CZECH TECHNICAL UNIVERSITY IN PRAGUE

Doctoral Thesis Statement

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Ing. Michal Hapala

Data Structures and Algorithms for Interactive Ray Tracing

A doctoral thesis statement submitted to
the Faculty of Electrical Engineering, Czech Technical University in Prague,
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy.

Ph.D. programme: Electrical Engineering and Information Technology
Branch of study: Information Science and Computer Engineering

Prague, June 2015

The doctoral thesis was produced in combined manner Ph.D. study at the Department of Computer Graphics and Interaction of the Faculty of Electrical Engineering of the CTU in Prague.

Candidate:

Ing. Michal Hapala
Department of Computer Graphics and Interaction
Faculty of Electrical Engineering of Czech Technical University in Prague
Karlovo nám. 13,
121 35 Prague 2, Czech Republic

Thesis Supervisor:

doc. Ing. Vlastimil Havran, Ph.D.
Department of Computer Graphics and Interaction
Faculty of Electrical Engineering of Czech Technical University in Prague
Karlovo nám. 13,
121 35 Prague 2, Czech Republic

Opponents:

. . . . . .

. . . . . .

. . . . . .

The doctoral thesis statement was distributed on: . . . . . .

The defence of the doctoral thesis will be held on . . . . . at . . . . . a.m./p.m. before the Board for the Defence of the Doctoral Thesis in the branch of study (to be specified) in the meeting room No. . . . . of the Faculty of Electrical Engineering of the CTU in Prague.

Those interested may get acquainted with the doctoral thesis concerned at the Dean Office of the Faculty of Electrical Engineering of the CTU in Prague, at the Department for Science and Research, Technická 2, Praha 6.

. . . . . .
Chairman of the Board for the Defence of the Doctoral Thesis
in the branch of study Information Science and Computer Engineering
Faculty of Electrical Engineering of the CTU in Prague
Technická 2, 166 27 Prague 6.

Contents

1 Current situation of the studied problem **1**
1.1 Data structures 1
1.2 Hardware acceleration 3

2 Aims of the doctoral thesis **5**

3 Working methods **5**

4 Results **9**

5 Conclusion **12**

6 Bibliography **13**

7 Publications of the Author **15**

8 Summary **18**

9 Résumé **19**

1 Current situation of the studied problem

Ray tracing [App68, Gla89] or ray casting is a method that finds intersections along an oriented half-line (ray) with geometric primitives in a virtual scene. This basic visibility computation is used in a core of many rendering algorithms to simulate the light distribution in a virtual environment. The naive algorithm for ray tracing with $O(N)$ complexity computes the intersections with all primitives and finds the closest ray-geometric primitive intersections, if any. This can be used efficiently only for a small number of primitives.

For larger scenes we need to restrict the number of computed intersections along the ray path. This is achieved by various spatial data structures which allow different structuring of spatial regions or objects of a scene. We have to pay the reduced number of computed intersections by the time spent on building and traversing these spatial data structures. The efficiency of build and traversal algorithms for these data structures along with their data layout is crucial for the overall performance of the rendering algorithms.

1.1 Data structures

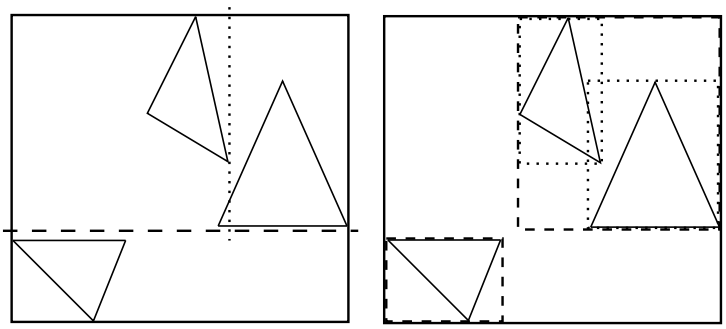
Ray tracing algorithms usually depend on spatial data structures that divide the scene into smaller parts to speed up the search for intersected objects by a particular ray. The requirements on how these data structure are created and when depends on the application and the type of the scene. It can be done as a preprocessing step or there may be need to update the data structure continually as is the case for e.g. animated scenes.

A simple example of a spatial data structure is a uniform grid [FTI86], that divides the scene non-adaptively into regular equally-sized voxels. Partitioning of the scene in this way is fast ($O(N)$ time complexity), but the search for objects to be intersected performs well only for certain types of scenes, typically with uniformly distributes primitives. The most commonly used spatial data structures are hierarchical structures which are essentially search trees.

Hierarchical spatial data structures can be divided into two major groups: space partitioning that recursively divide space, creating a hierarchy of subspaces and object partitioning that recursively divide objects, creating a hierarchy of volumes that are bounded to the objects that are inside of them (see Figure 1.1).

Typical construction of a hierarchical spatial data structure starts with a root node that encompasses all objects in the scene and follows with these recursive steps:

- Evaluate terminating criteria (e.g. reaching a certain depth in the tree or a certain minimal number of objects). If these are met, the current node is set as a leaf and the division in this part is terminated.
- Compute splitting position(s) with a chosen splitting method.
- Distribute geometry among children (not necessarily two) depending on the split.
- Repeat with all children.



Space partitioning Object partitioning

Figure 1.1: *Space and object partitioning. Dashed line signifies the first division, dotted line second division. Note that in space partitioning structures objects may overlap splitting planes and in object partitioning structures bounding volumes may overlap each other.*

The splitting method has a big influence on the quality of the spatial data structure as a whole. Two splitting approaches listed here are described in their general form, as their particular implementation in different data structures may vary. The simplest splitting method is the median i.e. one will split space/objects in the middle of the respective space range. This is definitely the fastest method but will usually create a tree with low performance.

Currently the most used approach to determine a splitting plane is the Surface area heuristic (SAH) [GS87], elaborated in [Hav00] and more recently described e.g. by Wald and Havran [WH06]. SAH computes the splitting position according to the equation:

$$C = K_T + K_I(\frac{SA(V_L)}{SA(V)}|L| + \frac{SA(V_R)}{SA(V)}|R|),$$

where $SA(X)$ is a surface area of X, V_L is the left child node, V_R is the right child node, V is the parent node, $|L|$ is the number of geometric primitives to the left of the splitting plane, $|R|$ is the number of geometric primitives to the right of the splitting plane, K_T is the cost of one traversal step and K_I is the cost of intersecting a triangle.

The cost function has to be computed for every change of the number of geometric primitives along the split axis, thus one has to compute SAH for every left and right boundary of a geometric primitive on each coordinate axis. It is a local greedy heuristic, but works well in practice. SAH also provides a terminating criterion. The splitting is terminated when the cost of intersecting triangles is lower than the cost of splitting.

Space partitioning

Space partitioning data structures divide space into disjoint subspaces usually defined by split planes. A disadvantage of this approach is that geometric primitives are usually referenced more than once, as it is quite common for them to straddle the split plane, thus belonging to more than one node.

The most common space partitioning data structure used in ray tracing is a kd-tree. It is a special case of a binary space partitioning tree, recursively partitioning space with splitting planes that are perpendicular to the axes of the coordinate system. Every interior node of this tree has a split value and a split axis, which together define a splitting plane. Half-spaces on each side

of the plane belong to the left and the right child of the original node and geometric primitives from the node are redistributed among its children. Those overlapping the split plane are copied into both of them.

Kd-trees were first used in ray tracing by Kaplan [Kap85] and in-depth elaborated by Havran [Hav00]. Wald and Havran [WH06] summarized techniques for kd-tree construction and have shown an optimal algorithm for it and Hapala and Havran [HH11] have summarized kd-tree traversal algorithms.

Object partitioning

Object partitioning data structure divides geometric primitives into disjoint subsets. The most common object partitioning data structure is a bounding volume hierarchy (BVH), a tree where each of its nodes is represented by a bounding volume that is the union of the bounding volumes of its children. A bounding volume of a leaf node then encompasses all objects in the leaf.

Memory requirements for a BVH can be easily pre-computed, since there are no duplicated geometric primitives in leaves as is the case for e.g. a kd-tree. The disadvantage is that a BVH traversal algorithm must check all children along the ray path, since the nodes might spatially overlap. However, if a ray has already found an intersection during traversal it may use this information to skip traversing some nodes.

Usage of BVHs in ray tracing dates to Rubin and Whitted [RW80]. Kay and Kajiya [KK86] used the spatial median to construct a BVH, while Goldsmith and Salmon [GS87] proposed the aforementioned SAH in the context of insertion based BVH build algorithms. We are not aware of a recent survey about the usage of BVHs for ray tracing, but e.g. Bittner et al. [BHH15] summarized all state-of-the-art BVH algorithms.

1.2 Hardware acceleration

The orientation of this research is towards efficient data structures and related algorithms for ray tracing with regard also to hardware architectures, that are able to accelerate ray tracing by means of data level parallelism e.g. single instruction multiple data (SIMD) instructions, multi-threaded parallel execution or even with a completely specialized ray tracing units. This section will shortly summarize the major hardware architectures related to ray tracing.

In 2002 Schmittler et al. [SWS02] presented the SaarCOR, providing a VLSI design and an implementation of a ray tracing chip on a FPGA. It was split into scalable shading and ray tracing core units and a memory management module. As a spatial data structure SaarCOR used a kd-tree. The tree is split based on an estimation of the cost of traversal operations versus intersection operations in the hardware. SaarCOR provided ray tracing and simple non-programmable shading on static scenes, but the algorithm to build a kd-tree was not implemented in hardware.

Woop et al. [WSS05] progressed the usage of specialized hardware for ray tracing even further in 2005 with a fully shader programmable chip, the Ray Processing Unit (RPU). At that point rendering a complex scene in high resolution still required a cluster of CPUs and the expectations of Woop et al. were that widespread use of multi-core CPUs was still 5 to 10 years

away. Graphic processing units (GPUs) already had a similar hardware architecture, but their programming model was rather simple. RPU used traversal processing units to traverse chunks of rays through a kd-tree, exploiting coherency to alleviate memory access costs with a shared so-called mailboxed list processing unit acting like a cache. Shader processing units were used for geometry intersection and programmable shading computations.

In the same year Thrane and Simonsen [TS05] compared several algorithmic techniques used to map ray tracing to a GPU. These methods used fragment and shader programs and in a way supported the notion that GPUs are currently unsuitable for a fast ray tracing implementation as compared to custom hardware. They used textures to pass different pre-computed spatial data structures to the GPU, on which they ran traversal programs using the fragment processor.

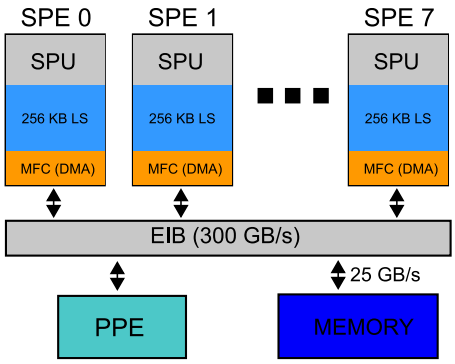


Figure 1.2: *IBM Cell consists of eight SPEs with 256kB of local memory that communicate with main memory and the PPE through the Element Interconnect Bus (EIB).*

Also in 2005 Sony, Toshiba and IBM released the IBM Cell Broadband Engine. It is a microprocessor designed for computationally intensive, mainly multimedia, tasks. Cell consists of one PPE (Power Processor Element), a Power architecture based processor, and eight Synergistic Processor Elements (SPEs) (see Figure 1.2). SPE is a RISC processor with most of its instructions being 128-bit wide SIMD. The PPE is intended as a work distributor with SPEs as worker units.

Cell's primary usage was in a gaming console, the Sony Playstation 3, but in the fall of 2006 IBM released the QS20 blade module as a computational unit. IBM supported ray tracing oriented research which resulted in the iRT by Minor et al. [MNM06], but the seminal work towards a Cell ray tracer was published by Benthin et al. [BWSF06]. Their heavily optimized implementation achieved traversal performance on a single SPU on par with x86 processors of that time.

In 2006 Nvidia released a general purpose GPU (GPGPU) computing platform known as CUDA (not an acronym, named after the Plymouth Barracuda). CUDA extends C/C++ and Fortran to give the developer access to the many-core computing capability of an Nvidia GPU multiprocessors and its memory. The basic element of a CUDA computation is a kernel, a function that is executed by each GPU thread where the number of concurrent threads depends on the hardware and runs in tens of thousands. The threads are, however, light-weight as compared to their CPU counterparts and their capabilities are limited.

A ray tracing traversal framework on the first CUDA capable Tesla GPU architecture was published in 2009 by Aila et al. [AL09]. Later they have added support for Fermi and Kepler architectures [ALK12]. Aila et al. proposed a number of algorithmic improvements to maximize GPU utilization and the implementation was able to cast over 100 million primary rays per second on Tesla and almost half a billion primary rays on Kepler. In 2010 Nvidia OptiX, a general programmable CUDA ray tracing framework that parallelize both building of spatial data structures and their traversal, was presented by Parker et al. [PBD+10].

We also have to mention general purpose architectures, such as the most common x86. These are utilized for high performance ray tracing mainly with explicit or implicit usage of SIMD instruction sets. First effort in this direction was the MMX instruction set, followed by SSE and later AVX sets. A current example of a ray tracer using all these features is the Embree [WWB+14], a collection of ray tracing kernels developed by Intel. It is not a complete rendering system, but rather a highly optimized implementation of ray tracing algorithms based on a BVH for different combination of instruction set extensions.

2 Aims of the doctoral thesis

Although ray tracing has been known for over four decades, it is still considered relatively slow to be massively used in interactive applications, particularly for animated scenes. Real-time rendering has traditionally been the domain of rasterization but with the rise of computational power and development of new parallel processing architectures, e.g. modern GPUs, ray tracing based methods are becoming viable alternatives and even provide ready access to computation of complex global illumination effects that are difficult or impossible to produce with rasterization based renderers.

The focus of the doctoral thesis is the development of new or improving on existing spatial data structures and associated algorithms for ray tracing to be used for efficient rendering of complex 3-dimensional virtual scenes.The applicability of this research is in all fields that are related to computer-generated imagery (CGI), but also others, such as lighting design or collision detection.

3 Working methods

First and foremost, the doctoral thesis is submitted as a collection of already published texts authored not only by the submitter. It contains published contributions in the form of journal and/or conference papers and the papers are also referenced where applicable. This chapter will summarize the content of each paper.

Review: Kd-tree Traversal Algorithms for Ray Tracing

In [HH11] we review the traversal algorithms for kd-trees for ray tracing. First we briefly introduce build algorithms and then continue with the description of basic traversal algorithms published prior to the year 2000: a sequential algorithm, a stack-based algorithm and those

based on neighbour-links. These have different limitations, which led to several new developments. We describe algorithms exploiting ray coherence and algorithms designed with specific hardware architecture limitations such as memory latency in mind. Memory consumption is taken into account with the description of different memory layouts of nodes (see Figure 3.1) and sub-trees. We also discuss the issue of robustness of traversal algorithms. A summary table of all traversal algorithms with their memory requirements is included to help with design choices.

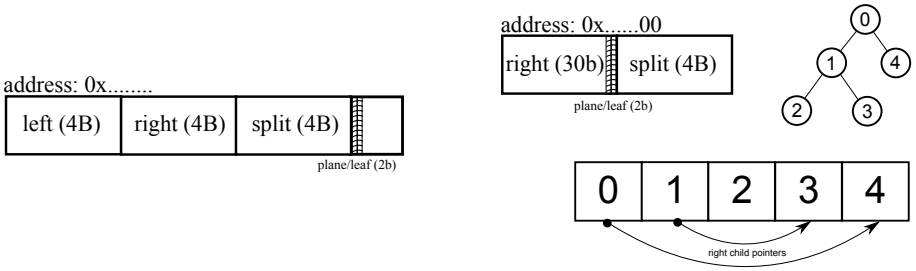


Figure 3.1: *Memory layout of a (a) kd-tree node for basic layout using 13 Bytes and (b) memory aligned and condensed layout with an implicit left child pointer reduced to 8 Bytes to a node with an example of a simple tree.*

Efficient Stack-less BVH Traversal for Ray Tracing

In [HDW+11] we propose a new, completely iterative traversal algorithm for ray tracing bounding volume hierarchies that is based on storing a parent pointer with each node, and on using simple state logic to infer which node to traverse next. This logic is based on three states which depend on how was the current node reached during the traversal, i.e. coming from its child, its sibling or its parent node (see Figure 3.2). Though our traversal algorithm does re-visit internal nodes, it intersects each visited interior node only once, and in general performs exactly the same ray-box tests and ray-primitive intersection tests, and in exactly the same order, as a traditional stack-based traversal algorithm. The chapter includes a listing with commentary of the algorithm in pseudo-code.

We also show a listing that formed a base of our implementation in CUDA, where we condensed and reordered the algorithm for efficiency reasons so as to have less diverging warps. Results were measured on an integration with the freely available Aila et al. CUDA ray tracer [AL09, KAL09]. The proposed algorithm can be used for computer architectures that need to minimize the use of local memory for processing rays or those that need to minimize the data transport such as distributed multi-CPU architectures.

When It Makes Sense to Use Uniform Grids for Ray Tracing

In [HKH11] we describe a hybrid algorithm that improves on performance by using a two-step approach. The algorithm uses a calibration phase requiring a set of scenes of different properties such as the number of geometric primitives or their spatial distribution. During this phase we measure the implementation efficiency constants of the building of the data structures and their traversal algorithms.

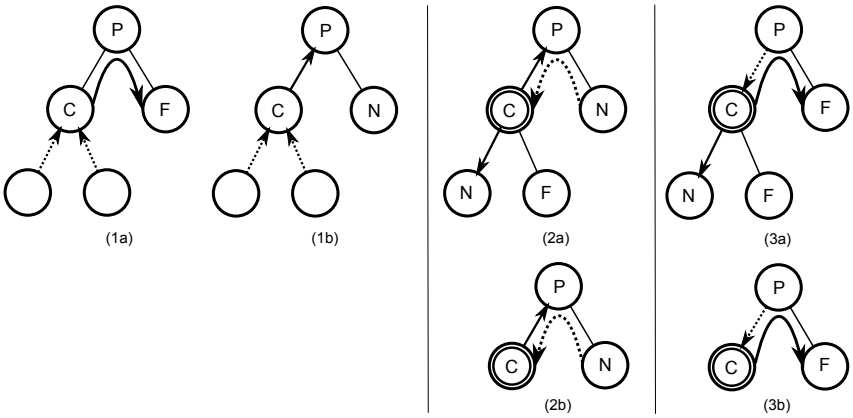


Figure 3.2: *Traversal states: 1. from child, 2. from sibling and 3. from parent. C is the current node, P is parent of C, N and F signify near and far nodes with regard to the current ray. Dotted lines show the traversal we have taken into the current node whereas thick lines show next traversal step. Doubled rings signify where a ray-box test is needed to decide where to traverse next.*

For an unknown scene to be ray traced we build a uniform grid and test its performance by sampling a small set of representative rays. Second, using an estimate on the number of rays to be queried we either continue using the grid or build a hierarchical data structure instead. Commonly used data structures have a rather high build time while building a uniform grid can be done much faster. This way we select a more efficient data structure given a particular implementation of the algorithms which yields with high probability an overall smaller computation time.

We evaluated the properties of this method for a set of 28 scenes. The computation was repeated 5000 times for combinations of n random scenes for calibration and 28 minus n scenes for estimation. For random rays the estimate was about 25 percent more optimistic about the quality of the uniform grid, i.e. the error of the estimate was about 25 percent (see Figure 3.3).

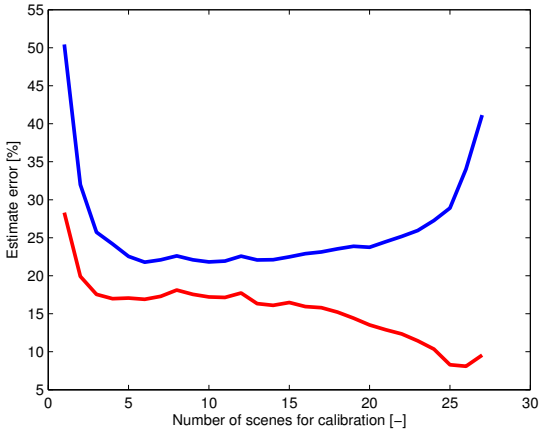


Figure 3.3: *Uniform grid quality estimate error based on a small set of random rays. Red line is the average of estimate errors. Blue line is the average of absolute values of these errors.*

Fast Insertion-Based Optimization of Bounding Volume Hierarchies

In [BHH13] we present an algorithm for fast optimization of bounding volume hierarchies (BVH) for efficient ray tracing. We perform selective updates of the hierarchy driven by the cost model derived from the surface area heuristic. In each step the algorithm updates a fraction of the hierarchy in order to minimize the overall hierarchy cost. Nodes to be updated can be chosen randomly or based on an inefficiency metric. We discuss three types of such metrics and their correlation with the final cost of the tree and the number of updates required to reach this number. The updates themselves are then realized by simple operations on the tree nodes: removal, search, and insertion. For each updated node we remove both its children from the tree, find a position with lower overall cost for each and then reinsert them (see Figure 3.4).

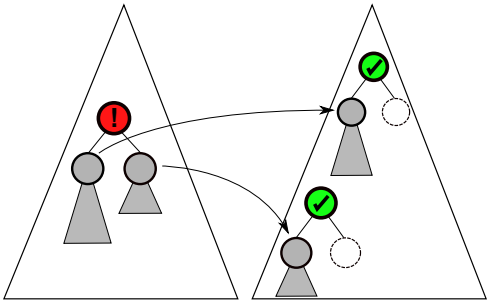


Figure 3.4: . *Removal of inefficient nodes from the tree and re-insertion of their children to positions that decrease the overall cost of the tree.*

Our method can quickly reduce the cost of the hierarchy constructed by the traditional techniques such as the surface area heuristic. We evaluate the properties of the proposed method on fourteen test scenes of different complexity including individual objects and architectural scenes. We also compare our algorithm to tree rotation methods based on hill climbing and simulated annealing as a state of the art at the time. The results show that our method can improve a BVH initially constructed with the surface area heuristic by up to 27% and a BVH constructed with the spatial median split by up to 88%.

Massively Parallel Hierarchical Scene Processing with Applications in Rendering

In [VBHH13] we present a method for massively parallel hierarchical scene processing on the GPU based on a sequential decomposition of a given hierarchical algorithm into small functional blocks. The computation is fully managed by the GPU using a specialized task pool which facilitates synchronization and communication of processing units, in this case persistent CUDA warps (see [AL09]. We try to maximize the GPU utilization by using as many threads as possible for a given task and also by computing tasks in different phases of the algorithm at the same time. We present two applications of the proposed approach: construction of the bounding volume hierarchies and collision detection based on divide-and-conquer ray tracing (see Figure 3.5).

We have compared our method to the HLVBH by Garanzha et al. [GPM11] with different settings. The results indicate that using our approach we achieve high utilization of the GPU even for complex hierarchical problems which pose a challenge for massive parallelization.

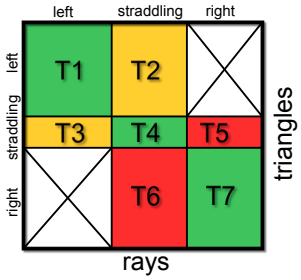


Figure 3.5: *Matrix representing a subdivision of the task into its child tasks for divide-and-conquer ray tracing.*

Incremental BVH Construction for Ray Tracing

In [BHH15] we propose a new method for incremental construction of Bounding Volume Hierarchies (BVH). This method has at its core an insertion algorithm published in [BHH13], but instead of just optimizing an already built BVH we rather create a new one from scratch. We use insertions to incrementally build the BVH interleaving it with global updates of the tree (see 3.6). Despite the belief that the incremental construction of BVH is inefficient we show that our method incrementally constructs a BVH with quality comparable to the best SAH builders. We illustrate the versatility of the proposed method using a flexible parallelization scheme that opens new possibilities for combining different BVH construction heuristics.

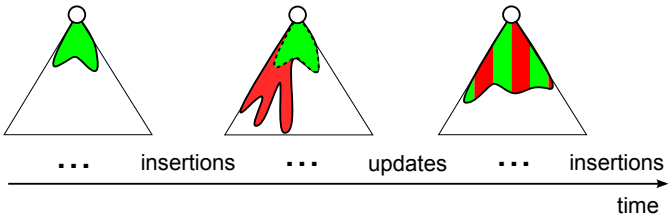


Figure 3.6: *The incremental insertion of nodes is searching for the best position of inserted nodes, however the overall structure of the tree might get imbalanced. This is corrected by BVH updates, which aim to globally optimize the current tree.*

Since our method is able to insert additional geometry without loss of performance we demonstrate the usage of the method in a proof-of-concept application for real-time preview of data streamed over the network. We also show different approaches to selecting relevant triangles to be sent according to a current camera view. The client application uses CPU to construct the BVH and CUDA based GPU ray tracer to render the image. We also report results of our build algorithm for four different settings compared to a BVH constructed with a high-quality SAH builder and a BVH built with spatial median splits.

4 Results

In this chapter we will summarize the results of the papers included in the doctoral thesis and discuss their impact on the ray tracing research.

In [HH11] we have summarized the traversal algorithms for kd-trees used in ray tracing. We

have described several basic traversal algorithms and also newer developments motivated by specific hardware issues such as lack of caches, the memory latency, and the penalty for performing branches. We have shown that the traversal algorithms can be interconnected to the kd-tree build algorithm as specific data may be needed to be precomputed and stored in the kd-tree representation. We have also discussed the memory layouts for this representation that can be optimized in particular for larger data sets or when limited by memory of a specific hardware. Finally, we have described several approaches for ray tracing of many rays at once and how the coherence of such rays can be exploited for better performance.

This paper was one of the most accessed on the Computer Graphics Forum website during 2011 [Joh12] and it was also cited as a survey article for kd-trees in a number of publications. The context of these publications was not only computer graphics [CCI13] [Mknd12] [WGD14] [YL14] [XCHZ13], but also artificial intelligence [CS13] or nuclear energy [CSZ+15].

In [HDW+11] we have presented a traversal algorithm for bounding volume hierarchies that does not need a stack and hence minimizes the memory needed for a ray. It is based on a three-state logic and keeping a parent link for all nodes. The proposed algorithm can be used efficiently in approaches where we process many rays in parallel. In these cases we need to minimize the memory usage for individual rays either locally or for data transfer among processing units. The algorithm also does only the necessary minimum of ray-box intersections, as would a stack-based algorithm do.

We have shown the results for implementations in CUDA for Tesla and Fermi architecture as the most commonly accessible highly parallel architectures at the time. We show that for these GPU architectures the traversal algorithm with local stack is more efficient than stack-less algorithm that needs twice as many traversal steps. Even though employing a stack demands frequent access to memory, modern GPUs can run thousands of threads at once and effectively hide memory latencies.

There are, however, hardware architectures or applications where having minimal memory per ray is paramount. These are e.g. special hardware units, memory distributed CPU/GPU architectures designed for tracing rays or ray-reordering traversal schemes and the usefulness of our algorithm on such an architecture was shown in FlexRender by Somers and Wood [SW13]. They have used an extension of our algorithm for a distributed rendering architecture used on commodity hardware. Their system balances load by suspending traversal on one computation node and resuming it at another. Our stackless approach helped them to minimize the amount of data required for a complete ray traversal state that needs to be transferred.

Kopta et al. [KSS+13, KSS+14] cited this paper in the context of special hardware that focuses on optimizing memory and energy consumption. Kim et al. [KJN14] used R-trees for range queries in large datasets on GPUs. Since R-trees are similar to BVHs the memory optimization ideas in our paper were applicable too. Gribble et al. [GFE+12] implemented a path tracer using OpenCL for their ray tracing visualization toolkit and our stackless traversal improved their performance. The paper was also referenced as a related work by Wu et al. [WDZ13], García et al. [GJK13] and Liu et al. [LCD15].

In [HKH11] we have proposed an algorithm that combines a uniform grid and a hierarchical data

structure for ray tracing so that it takes advantage of both types. Based on the scene properties and a small number of rays computed using the grid we decide either to continue ray tracing with the grid or to build a hierarchical data structure such as a kd-tree. To our best knowledge we presented the first algorithm that decides when it is advantageous to use uniform grids in dependence on the number of rays to be shot.

We have shown that the use of uniform grids is relatively limited for standard scenes with the exception of scenes with a special distribution of geometric primitives in space. But our results also show that the number of rays when it does not pay off to build a hierarchical structure can be significant. The paper was cited by Wu et al. [WYB13] as a related work.

In [BHH13] we proposed an algorithm for building a high quality BVH by incrementally updating a tree initially constructed by a top down method with surface area heuristic. The method is based on performing selective updates of the BVH by identifying inefficient nodes and reinserting them back in positions that will lower the total BVH cost. The updates are prioritized and thus the algorithm can be tuned to give preference to the update time or the quality of the hierarchy. The tree is also compacted with removal of sub-trees whose SAH cost is larger than a simple leaf node with all triangles in such a sub-tree. We have shown that for complex scenes our method achieves very good cost reduction in much shorter time than previous methods.

Gu et al. [GHFB13] referenced the BVH compaction optimization as similar to, and perhaps and idea for, their sub-tree flattening. The core algorithm was compared to other state-of-the-art methods by Aila et al. [AKL13]. Their results confirmed that our method constructs high-quality trees when compared by SAH costs and also to their new end-point overlap metric. However, mediocre results were achieved for a second newly proposed leaf count variability metric and why this happened is a matter of future work.

In [VBHH13] we have proposed a novel method for massively parallel processing in the context of hierarchical algorithms dealing with 3D geometrical data. Our method runs entirely on the GPU and requires no management of the computation from the CPU side. We propose a methodology of subdividing a given hierarchical algorithm into tasks, phases, steps, and work chunks in order to map the algorithm to the parallel framework. We show two applications of our method: construction of the BVH and divide-and-conquer ray tracing on the GPU. We evaluated two proof of concept applications, which indicate that our approach has a good potential for massive parallelization of complex hierarchical problems. This paper was referenced as a related work by Leyre et al. [LRA$^+$14] in Optics Express journal.

In [BHH15] we have proposed an incremental BVH construction algorithm, which builds a BVH with better or comparable quality to the traditional SAH based top-down BVH construction methods. We have implemented a sequential and a parallel version of this algorithm, achieving construction speeds of 0.8 and 2.9 million triangles per second respectively. This makes the proposed method significantly faster compared with the reference implementation of the precise top-down SAH build.

We have shown a possible application of the method for real-time ray tracing of scenes which are streamed over a network with a proof-of-concept client-server application. This application uses GPU ray tracing, while the networking layer and the incremental BVH construction is

implemented on the CPU. We have used several simple prioritization schemes allowing for fast previewing of large data sets even in the case of a low network bandwidth. This paper was published quite recently and there were no known references at the time of the publication of this thesis statement.

5 Conclusion

In conclusion, the papers presented in the doctoral thesis improved upon the performance of build and traversal algorithms for ray tracing data structures either directly by contributing a novel approach or indirectly as a survey. These papers were referenced in a number of publications in and also outside of the realm of computer graphics.

According to the citations we believe that ideas and techniques presented in the doctoral thesis will have applications in different scientific fields where ray tracing is used and they shall further the research about ray tracing itself.

6 Bibliography

[AL09] T. Aila and S. Laine. Understanding the Efficiency of Ray Traversal on GPUs. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG'09, pages 145–149, New York, NY, USA, 2009. ACM.

[ALK12] T. Aila, S. Laine, and T. Karras. Understanding the Efficiency of Ray Traversal on GPUs – Kepler and Fermi Addendum. NVIDIA Technical Report NVR-2012-02, NVIDIA Corporation, June 2012.

[App68] A. Appel. Some Techniques for Shading Machine Renderings of Solids. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, AFIPS '68 (Spring), pages 37–45, New York, NY, USA, 1968. ACM.

[BWSF06] C. Benthin, I. Wald, M. Scherbaum, and H. Friedrich. Ray Tracing on the Cell Processor. In *Proc. IEEE Symposium on Interactive Ray Tracing 2006*, pages 15–23, September 2006.

[FTI86] A. Fujimoto, T. Tanaka, and K. Iwata. ARTS: Accelerated Ray-Tracing System. *Computer Graphics and Applications, IEEE*, 6(4):16–26, April 1986.

[Gla89] A. S. Glassner, editor. *An Introduction to Ray Tracing*. Academic Press Ltd., London, UK, 1989.

[GPM11] K. Garanzha, J. Pantaleoni, and D. McAllister. Simpler and faster HLBVH with Work Queues. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG'11, pages 59–64, New York, NY, USA, 2011. ACM.

[GS87] J. Goldsmith and J. Salmon. Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, May 1987.

[Hav00] V. Havran. *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000.

[Joh12] John Wiley & Sons, Inc. Computer Graphics Forum: Most Accessed Articles, 2012. `http://web.archive.org/web/20120702235331/http://onlinelibrary.wiley.com/journal/10.1111/(ISSN)1467-8659/homepage/MostAccessed.html`.

[KAL09] T. Karras, T. Aila, and S. Laine. Understanding the Efficiency of Ray Traversal on GPUs; Google Code, 2009. `http://code.google.com/p/understanding-the-efficiency-of-ray-traversal-on-gpus/`.

[Kap85] M. Kaplan. Space-Tracing: A Constant Time Ray-Tracer. In *SIGGRAPH '85 State of the Art in Image Synthesis seminar notes*, pages 149–158, July 1985.

[KK86] T. L. Kay and J. T. Kajiya. Ray Tracing Complex Scenes. In D. C. Evans and R. J. Athay, editors, *SIGGRAPH '86 Proceedings*, volume 20, pages 269–278, August 1986.

[MNM06] B. Minor, M. Nutter, and J. Madruga. iRT: An Interactive Ray Tracer for the CELL Processor. Technical report, 2006.

[PBD+10] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich. OptiX: A General Purpose Ray Tracing Engine. In *ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, pages 66:1–66:13, New York, NY, USA, 2010. ACM.

[RW80] S. M. Rubin and T. Whitted. A 3-Dimensional Representation for Fast Rendering of Complex Scenes. In *SIGGRAPH '80 Proceedings*, volume 14, pages 110–116, July 1980.

[SWS02] J. Schmittler, I. Wald, and P. Slusallek. SaarCOR – A Hardware Architecture For Ray Tracing. In *Proceedings of the conference on Graphics Hardware 2002*, pages 27–36. Saarland University, Eurographics Association, 2002. available at http://www.openrt.de.

[TS05] N. Thrane and L. O. Simonsen. A comparison of acceleration structures for GPU assisted ray tracing. Technical report, 2005. University of Aarhus.

[WH06] I. Wald and V. Havran. On building fast kd-trees for ray tracing, and on doing that in O(N log N). In *Proceedings of IEEE Symposium on Interactive Ray Tracing 2006*, pages 61–69, September 2006.

[WSS05] S. Woop, J. Schmittler, and P. Slusallek. RPU: A Programmable Ray Processing Unit for Realtime Ray Tracing. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 434–444, New York, NY, USA, 2005. ACM.

[WWB+14] I. Wald, S. Woop, C. Benthin, G. S. Johnson, and M. Ernst. Embree: A Kernel Framework for Efficient CPU Ray Tracing. *ACM Trans. Graph.*, 33(4):143:1–143:8, July 2014.

7 Publications of the Author

Publications in high-impact journals:

[HH11] **M. Hapala** and V. Havran. Review: Kd-tree Traversal Algorithms for Ray Tracing. *Computer Graphics Forum*, 30(1):199-213, Mar 2011.
Contribution: M. Hapala 50%, V. Havran 50%.

[CCI13] Choi B., Chang B. and Ihm I.. Improving Memory Space Efficiency of Kd-tree for Real-time Ray Tracing. *Computer Graphics Forum*,32: 335–344, 2013.

[Mknd12] R. Mukundan. Collision Detection. *Advanced Methods in Computer Graphics*, 231-276, 2012.

[WGD14] Wang Y.,Guo P. and Duan F.. A Fast Ray Tracing Algorithm Based on a Hybrid Structure. *Multimedia Tools and Applications*, 1-16, 2014.

[YL14] Yin M. and Li S.. Fast BVH Construction and Refit for Ray Tracing of Dynamic Scenes. *Multimedia Tools and Applications*, 1823-1839, 2014.

[XCHZ13] Xiao K., Chen D., Hu X. and Zhou B.. Shell: A Spatial Decomposition Data Structure for 3D Curve Traversal on Many-Core Architectures. *Algorithms – ESA 2013*, 815-826, 2013.

[CS13] F. O. Cabrera and M. Sánchez-Marré. Using NIAR k-d Trees to Improve the Case-Based Reasoning Retrieval Step. *Advances in Soft Computing and Its Applications*, 314-325, 2013.

[CSZ+15] Chen Z., Song J., Zheng H., Wu B. and Hu L.. Optimal Spatial Subdivision Method for Improving Geometry Navigation Performance in Monte Carlo Particle Transport Simulation. *Annals of Nuclear Energy*, 314-325, 2015.

[LBF13] Lu H., Bao P. and Feng J.. OpenCL-based Real-time KD-tree and Ray-tracing for Dynamic Scene. *Journal of Computer-Aided Design and Computer Graphics*, 963-973, 2013.

[BHH13] J. Bittner, **M. Hapala**, V. Havran. Fast Insertion-Based Optimization of Bounding Volume Hierarchies. *Computer Graphics Forum*, 32(1):85-100, Feb 2013.
Contribution: J. Bittner 50%, M. Hapala 25%, V. Havran 25%.

[GHFB13] Y. Gu, Y. He, K. Fatahalian and G. Blelloch. Efficient BVH Construction via Approximate Agglomerative Clustering. *Proceedings of the 5th High-Performance Graphics Conference*, pages 81-88, 2013.

[AKL13] T. Aila, T. Karras and S. Laine. On Quality Metrics of Bounding Volume Hierarchies. *Proceedings of the 5th High-Performance Graphics Conference*, pages 101-107, 2013.

[PJR+14] J. Parulek, D. Jönsson, T. Ropinski, S. Bruckner, A. Ynnerman and I. Viola. Continuous Levels-of-Detail and Visual Abstraction for Seamless Molecular Visualization. *Computer Graphics Forum*, 33: 276-287. 2014.

[VBHH13] M. Vinkler, J. Bittner, V. Havran, **M. Hapala**. Massively Parallel Hierarchical Scene Processing with Applications in Rendering. *Computer Graphics Forum*, 32(8):13-25, Dec 2013.
Contribution: M. Vinkler 50%, J. Bittner 20%, V. Havran 20%, M. Hapala 10%.

[LRA+14] S. Leyre, J. Ryckaert, P. Acuña, J. Audenaert, Y. Meuret, G. Durinck, J. Hofkens, G. Deconinck and P. Hanselaer. A Hybrid Tool for Spectral Ray Tracing Simulations of Luminescent Cascade Systems. *Optics Express 22* , 20:24582-24593, Oct 2014.

[BHH15] J. Bittner, **M. Hapala**, V. Havran. Incremental BVH Construction for Ray Tracing. *Computers & Graphics*, pages 135-144, Apr 2015.
Contribution: J. Bittner 50%, M. Hapala 30%, V. Havran 20%.

Publications excerpted by ISI:

[HKH11] **M. Hapala**, O. Karlík, and V. Havran. When It Makes Sense to Use Uniform Grids for Ray Tracing. *In Proceedings of WSCG 2011*, Communication Papers, pages 193-200, Feb 2011.
Contribution: M. Hapala 35%, O. Karlík 35%, V. Havran 30%.

[WYB13] Wu Z., Yu H. and Bin C.. Divide and Conquer Ray Tracing Algorithm Based on BVH Partition. *2013 International Conference on Virtual Reality and Visualization (ICVRV)*, pages 49-55, Sept 2013.

Other publications:

[HDW+11] **M. Hapala**, T. Davidovič, I. Wald, V. Havran, and P. Slusallek. Efficient Stack-less BVH Traversal for Ray Tracing. In *27th Spring Conference on Computer Graphics (SCCG 2011)*, pages 29-34, Apr 2011.
Contribution: M. Hapala 25%, T. Davidovič 25%, I. Wald 25%, V. Havran 23%, P. Slusallek 2%.

[KSS+13] D. Kopta, K. Shkurko, J. Spjut, E. Brunvand and A. Davis. An Energy and Bandwidth Efficient Ray Tracing Architecture. *Proceedings of the 5th High-Performance Graphics Conference*, pages 121-128, 2013.

[KSS+14] D. Kopta, K. Shkurko, J. Spjut, E. Brunvand and A. Davis. Memory Considerations for Low Energy Ray Tracing. *Computer Graphics Forum*, 2014.

[BA14] R. Barringer and T. Akenine-Möller. Dynamic Ray Stream Traversal. *ACM Trans. Graph.*, pages 151:1–151:9, Jul 2014.

[GFE+12] C. Gribble, J. Fisher, D. Eby, E. Quigley and G. Ludwig. Ray Tracing Visualization Toolkit. *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 71–78, 2012.

[AS14] A. T. Áfra and L. Szirmay-Kalos. Stackless Multi-BVH Traversal for CPU, MIC and GPU Ray Tracing. *Computer Graphics Forum*, 33:129–140, 2014.

[LCD15] Liu B. and G. J. Clapworthy and Dong F.. IsoBAS: A binary accelerating structure for fast isosurface rendering on {GPUs}. *Computers & Graphics*, 2015.

[GJK13] M. Goldfarb, Y. Jo and M. Kulkarni. General Transformations for GPU Execution of Tree Traversals. *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 10:1–10:12, 2013.

[WDZ13] Wu F., Dong J. and Zhou F.. Indirect Illumination Algorithm Based on Cone-rays Cast. *Computer Engineering*, 294-297, 2013.

[SW13] B. Somers and Z. J. Wood. FlexRender: A Distributed Rendering Architecture for Ray Tracing Huge Scenes on Commodity Hardware. *GRAPP/IVAPP'13*, 152-164, 2013.

[KJN14] Kim J., Jeong W. and Nam B.. Exploiting Massive Parallelism for Indexing Multi-dimensional Datasets on the GPU. IEEE Transactions on *Parallel and Distributed Systems*, 2014.

[GJKR13] A. García, S. Murguia, U. Olivares and F. F. Ramos . Fast Parallel Construction of Stack-less Complete LBVH Trees with Efficient Bittrail Traversal for Ray Tracing. *Proceedings of the 13th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry*, pages 151-158, 2014.

8 Summary

Current trends in computer graphics used in film or video games industry lead to an increasing demand for methods than can deliver interactive high quality image synthesis with global illumination effects. Many of these algorithms use in their core a simple visibility computation, ray tracing, that computes intersections with primitives in a virtual scene along an oriented halfline. This thesis focuses on improving data structures and algorithms used in ray tracing. It also takes into consideration specialized hardware, such as graphics processing units, which can be used to increase rendering performance thanks to its parallel design.

This thesis is submitted as a collection of papers published in various journals by a group of authors including the submitter in the period of 2011 to 2015. We thoroughly review algorithms for traversing a kd-tree, a hierarchical data structure, with a focus on algorithms implemented on a specialized hardware. We also propose five novel algorithms: an iterative bounding volume hierarchy traversal that can traverse the tree structure without the use of a stack, a hybrid algorithm that combines the use of two different spatial data structures, a new method for incremental construction of bounding volume hierarchies, a method for massively parallel hierarchical scene processing on the GPU and, finally, an algorithm for fast optimization of bounding volume hierarchies based on selective updates.

Současné trendy v počítačové grafice používané ve filmové produkci nebo videoherním průmyslu vedou ke zvýšenému zájmu o metody, které dokážou přinést interaktivní syntézu obrazu s efekty globálního osvětlení. Mnoho z těchto algoritmů ve svém jádru používá jednoduchý výpočet viditelnost, tzv. sledování paprsku, který spočítá průsečíky mezi primitivy ve virtuální scéně a polopřímkou. Tato doktorská práce se soustředí na zlepšení datových struktur a algoritmů používaných právě pro metodu sledování paprsku. Bere do úvahy i specializovaný hardware, jako například grafické procesory, který se díky svému paralelnímu návrhu dá použít pro zrychlení vykreslování scény touto metodou.

Tato doktorská práce je předložena jako soubor již zveřejněných článku skupinou autorů včetně doktoranda v rozmezí let 2011 až 2015. Nejdříve důkladně zmapujeme algoritmy pro traverzování kd-stromu se zaměřením na algoritmy implementované na specializovaném hardware a dále navrhneme pět nových algoritmů: iterativní traverzaci hierarchie obálek bez použití zásobníku, hybridní algoritmus, který kombinuje použití dvou různých prostorových datových struktur, novou metodu pro inkrementální stavbu hierarchie obálek, metodu pro masivně paralelní zpracování scény na grafické kartě a algoritmus pro rychlou optimalizaci hierarchie obálek pomocí selektivní aktualizace.

Michal Hapala

CONTACT

Vítkova 262/4
Praha
186 00, Czech Republic

mikee@mikee.cz
http://www.mikee.cz
http://dcgi.felk.cvut.cz/home/hapalmic/

WORK EXPERIENCE

Game and Animations Programmer, Warhorse Studios 02/2012 - present

Researcher, Czech Technical University in Prague 09/2009 - 04/2013

Lecturer, Czech Technical University in Prague 03/2008 - 02/2014

Nintendo DS Programmer, Cinemax 03/2008 - 11/2009

Tools Programmer, Centauri Production 11/2006 - 02/2008

EDUCATION

Czech Technical University in Prague, *Faculty of Electrical Engineering* 09/2003 - 02/2009

Ing. (eq. to MSc.)
Study programme: Electrical Engineering and Computer Science
Field of study: Computer Graphics
Master's thesis: "Data Structures for Ray Tracing on Specialized Hardware"
Awarded first place in IBM Student Research Projects 2009

Bc. (eq. to BSc.)
Study programme: Electrical Engineering and Computer Science
Field of study: Computer Science
Bachelor's thesis: "Programming Techniques for the Development of Massive Multiplayer On-line Games"

ACTIVITIES

Chair of student volunteers
Eurographics 2007 (with Adam J. Sporka)

Reviewer
Siggraph Asia 2011, CAD/Graphics 2013

Volunteer
GDC Europe 2010, 2011

SKILLS

Programming languages:	C++ (5+ years); C# (2 years), GLSL/HLSL (familiar)
Frameworks, libraries and APIs:	CUDA (2 years), OpenGL (2 years), XNA, Qt, Maya SDK, 3ds Max SDK, OpenCollada, FBX SDK, Google Sketchup C++ API
Passive knowledge:	Assembly, Java, PHP, JavaScript, SQL
IDEs:	Microsoft Visual Studio, CodeWarrior

Cambridge Certificate of Proficiency in English

LANGUAGES

English	fluent
Czech, Slovak	native