN 1213-6972 Journal of WSCG http://www.wscg.eu/ Vol.26, 2018, No.1

Fast Indirect Lighting Approximations using the Representative Candidate Line Space

Kevin Keul Tilman Koß Stefan Müller
Department of Computer Graphics,
Institute for Computational Visualistics,
University of Koblenz-Landau,
Koblenz, Germany
{keul | tkoss | stefanm}@uni-koblenz.de

ABSTRACT

We propose a novel approach for using directional Line Space information in calculation of indirect illumination. Typically, the Line Space is build on top of regular recursive grids and contains visibility information which is used to perform an efficient empty space skipping during traversal. In our method we extend the stored information by precomputed representative candidates, which are based on the Line Space shafts and serve as an approximation of the actual scene geometry. By using these candidates it is not necessary to compute any intersection tests and therefore the traversal is accelerated. However, the candidate approximation leads to visible artifacts. We therefore present a technique that significantly reduces these artifacts by extrapolation of the actual surface and demonstrate that the artifacts are nearly not perceivable in the application of indirect illumination. Moreover we adapt the Line Space to other data structures like bounding volume hierarchies (BVHs) which further increases the performance in ray tracing. Compared to the pure data structures we achieve significantly better performance with nearly no drawback in quality of indirect lighting.

Keywords

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

1 INTRODUCTION

Calculation of global illumination and indirect lighting is a non-trivial task which significantly improves realism of generated images and renders the possibility for photo realistic effects. The two main ways for computation of global illumination are depth-based rasterization techniques and ray tracing. The former is typically used in interactive and real-time rendering, due to the high performance that is achieved. The basic idea is to determine the visible scene primitives through projection to the screen in object order. Adding complex rendering effects like global illumination without ray tracing is a non-trivial task and suffers from different quality problems [Rit12]. In ray tracing the visible surfaces are calculated per screen pixel by computing intersections between rays and the scene primitives. By using additional rays per pixel it is possible to calculate complex rendering effects and indirect lighting. However, be-

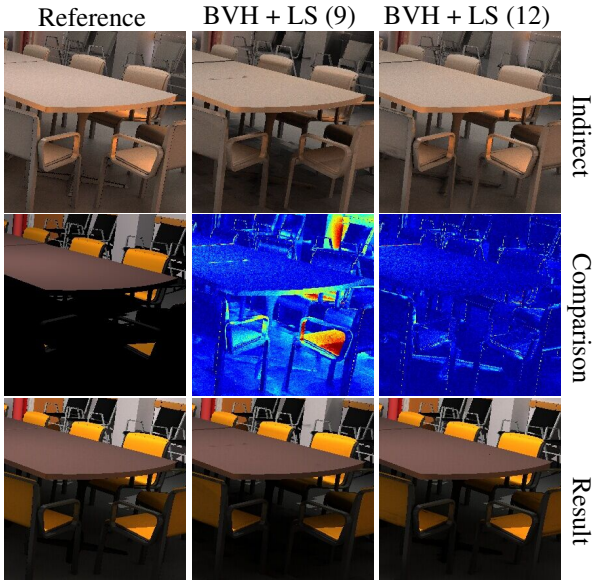


Figure 1: Example of our technique. The left column shows correct results as reference, the other columns show the utilization of precomputed scene primitives in the Line Space using a low and a high depth parameter. In the top row indirect illumination is presented. The middle row shows a comparison to ground truth, where the left image presents only direct illumination and shadows. The last row consists of the final images.

https://doi.org/10.24132/JWSCG.2018.26.1.2

cause of the huge number of intersection calculations, this process is quite slow and therefore a well defined acceleration data structure is needed.

Most data structures used for this purpose work in a spatial manner by grouping scene primitives within bounding volumes and thereby limiting the needed intersection tests to a minimum. During ray traversal the bounding volumes are tested for intersection first and only those scene primitives that are contained by intersected bounding volumes are tested for intersection with the ray. Most of the primitives are excluded within a short time. On a basic level spatial data structures are distinguished by the size and arrangement of these volumes [Hav00]. A common similarity of most of the used data structures is that axis aligned bounding boxes are used as bounding volumes because of their simplicity. Still, a lot of intersection tests need to be calculated.

A further approach is to precompute visibility in a data structure and therefore eliminating the need of intersection tests. More recently this technique received renewed interest and was used to accelerate the traversal of shadow rays and intersection finding. Recursive grids were extended by directional information of the Line Space, which uses ray clustering into predefined shafts. Binary visibility information based on the shafts was computed and during runtime applied to the contained rays. This allowed a direct access of visibility information instead of complex intersection tests.

While in previous work only binary visibility information was used, we further extend the Line Space by precomputing a representative candidate (i.e. a triangle) for each non-empty shaft. This leads to significantly faster but approximated results, which can be used for the acceleration of indirect lighting computations. While the results suffer from approximation artifacts, it was shown in [Yu09] that indirect illumination does not require correct results and therefore the artifacts can be disregarded in this context, as shown in Figure 1. Moreover we use a general Line Space description on basis of bounding boxes. With this our approach can be applied to almost every spatial data structure used as base structure. We demonstrate this applicability with the NTree, a regular recursive grid structure, as used in previous work, and BVHs and therefore show the general utility in terms of accelerating performance. The main contributions of our paper are:

- An approach for precomputation of possible intersection candidates based on the simplification of clustering rays into shafts in the Line Space with the application of indirect lighting calculation without the need of intersection tests.
- A generalization of the Line Space to bounding boxes and therefore the adaption to almost all commonly used spatial data structures.
- An evaluation in terms of performance, memory consumption, initialization speed and quality of the base data structure in combination with the Line Space and the comparison to the pure data structure.

2 RELATED WORK

Rendering of indirect lighting and global illumination is a well studied and complex topic. Therefore we refer to [Wal03] [Pha16] and [Rit12] for a broad overview of techniques and possibilities in ray tracing and rasterization based techniques. We focus specifically on the acceleration of approximated intersection point computation through the usage of sophisticated data structures.

Spatial data structures.

Nowadays most acceleration data structures for ray tracing work with a hierarchical spatial subdivision of the scene space [Hav00]. All geometrical objects of the scene are arranged depending on their spatial localization and clustered in separate bounding volumes of the data structure. During rendering the rays are traversed along those bounding volumes and only the scene objects within passed volumes are used for intersection tests. In this kind recursive grids [Jev89] are the combination of a grid like subdivision of volumes in a recursive hierarchical structure. It was shown, that ray traversal greatly benefits from those data structures. The bounding volume hierarchy (BVH) works by recursively grouping of adjacent scene primitives within axis aligned bounding boxes [Ail13]. It is currently the most used data structure, mainly because of the good performance even in dynamic scenes, the small memory footprint and fast build time in comparison to other data structures [Vin16]. It is important for BVHs to construct a high quality tree and many different initialization algorithms were proposed for this purpose. The surface area heuristic (SAH) used in combination with spatial splits is an example for a good tree construction [Sti09]. Moreover there exist build algorithms that work on already constructed BVHs, which are then optimized to gain better quality [Kar13]. The bonsai algorithm uses a two-level construction with optimization in so called mini trees resulting in high performance and good build times [Gan15]. By using agglomerative clustering and multi-threaded CPU approximations a good trade-off between quality and construction time can be found [Gu13]. By optimizing spatial splitting during construction this trade-off is further improved [Wod17] [Mei17]. While all previously mentioned approaches result in good tree quality, their construction takes quite a long time and dynamic scenes are not covered. Using linear sorting through morton codes leads to the linear BVH (LBVH) with fast build times due to parallelization on the GPU, however with inferior tree quality [Lau09]. An advancement to this is

the hierarchical LBVH (HLBVH), which is especially used in full dynamic scenes because of the fast creation [Pan10] [Gar11]. Further optimization of fast BVH construction can be made by refitting of splitting planes in dynamic scenes [Yin14]. Usually the traversal of a BVH works in a stack-based manner [Ail12]. More recently combinations of multiple data structures are explored [Wan16].

Directional data structures.

Using directional information instead of or in addition to spatial information provides the possibility for visibility precomputation. Most attempts aim to generalize rays on a higher level and then precompute information on an intermediate representation. Examples for this are the generalization of rays to cones [Ama84], beams [Hec84] [Res05] [Lai09] or more generally to higher dimensional generalizations [Arv87]. In the latter, rays are classified by their three dimensional origin and their two dimensional direction and for each resulting five dimensional generalization a sorted list of intersecting objects is stored. This was optimized by reducing the generalization to four dimensions [Kwo98]. The four dimensional visibility field could be projected onto a bounding sphere which was then used to speed up ambient occlusion and stochastic ray tracing calculations [Mor07] [Gai10]. In a similar manner visibility information can be projected on planes, leading to intersection fields which were used for fast computation of global illumination [Ren05]. The concept of generalizing rays to shafts was introduced, where each shaft is the volume that is constructed by connecting two patches and forming their convex hull [Hai94] [Dre97]. There, a candidate list per shaft is created and later on used for all rays that pass a given shaft, which was applied to ray tracing and radiosity calculations. Recently, this approach was combined with a spatial recursive grid structure in terms of empty space skipping [Keu16] and shadow calculation [Bil16] [Keu17]. In this context, shafts have binary visibility information and are created between all patches of the regular subdivided boundary of each branching node in the tree hierarchy resulting in the Line Space.

In our approach the Line Space can be adapted to all spatial data structures that use bounding boxes of any kind. In addition to the binary visibility information of previous approaches, we store actual geometry information in the Line Space.

3 LINE SPACE WITH REPRESENTATIVE CANDIDATE DATA

The Line Space, as proposed by previous work [Keu16], is a data structure providing directional information for a given bounding box. The six faces of the box are equally divided into N^2 rectangular patches. Pairs of those patches that are arranged on different box faces are defined as *shafts*. The volume of a shaft is the convex set of all line segments connecting any point in the start patch with any point in the end patch. The Line Space stores arbitrary data for each shaft. A Line Space where a substitution of the start and end patches leads to the same result as the original one is called *symmetric*. There are $30N^4$ shafts in a non-symmetric Line Space and $15N^4$ shafts in a symmetric Line Space, therefore potentially resulting in a big memory consumption, as shown by previous work.

3.1 Representative Candidate per shaft

Until now, the Line Space was only used to store binary visibility information, i.e. whether a given shaft contains any geometry at all. This approach was utilized for empty space skipping [Keu16] and accelerated shadow computation [Bil16] [Keu17]. We extend it by storing a representative triangle for each shaft, which serves as an approximation for the geometry inside of the shaft. The stored information can be used during the traversal step of ray tracing to get a possible intersection between a given ray and the scene geometry. Instead of testing all candidate triangles of the given bounding box for intersections, only the previously stored representative triangle is considered. The intersection between the ray and the bounding box geometry is approximated as shown in algorithm 1.

Algorithm 1 The accelerated intersection algorithm between a ray and a Line Space bounding box.

```
(t_start, t_end) ← points where ray intersects box
i ← CALCPATCH(t_start)                    ▷ start patch
j ← CALCPATCH(t_end)                      ▷ end patch
shaftID ← CALCSHAFT(i, j)
triangle ← GETCANDIDATETRIANGLE(shaftID)
if triangle exists then
    return ray triangle intersection
return 0
```

The representative candidate Line Space is non-symmetric. The candidate used as the shaft representative is the triangle that optimally approximates the object surface within the shaft. To find it we search for an intersection between the geometry and the ray defined by the centroids of the shaft's start and end patches. This is illustrated in Figure 2. If no intersection is found, the shaft is marked as empty. The percentage of empty Line Space shafts is typically between $30\% - 70\%$ for manifold meshes and therefore a lot of memory can be saved with an appropriate memory layout, which is explained later on.

The surface inside a shaft can be classified into three categories:

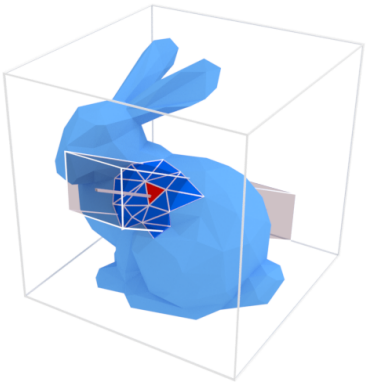


Figure 2: The representative candidate triangle (shown in red) is the triangle that is used to approximate the geometry in one shaft. This triangle is found by computing an intersection between the geometry and the centroid ray of the shaft.

1. The surface is closed and covers the whole shaft width.

2. The shaft lies at the boundary of a surface or contains multiple disconnected surfaces.

3. The shaft is empty.

Since we only store the single triangle per shaft, it is not inherently possible to distinguish the first two cases. The candidate triangle does not necessarily cover the whole shaft surface, as it is shown in Figure 2. To compensate this, the triangle is treated as infinite plane defined by its vertices. This approximation is used when searching for an intersection between a ray and the geometry contained in the shaft. Per-vertex normals can be interpolated by using the intersection parameters of the constructed plane. If the intersection point is outside of the triangle, it is computed by the extrapolation, therefore providing smooth normals for the whole shaft width. This is a rather big approximation, especially for highly curved surfaces, however it lowers discontinuity artifacts. To reduce artifacts for shafts that are not fully covered by a surface, edges can be found by calculating the angle between the extrapolated normal and the mean normal of the triangle. If the angle is bigger than a given threshold then the intersection is discarded.

The representative candidate Line Space stores a reference to a triangle for each shaft. In our case, this reference is a 32-bit index pointing to a buffer containing all triangles of the scene geometry. Depending on the storage layout of the scene geometry, more space efficient data types are possible. Since the Line Space stores data for every combination of start and end patch, it contains $M = 30N^4$ elements. While most of these shafts are empty and do not point to a valid triangle, we are able to only store the shaft information of filled shafts. This

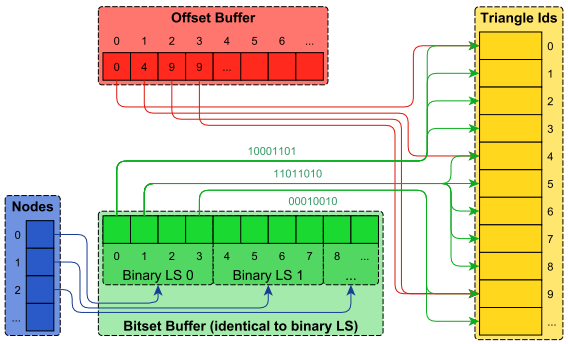


Figure 3: The sparse memory layout of our structure. Nodes have references to their bitsets (shown in green), which signal whether associated shafts are filled or empty. To access the triangle data, an additional offset per bitset is needed (shown in red).

is done by using a simple sparse scheme based on a bitset and offset buffer to skip empty shaft entries and only store filled shafts consecutively in memory. In addition we need to store one bit of information for each shaft, signaling whether the shaft is empty or filled. These bits are grouped in bitsets and are efficiently represented by 32-bit words. Moreover an offset for every bitset is stored, which specifies where the corresponding filled shaft entries withing the shaft buffer are located. This is shown in Figure 3. The memory overhead of this scheme therefore sums up to $\frac{M}{16}$ 32-bit words. Finally, the sparse storage is more memory efficient compared to dense storage if less than $\frac{15}{16} \approx 93\%$ of the shafts are filled, which is always the case. The offset computation is done by a parallel prefix sum, efficiently calculated on the GPU. It should be noted that the bitsets are identical with the binary Line Space, and therefore they are implicitly calculated. The data access with the sparse memory layout has a constant time complexity and is presented in algorithm 2. We use 32-bit words for all bitset and offset operations due to the better interaction with GPU computations. However, this can be generalized for arbitrary sizes like 64-bit words.

Algorithm 2 The algorithm presents the access of the shaft data in the sparse memory scheme.

```
procedure GETSPARSEDATA(ShaftIndex n)
    bitsetID ← ⌊n/32⌋
    bitset ← GETBITSET(bitsetID)
    bit ← n mod 32
    if (bitset & (1 ≪ bit)) ≠ 0 then
        offset ← GETOFFSET(bitsetID)
        id ← BITCOUNT(bitset ≪ (31 - bit)) - 1)
        return GETDATA(offset + id)
    else
        return 0
```

3.2 General Line Space for spatial datastructures

Because of the construction on top of bounding boxes, the Line Space can be adapted and integrated into each spatial data structure that consists of bounding boxes in any way. It provides directional information in addition to the spatial subdivision and therefore is able to minimize the traversal cost. Typical data structuces that can be used for this are Octrees, BVHs, k-d trees, uniform grids or recursive grids (such as the NTree in previous work).

We generate a representative candidate Line Space for selected tree nodes of the data structure to approximate the scene geometry. While the produced errors are too striking for direct illumination of primary rays, they are less perceivable when used for indirect illumination, which is in accordance to [Yu09], stating that accurate calculations are not required for global and indirect illumination. Therefore we use the representative candidate as approximation instead of the correct triangle data to calculate the intersection points in indirect illumination. In terms of the underlying base data structure it is necessary to consider which nodes in the tree need to store the Line Space information.

Adaptation to NTree

The data structure previously used for Line Space computations is the *NTree*, which is a regular recursive grid that repeatedly subdivides the scene and the already produced subdivisions into N^3 equally sized bounding boxes. In our case, we constrain the size of theses boxes to be cubes. This simplifies some computations when building and traversing the NTree while not having any detrimental effect for the data structure. The NTree provides increased traversal performance in comparison to a regular Octree or single layer uniform grid. This is because the tree width of an NTree with $N > 2$ can be larger than for Octrees, effectively lowering the tree depth. Since the bounding boxes of NTree nodes are equally sized, the NTree is a natural fit for Line Space computations, as shown in previous work. The Line Space patch size for a specific tree depth is equal for all nodes and correlates with the size of the node subdivisions.

To use the NTree with the representative candidate Line Space for indirect lighting approximations we first generate the NTree including the exact triangle data of the scene. This NTree can be used for all exact computations like primary or shadow rays. Moreover we utilize it for faster initialization of the representative candidates. We only compute the Line Space data on specific nodes in the NTree, which are determined by the depth D or a triangle count lower than T (i.e. $T = 8$). In our case the NTree and therefore also the representative candidate Line Space have a parameter values $N = 6$ or $N = 10$ and $D = 2$ for Line Space utilization, which is consistent with the results of previous work. The N value describes the branching factor of the NTree as well as the Line Space resolution. It was found to be accurate enough for the approximation while also granting sufficient performance. The depth parameter D also determines the performance, memory requirements and the approximation accuracy of the Line Space. Higher depth values lead to more Line Space nodes and therefore higher computation times and memory consumption. However, lower depth values decrease the accuracy of scene approximations but significantly increase the traversal performance. This is due to the fact that the number of nodes greatly increases with the depth of the underlying tree. The shown value of $D = 2$ for the usage of Line Spaces within the NTree is sufficient for quality, traversal speed and memory consumption.

When traversing indirect rays, these Line Space nodes are treated as leaf nodes in the NTree and are therefore able to terminate the traversal. Intersections with the scene are calculated by the procedure explained in subsection 3.1. The general traversal algorithm of the data structure does not need to be changed, only the handling of leaf nodes needs to be replaced by the appropriate Line Space calculations as shown in algorithm 1.

Adaptation to BVH

By using a BVH, the scene is recursively subdivided by axis aligned bounding boxes that tightly enclose the geometry. Besides the NTree, the BVH is also used as a base data structure for the Line Space in our work. The representative candidate in the Line Space nodes are used in the same way as described with the NTree. Due to the reason that every BVH node branches into only 2 subnodes, the tree depth is normally much higher than for the NTree. With this BVH nodes converge to the actual scene geometry and they are not constrained to be equal in size in every tree layer. Typically less nodes are needed in the BVH for scenes with a high amount of empty space. Again, the depth D and the triangle count T are used to determine whether a node is extended by a Line Space. Furthermore, it is possible to consider the box size as criterion for this determination, but this was not done in our work.

Nevertheless, the usage of a BVH with the representative candidate Line Space has two disadvantages. Since the bounding boxes are not cubical, the shaft patch size will slightly differ for each bounding box. The approximation artifacts in that case are not distributed in any predictive manner, and therefore have significant impact, depending on the used parameter set. This is visible in the results using low parameter sets for the BVH Line Space. Additionally, BVH nodes may overlap, especially in scenes where the triangle size is highly diverse. Therefore, multiple Line Space nodes and their shafts may overlap and approximate the same scene geometry using different representative candidates. These

effects are mostly visible in architectural scenes, as shown in the results. A spatial split BVH construction might reduce these effects significantly, however this was not used in our work.

4 RESULTS

For the evaluation we used two different base data structures: the NTree, a regular recursive grid as it was used in previous work on the binary Line Space [Keu16] [Bil16], and a state of the art BVH algorithm [Ail12], which is used as comparison in multiple related works. For the NTree we used a branching factor of 6 and 10 and a hierarchical depth of 3, as those are the proposed parameters by [Keu16]. We did not incorporate any further optimizations of the BVH tree quality, as recently proposed [Gan15] [Yin14]. The Line Space with precomputed representative shaft candidates is then used in a given hierarchical depth of the base data structure. Within the NTree this depth is set to the lowest branching nodes in the hierarchy, i.e. depth 2. The Line Space depth within the BVH is more versatile and can be set arbitrarily to achieve a good trade-off between performance and memory consumption as well as initialization time. The chosen depths and their impact are shown in the diagram and the visual results.

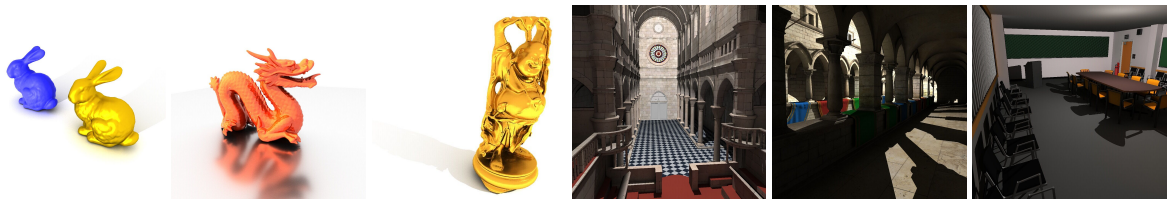
We measured the quality and the differences in performance, initialization time and memory consumption. For this purpose different widely used test scenes with special characteristics are evaluated. In principle they are dividable into two categories: scenes containing a single object (BUNNY, DRAGON and BUDDHA) and architectural scenes (SIBENIK, SPONZA and CONFERENCE), which are more suitable for usage in video games. Apart from this, the number of scene primitives varies significantly in the used scenes and ranges from ~70k triangles up to ~1 million triangles. The test results were produced on a GeForce GTX 1080, however the relative performance is the same on similar systems. All data structures are implemented in the same environment and supported by acceleration in agreement. Hence, a fair comparison of the used structures is guaranteed. The resolution of the renderings was in all cases 720p. Primary and shadow rays were rendered with fast rasterization accelerated by a binary Line Space techniques as proposed by [Bil16] and [Keu17] and are therefore out of our scope. Regarding this, our approach accelerates calculations of all indirect lighting effects, resulting for example in ambient occlusion, diffuse illumination and glossy reflections.

Table 1 shows the quantitative results using the two main data structures with and without the acceleration of the Line Space with the mentioned depth parameter. We evaluated the build time, the memory consumption and the performance in ray tracing for indirect rays. Obviously, the computation time and memory consumption of the Line Space need to be summed up on the values of the base data structure. The illustrated Line Space values in the table are already combined and therefore show the total sum. Moreover the build times and the memory consumption of the Line Space significantly scale with the total number of computed Line Spaces and not the number of scene triangles. For BVH initialization we use a binned SAH construction, resulting in good quality but non-interactive build times. The used build algorithm significantly affects the build time of the BVH, but as our work focuses on the relative comparison of the data structure with the usage of the Line Space, this does not affect our results.

The runtime performance was measured in frames per second only counting indirect illumination. Apart from absolute values, the relative differences between pure and Line Space supported data structures show the benefit of our approach. The BVH performance is near state-of-the-art in ray tracing performance. It is mainly regulated by the quality of the underlying tree and can be further optimized by recent techniques as proposed by [Gan15] and [Yin14]. Using our technique gives a significantly better performance, however with approximated results. This is due to the simplification of shaft data, where only a single candidate is stored and used for all rays passing a given shaft. Moreover it is noticeable that the usage of the Line Space accelerates the data structure to an acceptable level, even in those cases where the base data structure performs very poorly.

Figure 5 shows the qualitative differences of various depths used in Line Space accelerated BVH in comparison to ground truth data. It is observable that lower parameter sets result in visible artifacts due to shaft simplification. However, by using higher depths for the Line Space within the BVH hierarchy the artifacts become manageable. The artifacts are especially noticeable in bigger scenes when the camera is positioned close to an object. This is mostly observable in the architectural scenes as shown in figure 6 on the last page. There, also the NTree results with the Line Space are shown. As with the BVH, the quality of the Line Space accelerated NTree improves when the Line Space is used in a deeper level of the tree hierarchy. Because of its regular structure, the NTree Line Space is mostly better suited for big architectural scenes and produces less approximation artifacts for these cases in comparison to the BVH Line Space. However, as it was shown by [Yu09], correctness in indirect illumination is not required and approximations are sufficient in most cases. Following this the BVH Line Space has better performance with mostly sufficient quality.

The main factor for quantitative and qualitative analysis is the value of the used depth parameter where Line Spaces are created and used. A deeper depth results in more Line Spaces, therefore causing higher build time



BUNNY DRAGON BUDDHA SIBENIK SPONZA CONFERENCE

Figure 4: The evaluated test scenes. Renderings were done in 720p. Primary rays were calculated with rasterization, shadow rays were rendered with a binary Line Space and indirect rays were produced with our technique.

Scene		BVH					NT (6, 3)			NT (10, 3)		
		pure	LS (9)		LS (12)		pure	LS (2)		pure	LS (2)	
Bunny	**init**	0,3	2,2		10,4		3,7	9,6		17,2	44,3	
69k tris	**size**	5,5	38,7		227,2		1,5	17,6		23,9	149,7	
	perf	76,7	194,7	**2,5x**	131,0	**1,7x**	20,6	89,4	**4,3x**	36,7	63,0	**1,7x**
Dragon	**init**	1,7	5,6		17,1		22,5	50,3		100,6	209,7	
871k tris	**size**	35,5	74,8		280,0		4,1	10,8		13,2	65,5	
	perf	45,9	169,4	**3,7x**	110,2	**2,4x**	1,4	107,9	**77x**	11,7	62,1	**5,3x**
Buddha	**init**	2,0	6,2		17,6		27,6	61,6		123,2	254,2	
1087k tris	**size**	40,7	80,0		285,0		4,5	10,9		11,6	57,5	
	perf	62,4	195,4	**3,1x**	128,0	**2,1x**	1,1	121,4	**108x**	12,1	69,4	**5,7x**
Sibenik	**init**	0,1	1,9		7,4		2,3	18,6		13,0	102,3	
75k tris	**size**	3,0	57,7		251,6		8,3	286,4		106,9	2114,3	
	perf	19,3	38,3	**2x**	28,4	**1,5x**	8,8	23,1	**2,6x**	8,6	12,8	**1,5x**
Sponza	**init**	0,5	3,0		9,2		7,1	28,5		36,3	153,0	
262k tris	**size**	10,1	54,1		231,2		8,9	347,6		133,3	2680,4	
	perf	16,7	48,5	**2,9x**	33,2	**2x**	5,8	27,4	**4,7x**	10,1	16,2	**1,6x**
Conference	**init**	0,7	3,2		9,6		7,3	27,0		36,8	115,2	
331k tris	**size**	13,4	61,3		213,2		6,3	170,9		86,3	1014,9	
	perf	16,0	44,6	**2,8x**	33,6	**2,1x**	1,4	26,3	**19x**	8,4	20,7	**2,5x**

Table 1: Test results of our evaluation. We measured the initialization time in seconds, the memory consumption in MB and the ray tracing performance in FPS for BVH and NTree as base data structures without and with the usage of the Line Space with varying depth parameter. Line Space values are already combined with the base structure. BVH initialization was optimized in terms of ray tracing performance and not initialization speed. The relative differences in comparison to the base data structure without Line Space acceleration are marked.

and memory consumption, as well as lower ray tracing performance. This is due to the fact, that with a deeper Line Space depth more nodes in the hierarchy need to be traversed. However the quality gets better when more Line Spaces are used. With this the Line Space depth can be used as an arbitrary parameter for setting a trade-off between quality and performance. This is especially true, when the base data structure produces a deep tree hierarchy, as it is done with BVHs. The NTree naturally only has a shallow tree hierarchy, therefore is not that suitable for a dynamic trade-off.

5 CONCLUSION AND FUTURE WORK

We presented the non-binary Line Space with visibility precomputation of scene information, which stores a single candidate as a representative per shaft. With this work, we explored the general approach of precomputing directional information per Line Space shaft in application of indirect and global illumination. Through the representative candidate precomputation, the need for intersection tests during traversal could be eliminated as far as possible. Although this technique results in approximation artifacts if the depth parameter is not high enough, we were able to show that these errors are nearly non-perceivable in the context of indirect illumination. When compared to the base data structure, this technique results in higher memory size and build time but is in all cases able to significantly surpass the base structure in terms of performance.

Moreover, we showed a generalization of the Line Space to all spatial data structures based on bounding boxes. We demonstrated this with an adaptation to a state-of-the-art BVH, resulting in higher performance in comparison to the NTree, the typically used base data structure of previous work.

Future Work

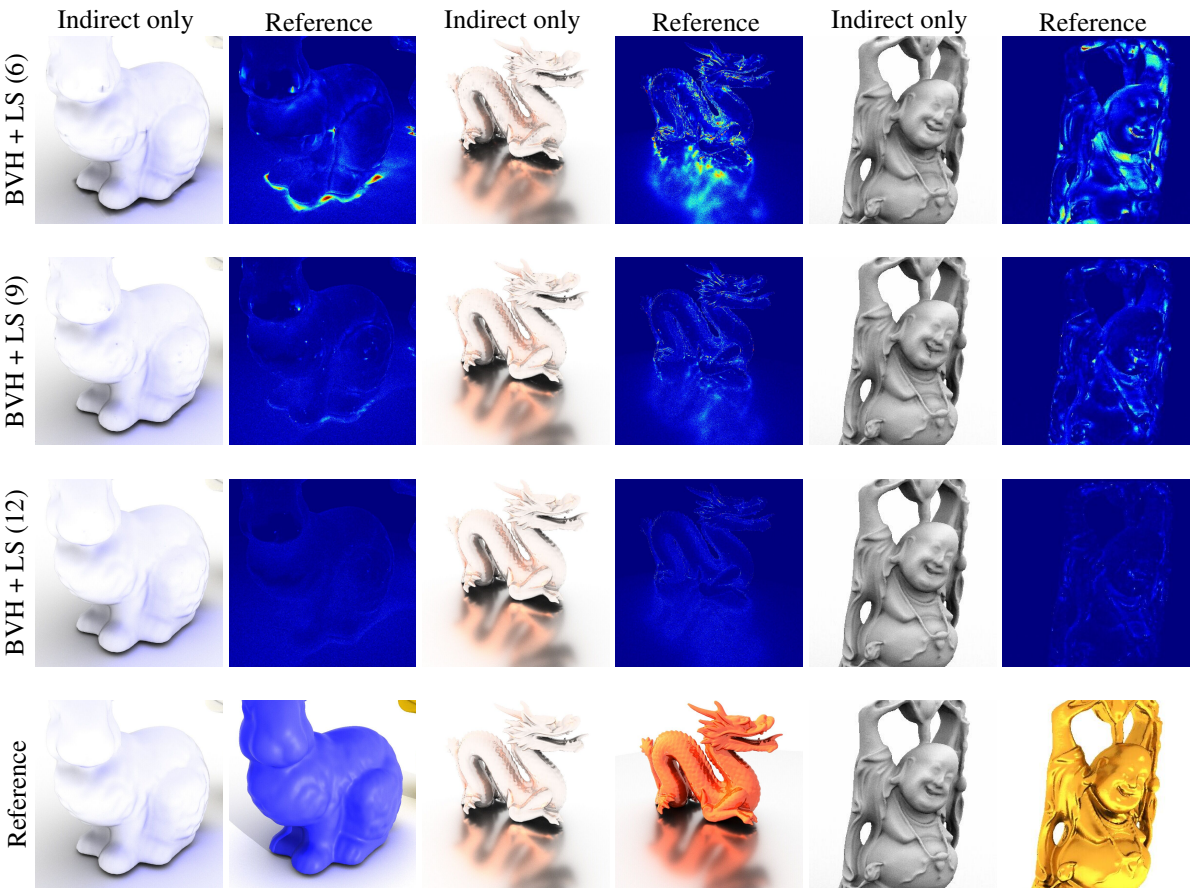


Figure 5: Some of the results of our evaluation. As illustrated, the buddha scene especially focuses on ambient occlusion, the bunny scene on diffuse materials and the dragon scene on glossy reflections. Nevertheless, all visual effects were indifferently produced with the same technique with the only difference in the object material. Therefore the results are not optimized to show specific effects. The maximum Line Space depth d within the BVH is shown in $LS(d)$.

The precomputation of scene information is only a beginning. By calculating and storing the lighting state in a shaft it may be possible to use a great variety of rendering techniques without further computation overhead during traversal. Although this leads to a significantly increase in memory consumption, the gain in tracing performance can be a big improvement in path tracing scenarios. Using a dynamic combination with the base data structure has the potential to accelerate most ray tracing systems, using the base structure in situations where correct information is needed and the Line Space where fast approximated data is sufficient.

An approach to further tackle the difficulties of memory size and initialization speed is to use the Line Space based on objects rather than the whole scene. With this each geometrical object has its own single Line Space. This grants the possibility for object instancing and a significant reduction of memory needed. Furthermore, Line Space information is more accurate, solving the teapot in a stadium problem. Object Line Spaces can be created beforehand and therefore significantly reducing initialization time. By combining the instancing aspect with local transformations creates the possibility to render dynamic scenes.

Another aspect that needs further investigation is the selection of the used parameter set. Currently, a fixed depth parameter is used for the whole Line Space tree, resulting in unnecessary high subdivision rate in sparsely filled areas. A dynamic subdivision scheme based on the number of candidates and the size of the current node would benefit the traversal and the Line Space accuracy.

6 REFERENCES

[Ail12] Aila, T., Laine, S., and Karras, T. Understanding the efficiency of ray traversal on gpus–kepler and fermi addendum. *NVIDIA Technical Report*, 2012.

[Ail13] Aila, T., Karras, T., and Laine, S. On quality metrics of bounding volume hierarchies. In *Proc. 5th High-Performance Graphics Conference*. 2013.

[Ama84] Amanatides, J. Ray tracing with cones. In *ACM Siggraph Computer Graphics*. 1984.

[Arv87] Arvo, J. and Kirk, D. Fast ray tracing by ray classification. In *ACM Siggraph Computer Graphics*. 1987.

[Bil16] Billen, N. and Dutré. Visibility acceleration using efficient ray classification. *Department of Computer Science, KU Leuven*, 2016.

[Dre97] Drettakis, G. and Sillion, F. Interactive update of global illumination using a line-space hierarchy. In *Proc. ACM SIGGRAPH*. 1997.

[Gai10] Gaitatzes, A., Andreadis, A., Papaioannou, G., and Chrysanthou, Y. Fast approximate visibility on the gpu using precomputed 4d visibility fields. *WSCG*, 2010.

[Gan15] Ganestam, P., Barringer, R., Doggett, M., and Akenine-Möller, T. Bonsai: rapid bounding volume hierarchy generation using mini trees. *Journal of Computer Graphics Techniques Vol 4*, 2015.

[Gar11] Garanzha, K., Pantaleoni, J., and McAllister, D. Simpler and faster hlbvh with work queues. In *Proc. ACM Siggraph Symp. High Performance Graphics*. 2011.

[Gu13] Gu, Y., He, Y., Fatahalian, K., and Blelloch, G. Efficient bvh construction via approximate agglomerative clustering. In *Proc. 5th High-Performance Graphics Conference*. 2013.

[Hai94] Haines, E. A. and Wallace, J. R. Shaft culling for efficient ray-cast radiosity. In *Proc. Second Eurographics Workshop on Rendering*. 1994.

[Hav00] Havran, V. *Heuristic ray shooting algorithms*. Ph.D. thesis, Ph.D. Thesis, Czech Technical University in Prague, 2000.

[Hec84] Heckbert, P. S. and Hanrahan, P. Beam tracing polygonal objects. *ACM Siggraph Computer Graphics*, 1984.

[Jev89] Jevans, D. and Wyvill, B. Adaptive voxel subdivision for ray tracing. In *Proc. Graphics Interface*. 1989.

[Kar13] Karras, T. and Aila, T. Fast parallel construction of high-quality bounding volume hierarchies. In *Proc. 5th High-Performance Graphics Conference*. 2013.

[Keu16] Keul, K., Müller, S., and Lemke, P. Accelerating spatial data structures in ray tracing through precomputed line space visibility. *Computer Science Research Notes, WSCG*, 2016.

[Keu17] Keul, K., Klee, N., and Müller, S. Soft shadow computation using precomputed line space visibility information. *Journal of WSCG*, 2017.

[Kwo98] Kwon, B., Kim, D. S., Chwa, K.-Y., and Shin, S. Y. Memory-efficient ray classification for visibility operations. *IEEE Transactions on Visualization and Computer Graphics*, 1998.

[Lai09] Laine, S., Siltanen, S., Lokki, T., and Savioja, L. Accelerated beam tracing algorithm. *Applied Acoustics*, 2009.

[Lau09] Lauterbach, C., Garland, M., Sengupta, S., Luebke, D., and Manocha, D. Fast bvh construction on gpus. In *Computer Graphics Forum*. 2009.

[Mei17] Meister, D. and Bittner, J. Parallel locally-ordered clustering for bounding volume hierarchy construction. *IEEE Transactions on Visualization and Computer Graphics*, 2017.

[Mor07] Mortensen, J., Khanna, P., Yu, I., and Slater, M. A visibility field for ray tracing. In *Computer Graphics, Imaging and Visualisation, CGIV'07*. 2007.

[Pan10] Pantaleoni, J. and Luebke, D. Hlbvh: hierarchical lbvh construction for real-time ray tracing of dynamic geometry. In *Proc. High Performance Graphics*. 2010.

[Pha16] Pharr, M., Jakob, W., and Humphreys, G. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[Ren05] Ren, Z., Hua, W., Chen, L., and Bao, H. Intersection fields for interactive global illumination. *The Visual Computer*, 2005.

[Res05] Reshetov, A., Soupikov, A., and Hurley, J. Multi-level ray tracing algorithm. *ACM Transactions on Graphics (TOG)*, 2005.

[Rit12] Ritschel, T., Dachsbacher, C., Grosch, T., and Kautz, J. The state of the art in interactive global illumination. In *Computer Graphics Forum*. 2012.

[Sti09] Stich, M., Friedrich, H., and Dietrich, A. Spatial splits in bounding volume hierarchies. In *Proc. High Performance Graphics*. 2009.

[Vin16] Vinkler, M., Havran, V., and Bittner, J. Performance comparison of bounding volume hierarchies and kd-trees for gpu ray tracing. In *Computer Graphics Forum*. 2016.

[Wal03] Wald, I., Purcell, T. J., Schmittler, J., Benthin, C., and Slusallek, P. Realtime ray tracing and its use for interactive global illumination. *Eurographics State of the Art Reports*, 2003.

[Wan16] Wang, Y., Guo, P., and Duan, F. A fast ray tracing algorithm based on a hybrid structure. *Multimedia Tools and Applications*, 2016.

[Wod17] Wodniok, D. and Goesele, M. Construction of bounding volume hierarchies with sah cost approximation on temporary subtrees. *Computers & Graphics*, 2017.

[Yin14] Yin, M. and Li, S. Fast bvh construction and refit for ray tracing of dynamic scenes. *Multimedia tools and applications*, 2014.

[Yu09] Yu, I., Cox, A., Kim, M. H., Ritschel, T., Grosch, T., Dachsbacher, C., and Kautz, J. Perceptual influence of approximate visibility in indirect illumination. *ACM Transactions on Applied Perception (TAP)*, 2009.

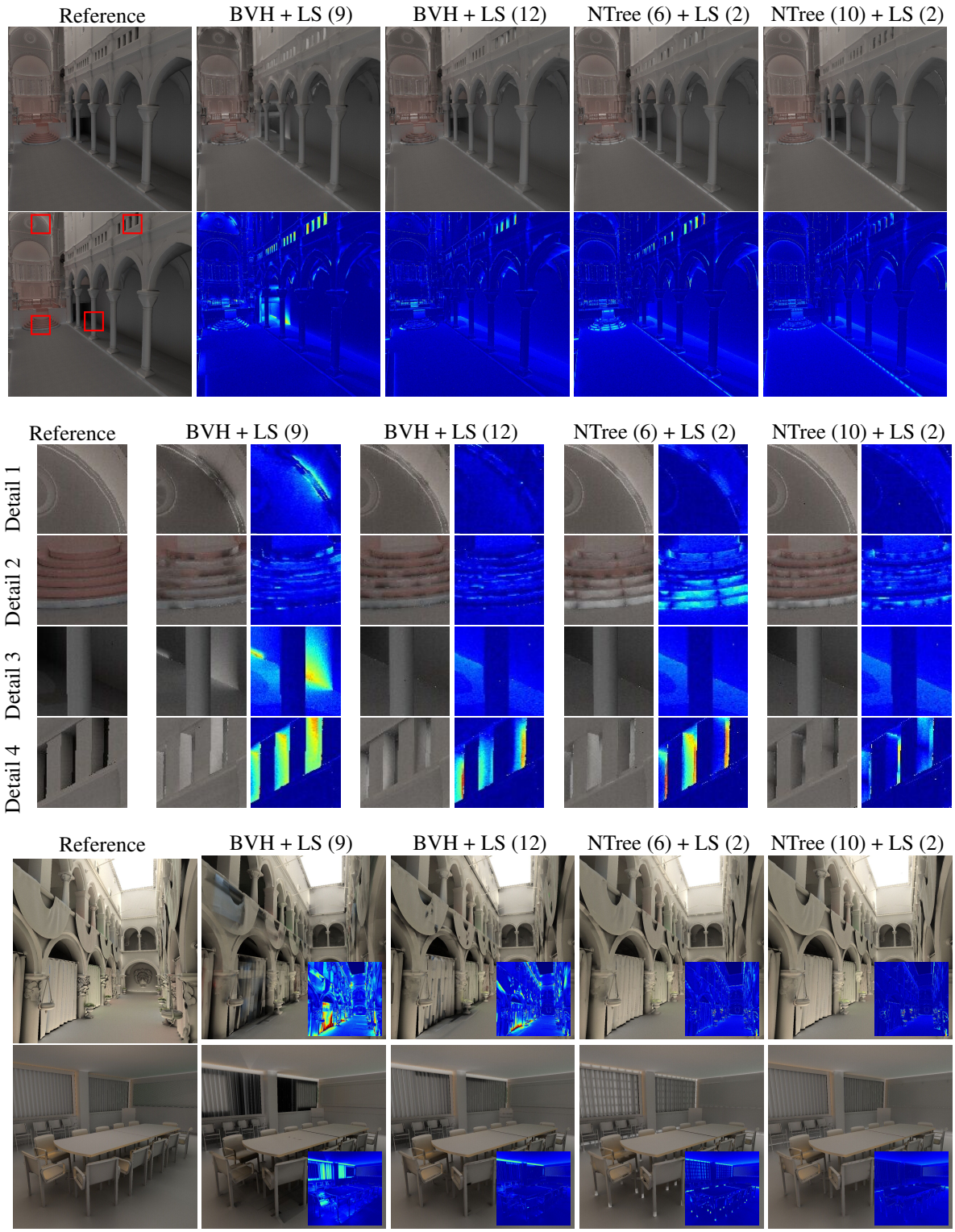


Figure 6: The test results with the architectural scenes. All images were rendered in 720p and only present indirect illumination. In bigger scenes using a lower parameter for the depth of the Line Space usage within the base data structure, more approximation artifacts due to shaft simplification occur. The detailed magnifications and the heatmaps specifically show the weaknesses of our technique using a low depth parameter. These artifacts especially occur in the transitions of different Line Spaces. However, a deeper Line Space depth improves image quality significantly, making Line Space accelerated results suitable for indirect illumination. Overall, the perception in indirect illumination given a suitable depth parameter is mostly similar to ground truth renderings, but granting significantly better performance.