CZECH TECHNICAL UNIVERSITY IN PRAGUE

Doctoral Thesis Statement

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

Daniel Meister

Bounding Volume Hierarchies for High-Performance 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, May 2018

The doctoral thesis was produced in full-time manner Ph.D. study at the Department of Computer Graphics and Interactionof the Faculty of Electrical Engineering of the CTU in Prague.

Candidate:
Daniel Meister
Department of Computer Graphics and Interaction
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo nám. 13, 121 35, Prague 2, Czech Republic

Thesis Supervisor:
Jiří Bittner
Department of Computer Graphics and Interaction
Faculty of Electrical Engineering
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.

Abstract

Photorealistic image synthesis relies heavily on the concept of ray tracing. The problem of ray tracing is to find the nearest intersection between a given ray and scene primitives. Although this problem is geometrically quite simple; in practice, it is necessary to test millions of rays against millions of scene primitives. To improve the efficiency of ray tracing, scene primitives are usually arranged into various acceleration data structures. This thesis focuses on the bounding volume hierarchy (BVH), which is one of the most popular acceleration data structures for ray tracing.

The thesis is a compilation of research papers published in journals with impact factor with the introductory text describing the contributions and putting the individual papers in a common context. The work consists of five novel methods addressing the BVH for high-performance ray tracing. The first one is a technique for handling dynamic geometry based on constructing a single BVH taking into account geometric changes through the animation. The second one is a parallel BVH construction algorithm based on a combination of the k-means algorithm and agglomerative clustering. The third one is a parallel BVH construction algorithm based on the progressively refined cut of an existing BVH targeting multi-core CPUs. The fourth one is a GPU-based algorithm based on locally-ordered clustering building the BVH in a bottom-up manner by merging a batch of cluster pairs in each iteration. The last one is a parallel insertion-based optimization for BVHs targeting contemporary GPU architectures. Each of these methods to a certain extent advances the current state-of-the-art in ray tracing.

Abstrakt

Fotorealistická syntéza obrazu staví na algoritmu sledování paprsku, který řeší problém nalezení nejbližšího průsečíku mezi daným paprskem a geometrickými primitivy scény. Přestože je tento problém geometricky velmi jednoduchý, v praxi musíme nalézt miliony nejbližších průsečíků ve scénách obsahující až miliony geometrických primitiv. Abychom výpočet urychlili, uspořádáváme geometrická primitiva do různých akceleračních datových struktur. Tato práce se zabývá hierarchií obálek, jednou z nejpoužívanějších akceleračních datových struktur pro urychlení algoritmu sledování paprsku.

Tato práce je kolekce výzkumných článků publikovaných v časopisech s impakt faktorem zasazená do společného kontextu pomocí integrujícího textu. Tato práce obsahuje pět nových metod založených na urychlení sledování paprsku pomocí hierarchie obálek. První metoda se zabývá efektivní aktualizací hierarchií obálek v kontextu dynamických scén, která je založená na jedné hierarchii obálek, která je optimalizovaná pro celou animační sekvenci. Druhá metoda se zabývá paralelní stavbou hierarchie obálek pomocí aglomerativního shlukování a shlukování pomocí k-středů. Třetí metoda se zabývá stavbou hierarchie obálek, která je založená na progresivně zjemňovaných řezech v již existující pomocné hierarchii obálek. Čtvrtá metoda se zabývá rychlou stavbou hierarchie obálek na grafických procesorech pomocí lokálně uspořádaného shlukování. Poslední metoda se zabývá paralelní optimalizací hierarchie obálek, která je založená na iterativním vyjímání a vkládání uzlů. Každá z těchto metod určitým způsobem vylepšuje současný stav vědní problematiky zabývající se algoritmem sledování paprsku.

Contents

1 Introduction

Computing high-quality images indistinguishable from the real-world photographs is a long-standing goal of computer graphics (see Figure 1.1). Contemporary image synthesis algorithms are based on solving the rendering equation [41] describing the light transport in a virtual scene by means of radiometric quantities. This model is based on assumptions of geometric optics; i.e. light travels instantaneously through the medium in straight lines, and light is not influenced by external factors such as gravity or magnetic field [22]. The rendering equation is the Fredholm integral equation of the second kind, for which the closed-form solution generally does not exist.

Figure 1.1: *An example of photorealistic image synthesis (courtesy of Davidovič et al. [20].)*

Veach [69] formulated the problem of global illumination as an integration over the space of light paths, where each light path is a sequence of vertices on scene surfaces such that neighboring light path vertices are mutually visible (see Figure 1.2). Interesting light paths are those with end vertices on a camera and a light source since we want to estimate the contribution of light coming from the light sources to the camera. The rendering equation is usually solved by path space sampling and Monte Carlo integration, which uses the probabilistic estimate that converges to correct values. However, it suffers from high-frequency noise, which disappears with increasing number of samples (see Figure 1.3). To obtain a plausible result, it is necessary to sample billions of light paths.

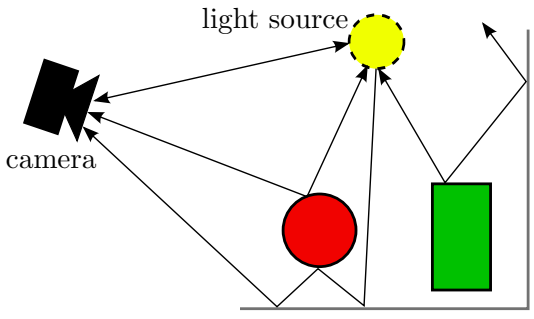


Figure 1.2: *An illustration of various light paths.*

Ray tracing [5, 86] is an underlying engine of the image synthesis algorithms, which is used

to find successive vertices of light paths. An elementary operation of ray tracing is to find the nearest intersection with a scene for a given ray. We start by sampling the first vertex of a light path on a camera or a light source, from which we cast a ray into the scene. We find the nearest intersection with the ray. We sample reflected or refracted ray originating from the intersection point, for which we find the nearest intersection and so on incrementally constructing the light path.

The major obstacle of ray tracing is its time complexity which is linear in the number of scene primitives for a naïve approach. The time complexity can be reduced by arranging scene primitives into an acceleration data structure exploiting spatial coherence to prune the search for the nearest intersection efficiently. The motivation for using acceleration data structures is to speed up ray tracing itself. The problem is that the construction of the acceleration data structure may take a significant amount of time. Therefore, the goal is to minimize the total time including both time for the construction of the data structure and ray tracing itself. Generally, we can say that the acceleration of the data structure increases with time spent on the construction to a certain extent. Depending on the application, we must balance both these criteria.

This thesis addresses various optimizations of the bounding volume hierarchy (BVH), which is a well-known concept with applications in ray tracing [63, 85], visibility culling [17, 11], or collision detection [23]. Nowadays, the BVH is one of the most popular data structures for ray tracing thanks to the predictable memory footprint, fast construction, excellent ray tracing performance, and easy updates for dynamic scenes. We can estimate the efficiency of a particular BVH through a cost function based on surface areas of bounding volumes [31, 49]. The problem of constructing an optimal BVH is believed to be NP-hard due to the combinatorial explosion [43]. In practice, we use various heuristics providing better or worse local minima without any guarantees [31, 76, 84]. In this thesis, we study this problem more deeply in order to propose new more efficient algorithms.

1.1 Goals

Traditionally, researchers studied ray tracing in the context of the offline rendering (e.g. movie industry), where rendering a single image takes hours or even days. In last years, ray tracing is becoming more favorable also in the context of the online rendering (e.g. video games), where each frame must be rendered in a fraction of a second. With the development of parallel architectures such as multi-core CPUs and many-core GPUs, we are able to achieve interactive framerates even on commodity hardware such as personal computers or mobile devices. Currently, there are two major industrial ray tracing engines: nVidia OptiX [60] and Intel Embree [83]. Recently, nVidia announced the RTX technology driven by the next-generation GPU Volta architecture. The RTX technology enables to call ray tracing routines through Vulkan and Microsoft DirectX. Simultaneously, AMD released Radeon-Rays, which is a cross-platform GPU-based ray tracing engine. In future, we can expect that the vendors of graphics cards will hardwire the ray tracing routines directly in the hardware.

The ultimate goal is to synthesize photorealistic images in real-time framerates. Even with the recent advances both in software and hardware, we are still far away from this ultimate goal. The problem is that the physically-based image synthesis is still very time demanding. Monte Carlo integration progressively refines the final image; however, if there is not enough time, the image contains disturbing high-frequency noise (see Figure 1.3). The goal is to advance the image synthesis closer to real-time framerates by accelerating ray tracing while using BVHs. In

Figure 1.3: *An example of Monte Carlo estimate (courtesy of Kelemen et al. [44]): unconverged result with high-frequency noise (left) and converged result (right).*

particular, this thesis aims at the following three goals that we can commonly encounter in the both offline and online rendering.

G.1 Efficient intersection query We endeavor to optimize a BVH in order to provide the ray intersection query as fast as possible even at the cost of higher construction times. In the offline rendering, a large number of rays must be traced to produce a plausible result, and thus the construction time is rather marginal since the BVH is reused many times. In the online rendering, a typical scenario is that there is a large static background with a few smaller dynamic objects. If the static geometry is known a priori, we can precompute high-quality BVH and reuse it in many frames even with fewer rays.

G.2 Efficient construction We need fast BVH construction algorithms either for static geometry that is not known a priori; or for dynamic geometry which changes in every frame invalidating the bounding volumes. We can refit bounding volumes preserving the topology of the BVH. However, if the changes are significant, the BVH will become completely inefficient. In such case, it pays off to reconstruct it from scratch.

G.3 Temporal coherence Besides the spatial coherence, we can also exploit temporal coherence, i.e. using similarities between consecutive frames of animation. However, it might be difficult to distinguish which results are still valid. We need efficient updating methods for the BVH exploiting the temporal coherence as the reconstruction from scratch might be too wasteful since it does not use any information from previous frames.

2 State-of-the-Art

This chapter summarizes the state-of-the-art in ray tracing with focus on the bounding volume hierarchy. In Section 2.1, we review theoretical background of ray tracing. In Section 2.2, we survey work related to the bounding volume hierarchy.

2.1 Preliminaries

In this section, we formulate the problem of visibility and ray tracing, discuss its time complexity, and present the concept of acceleration data structures.

2.1.1 Visibility and Ray Tracing

Visibility is a fundamental problem of computer graphics. Visibility can be defined in terms of mathematical relations. Visibility for two points is binary relation V on $\mathbb{E}^d$, where $\mathbb{E}^d$ denotes the d-dimensional Euclidean space. Two points $\mathbf{x}$ and $\mathbf{y}$ in d-dimensional Euclidean space are mutually visible if and only if line segment $\overline{\mathbf{xy}}$ with endpoints $\mathbf{x}$ and $\mathbf{y}$ does not intersect any scene primitive [36]. The relation is reflexive and symmetric. Using this definition, we simply define visibility function v:

$$v(\mathbf{x},\mathbf{y}) = \begin{cases} 1 \text{ if } (\mathbf{x},\mathbf{y}) \in V, \\ 0 \text{ otherwise.} \end{cases} \tag{2.1}$$

The ray tracing problem is similar to the visibility problem. For a given ray, we want to find the nearest scene primitive intersecting the ray. Ray $\mathbf{r} = [\mathbf{o}, \mathbf{d}]$ is a semi-infinite line specified by origin $\mathbf{o} \in \mathbb{E}^d$ and direction $\mathbf{d} \in \mathbb{E}^d$. Generally, it is not assumed that the direction is normalized ($\|\mathbf{d}\| = 1$); however, it is a usual convention. The parametric form expresses ray $\mathbf{r}$ as a function of scalar value t [61]:

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d},\ 0 \leq t < \infty,\ t \in \mathbb{R}. \tag{2.2}$$

The parametric form is useful for deriving various ray-primitive intersection algorithms, e.g. the ray-triangle intersection algorithm proposed by Möller and Trumbore [54]. The scene primitive is a compact subspace in $\mathbb{E}^d$ with $(d-1)$-dimensional continuous boundary (e.g. polygons or implicit surfaces in $\mathbb{E}^3$). Using the previous definitions, we define the ray tracing problem. Formally, for ray $\mathbf{r} = [\mathbf{o}, \mathbf{d}]$ and set of scene primitives $\mathcal{P}$, we want to find primitive $\mathbf{p} \in \mathcal{P}$ such that $\mathbf{p} \cap \mathbf{r}(t) \neq \emptyset$ and t is a positive minimum. A primitive intersected by the ray may not exist. A ray tracing algorithm is an algorithm providing a solution to the ray tracing problem. The visibility problem is trivially reducible to the ray tracing problem (see Figure 2.1).

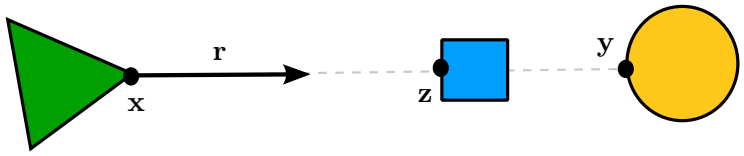


Figure 2.1: *A scheme illustrating the visibility problem and the ray tracing problem. Points* $\mathbf{x}$ *and* $\mathbf{y}$ *are not mutually visible. Points* $\mathbf{x}$ *and* $\mathbf{z}$ *are mutually visible. The solution of the ray tracing problem for ray* $\mathbf{r}$ *is point* $\mathbf{z}$ *on the blue rectangle.*

2.1.2 Algorithmic Complexity

A naïve ray tracing algorithm sequentially tests all scene primitives against a given ray. Thus, the time complexity of the naïve algorithm is $\mathcal{O}(n)$, where n is the number of scene primitives. In practice, it is necessary to test millions of rays against millions of scene primitives, and thus the naïve approach becomes practically inapplicable. Whitted noted in his seminal work that up to 95% of the execution time is spent on ray-primitive intersection computations [86], which motivated researchers to reduce the amount of these intersection computations.

Several theoretical facts were proven by Szirmay-Kalos and Márton [65]. The worst-case time complexity of the ray tracing problem itself is $\Omega(\log n)$. However, authors also proved that solving the ray tracing problem in $\mathcal{O}(\log n)$ requires $\mathcal{O}(n^4)$ space complexity, which is prohibitive for any practical use in computer graphics.

2.1.3 Acceleration Data Structures

In practice, we use various heuristics based on acceleration data structures exploiting spatial coherency of scene primitives. During the intersection computation, only spatially coherent parts are tested with the ray, others are pruned. This approach does not guarantee $\mathcal{O}(\log n)$ worst-case time complexity but it requires only $\mathcal{O}(n)$ space complexity, and it is faster by orders of magnitude than the naïve algorithm. The scene can be split in the spatial domain or object domain. Thus, acceleration data structures consist of two major classes: object partitioning structures (bounding volumes [17, 63], bounding volume hierarchy [63, 85]) and spatial subdivision structures (uniform grid [27], kD-tree [8, 36], octree [30]). Nowadays, the bounding volume hierarchy is one of the most popular acceleration data structures for ray tracing. The bounding volume hierarchy is a core component of the industrial ray tracing engines such as nVidia OptiX [60] and Intel Embree [83], and it is also the most addressed acceleration data structure in research projects.

2.2 Bounding Volume Hierarchy

The *bounding volume hierarchy* (BVH) was introduced already in the 80s by Rubin and Whitted [63]. The concept of the BVH itself is very simple. In the terminology of the graph theory, the BVH is a rooted tree with arbitrary branching factor with references to scene primitives in leaves and bounding volumes in interior nodes. The bounding volumes tightly enclose scene primitives in the corresponding subtrees.

In the context of ray tracing, we used traditionally binary BVHs. Recently, with development of modern multi-core CPUs and many-core GPUs, also *wide-BVHs*, i.e. BVHs with branching factors 4 or 8, became popular [18, 78, 25, 66, 34, 88]. The choice of bounding volume type is a compromise between tightness of the bounding volume and the complexity of the intersection test (see Figure 2.2). In the context of ray tracing, we use almost exclusively axis-aligned bounding boxes which are further referenced simply as bounding boxes.

2.2.1 Cost Model

We can estimate the quality of a particular BVH in terms of the expected number of operations needed for finding the nearest intersection with a random ray. The cost of a BVH with root N is given by the recurrence equation:

$$c(N) = \begin{cases} c_T + \sum_{N_c} P(N_c|N)c(N_c) & \text{if } N \text{ is interior node,} \\ c_I|N| & \text{otherwise,} \end{cases} \tag{2.3}$$

where $c(N)$ is the cost of a subtree with root N, N_c is a child of node N, $P(N_c|N)$ is the conditional probability of traversing node N_c when node N is hit, and $|N|$ is the number of scene primitives in a subtree with root N. Constants c_T and c_I express average cost of the traversal step and ray-primitive intersection respectively. Using the *surface area heuristic* (SAH) [31, 49],

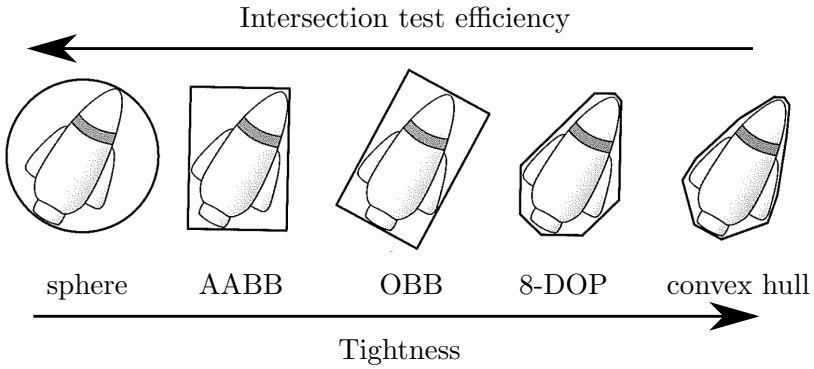


Figure 2.2: *An example of various types of bounding volumes (courtesy of Ericson [23]): sphere, axis-aligned bounding box (AABB), oriented bounding box (OBB), discrete oriented polytop (DOP), and convex hull.*

we can express the conditional probabilities as geometric probabilities, i.e. the ratio of the surface area of a child node and the parent node:

$$P(N_c|N)^{SAH} = \frac{SA(N_c)}{SA(N)}, \tag{2.4}$$

where $SA(N)$ is the surface area of the bounding box of node N. By substituting Equation 2.4 into Equation 2.3, we get the following expression:

$$c(N)^{SAH} = \begin{cases} c_T + \sum_{N_c} \frac{SA(N_c)}{SA(N)} c(N_c) & \text{if } N \text{ is interior node,} \\ c_I|N| & \text{otherwise.} \end{cases} \tag{2.5}$$

By unrolling, we get rid of the recurrence:

$$c(N)^{SAH} = \frac{1}{SA(N)} \left[c_T \sum_{N_i} SA(N_i) + c_I \sum_{N_l} SA(N_l)|N_l| \right], \tag{2.6}$$

where N_i and N_l denote interior and leaf nodes of a subtree with root N respectively. The problem of finding an optimal BVH is believed to be NP-hard [43].

The SAH makes following assumptions [26]:

- Ray origins are uniformly distributed outside the scene bounding box.
- Ray directions are uniformly distributed.
- Rays are not occluded.

These assumptions are quite unrealistic, and thus several corrections were proposed. Fabianowski et al. [26] proposed a modification for handling rays with origins inside the scene bounding box. The modification is reasonable as many rays originate on scene primitives:

$$P(N_c|N)^{Inside} = \frac{V(N_c)}{V(N)} + \frac{1}{V(N)} \int_{N \setminus N_c} \frac{\alpha_{\mathbf{x}}}{4\pi} d\mathbf{x}, \tag{2.7}$$

where $V(N)$ is the volume of the bounding box of node N, and $\alpha_{\mathbf{x}}$ is a solid angle obtained by projecting the bounding box onto the unit sphere around $\mathbf{x}$ (see Figure 2.3). However, this expression is much harder to evaluate than the simple ratio of surface areas. Since there is no closed-form solution of the integral, authors proposed a numerical approximation.

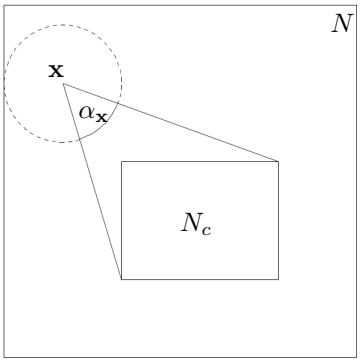


Figure 2.3: *An illustration of the cost correction taking into account rays originating inside the bounding box (courtesy of Fabianowski et al. [26]).*

Bittner and Havran [14] proposed the *ray distribution heuristics* (RDH), which is a method taking into account a given ray distribution. The authors proposed to sample rays of the ray distribution, and then use directly the ratio of the number of ray hits instead of surface areas:

$$P(N_c|N)^{RDH} = \frac{R(N_c)}{R(N)}, \tag{2.8}$$

where $R(N)$ is the number of rays hitting the bounding box of node N (see Figure 2.4). Similarly, Vinkler et al. [75] proposed the *occlusion heuristic* (OH) using the ratio of the number of visible scene primitives:

$$P(N_c|N)^{OH} = \frac{O(N_c)}{O(N)}, \tag{2.9}$$

where $O(N)$ is the number of visible scene primitives in a subtree with root N.

It is difficult to evaluate both of these heuristics as they require a ray set representing the ray distribution. Using these probabilities directly may lead to unstable results due to either undersampling or oversampling the ray distribution. Hence, the authors proposed to blend these probabilities with the geometric probabilities given by the SAH to make the results more robust:

$$\overline{P}^{RDH}(N_c|N) = wP(N_c|N)^{RDH} + (1-w)P(N_c|N)^{SAH}, \tag{2.10}$$

$$\overline{P}^{OH}(N_c|N) = wP(N_c|N)^{OH} + (1-w)P(N_c|N)^{SAH}, \tag{2.11}$$

where w is a blending parameter in range $[0, 1]$.

2.2.2 Top-Down Construction

The most common way how to construct a BVH is a recursive division of scene primitives. At the beginning, we have a root node containing the scene bounding box with all scene primitives. In each step of the construction, one node is processed. A node is split into two new nodes, which are further processed. Scene primitives of the split node are divided into its children and

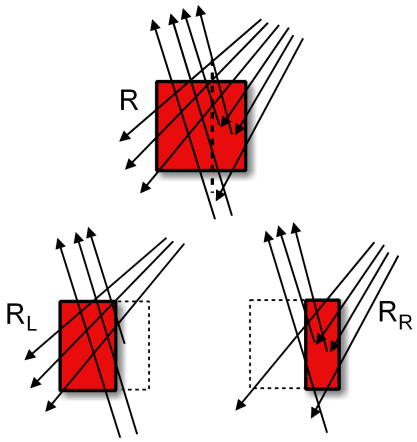


Figure 2.4: *An illustration of the RDH method (courtesy of Bittner and Havran [14]).*

bounding boxes of the children are computed. The division continues until at least one of the termination criteria is satisfied. The common termination criteria are maximum primitives in a node, maximum tree depth, or maximum memory used.

For each node, there are $\mathcal{O}(2^n)$ ways how to divide scene primitives into its children, where n is the number of scene primitives in the node. Popov et al. [62] showed that the number of partitioning of scene primitives in the node for axis-aligned bounding boxes is $\mathcal{O}(n^6)$, which is still prohibitive for any practical use. Hence, we use heuristics dividing scene primitives by axis-aligned planes. There are three basic approaches how to split the node: object median split, spatial median split, and split based on the cost model. Since each scene primitive must be only on one side of the plane even if it overlaps, scene primitives are approximated by points (e.g. centroids of the bounding boxes) [76]. All three splitting axes are used, or one axis is chosen according to some heuristic. The greatest extent of the bounding box or round-robin are the most common ones.

During splitting, we cannot use the cost model directly because we do not know the cost of children. Thus, we approximate the cost by treating children as leaves:

$$c(N) \approx c_T + c_I \sum_{N_c} \frac{SA(N_c)}{SA(N)} |N_c|. \tag{2.12}$$

This is an upper bound of the cost [2]. There are two approaches how to evaluate the cost approximation:

Sweeping: There are $|N|-1$ possible splitting planes induced by $|N|$ scene primitives for each axis. This approach evaluates all of them and chooses the one with the lowest cost approximation. To guarantee time complexity $\mathcal{O}(n \log n)$, we keep three arrays with primitive indices sorted according to all axes. To prevent the sorting in each step, we must reorder not only indices of the splitting axis but also indices of the other two axes [81].

Binning: Evaluating all possible splitting planes might be costly. Thus, several researchers proposed an evaluation known as *binning* [37, 76]. The splitting extent is uniformly divided into k equally-spaced *bins*. Scene primitives are then projected into these bins. Then, only $k-1$ splitting planes between bins are evaluated using two prefix scan passes on the bins. All scene primitives may fall into the same bin, then the object median split must be performed.

Unlike the sweeping algorithm, this approach can be easily parallelized on CPUs using SIMD instructions and multithreading [76].

2.2.3 Bottom-Up Construction

On the other hand, we can proceed bottom-up using agglomerative clustering. This approach was introduced by Walter et al. [84]. At the beginning, all scene primitives are treated as clusters. In each step, the nearest cluster pair based on a distance function is merged. The distance function is defined as a surface area of the bounding box tightly enclosing both clusters. We repeat merging until only one cluster remains. This approach generally produces BVHs of higher quality than the traditional top-down approach. The disadvantage is the time complexity, which is $\mathcal{O}(n^3)$ for naïve approach. To accelerate the nearest neighbors search, the authors proposed to use an auxiliary kD-tree and heap.

2.2.4 Incremental Construction

The incremental construction by insertion was originally proposed by Goldsmith and Salmon [31]. Scene primitives are piece-by-piece inserted into the BVH based on the cost model. The advantage of the incremental construction is that we do not need to know the whole input at the beginning of the construction. Traditionally, it was believed that the BVHs constructed by insertion suffer from lack of quality [36]. Recently, Bittner et al. [13] proposed an incremental construction algorithm producing high-quality BVHs using the insertion-based optimization [12]. The authors perform a batch of insertion operations, and then perform global updates as in the original algorithm. They demonstrated the usage in a proof-of-concept application streaming the data over the network.

2.2.5 Parallel Construction

Researchers were looking for the inspiration in traditional sequential methods to design parallel algorithms. The problem is the parent-child dependency. In top-down construction, we cannot build children until we know the parent. In bottom-up construction, we cannot build the parent until we know its children. In other words, we do not have enough independent work to utilize parallel processors, especially at the beginning. Therefore, many parallel algorithms are based on sorting scene primitives along the *Morton curve* to partition scene primitives into coherent clusters which can be processed independently by parallel algorithms.

Morton Curve The Morton curve is a well-known space-filling curve [55]. The order along the curve is given by $3k$-bit Morton codes. The space filled by the curve is subdivided into a grid of $2^k \times 2^k \times 2^k$ space elements. Each $3k$-bit Morton code corresponds to one space element in the grid. The advantage is that the mapping between the Morton codes and coordinates of space elements is very simple. A $3k$-bit Morton code can be computed by interleaving successive bits of the corresponding space element coordinates. The following two equations define the relation between coordinates of a space element and its Morton code respectively:

$$\mathbf{x} = [x, y, z] = [x_k \dots x_1, y_k \dots y_1, z_k \dots z_1], \tag{2.13}$$

$$m = z_k y_k x_k \dots z_1 y_1 x_1. \tag{2.14}$$

Space element $\mathbf{x}$ corresponds to $3k$-bit Morton code m, where z_i, y_i, x_i are i-th bits of coordinates z, y, x respectively. The Morton curve can be generalized into an arbitrary dimension. An example of the Morton Curve in 2D is depicted in Figure 2.5.

The Morton curve not only coherently fills the space but also implicitly encodes the BVH constructed by spatial median splits [42], which can be used directly or as an auxiliary data structure. Segments of the Morton curve are coherent clusters corresponding to nodes in the BVH, which can be processed independently. That is exactly what parallel algorithms need. Sorting along the Morton curve is convenient as general libraries provide efficient implementations of parallel sorting algorithms. Scene primitives are usually approximated by the centroid of the bounding box, from which the corresponding Morton code is generated. Vinkler et al. [70] extended the Morton codes by encoding the size of scene primitives, applying adaptive ordering of the code bits, and using a variable number of bits for different dimensions.

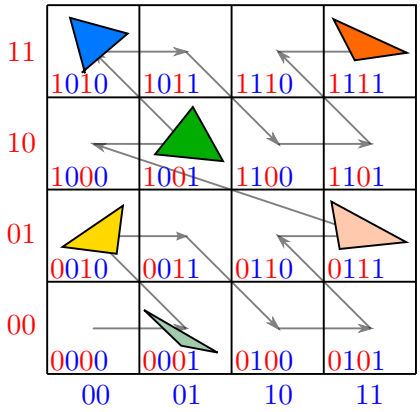


Figure 2.5: *An example of the Morton curve and 4-bit Morton codes in 2D.*

Multi-core CPUs Wald [76] proposed *horizontal* and *vertical* parallelization of the top-down construction method using the binning algorithm. The horizontal parallelization is used for the upper levels where only a few interior nodes contain lots of scene primitives. Scene primitives are equally divided between threads. Each thread projects its scene primitives into its private set of bins. After binning, the bin sets are merged, and the best splitting plane is selected. Once, the number of subtrees is equal to the number of threads, the algorithm switches to vertical parallelization, where each subtree is processed by a single thread. The algorithm is designed to utilize both SIMD instructions and multithreading, and it was also extended for the MIC architecture [77].

Gu et al. [33] proposed a parallel algorithm using approximate agglomerative clustering (AAC). The idea is to restrict the search space for the nearest neighbor by proximity given by the spatial median splits. At the beginning, each scene primitive is considered to be a cluster. The algorithm recursively splits scene primitives using the spatial median splits based on Morton codes until subtrees contain less than δ clusters, where δ is a parameter of the method. To reduce the number of clusters in such subtrees, it merges the clusters using agglomerative clustering until only $f(\delta)$ clusters remains, where cluster reduction function f is another parameter of the method. Then, it continues to the parent, where remaining clusters and clusters of its sibling are put together. Again, it merges them using the same procedure. This continues until the

root cluster is merged. The algorithm is very fast, and the constructed BVHs are roughly of the same quality as BVHs produced by full agglomerative clustering. The Bonsai algorithm proposed by Ganestam et al. [28] is based on a similar principle using the sweeping algorithm instead of agglomerative clustering. In Section 3.3, we propose a novel parallel construction algorithm based on the progressively refined cut of an existing BVH.

Many-core GPUs Lauterbach et al. [48] proposed an algorithm known as LBVH[1] using the concept of the Morton codes. The BVH is constructed level-by-level using one kernel launch for each level. Karras [42] improved the LBVH method by using the concept of radix trees building the topology of a BVH in a single kernel launch. However, the method requires an additional bottom-up pass to refit the bounding boxes. This issue was recently addressed by Apetrei [4] who proposed to build the topology in a bottom-up manner simultaneously refitting the bounding boxes in a single kernel launch. The LBVH method is the fastest construction algorithm up to date but the resulting BVHs suffer from lack of quality as the method employs only spatial median splits. Pantaleoni and Luebke [59], and Garanzha et al. [29] extended the LBVH method into a method known as HLBVH[2] using the binning algorithm for top levels of the hierarchy. Karras and Aila [43] proposed an optimization algorithm based on treelet restructuring (TRBVH). The algorithm rebuilds treelets of fixed size in parallel using the algorithm based on dynamic programming. Domingues and Pedrini [21] showed that BVHs of similar quality can be produced in shorter times using agglomerative clustering instead of dynamic programming (ATRBVH). Vinkler et al. [74] introduced a concept of on-demand BVH construction running entirely on the GPU using a versatile GPU framework based on a task pool with persistent warps [71] and an efficient GPU allocator [72]. In Chapter 3, we propose two GPU-based algorithm. In Section 3.2, we propose an algorithm based on k-means clustering. In Section 3.4, we propose an algorithm based on locally-ordered clustering inspired by work of Walter et al. [84].

2.2.6 Optimization

The problem of both top-down and bottom-up approaches is that we are not able to compute the cost because the BVH is not complete. Thus, researchers proposed to build an initial BVH and then optimize it in order to reduce the global cost. Kensler [45] proposed to use tree rotations inspired by a technique from the binary search trees (see Figure 2.6). A different approach was proposed by Bittner et. al [12]. The authors proposed to remove a subtree which causing the cost overhead and then reinsert it to a new position decreasing the global cost (see Figure 2.7). This algorithm is inherently sequential. In Section 3.5, we propose a parallel version of this algorithm which is up to two orders of magnitude faster than the sequential version.

2.2.7 Collapsing Subtrees

The problem of the bottom-up construction is that the resulting BVH contains exactly one scene primitive per leaf, which may cause the memory overhead. Furthermore, it might also pay off to collapse some subtrees into leaves in order to reduce the cost. We proceed from leaves to the root. If the cost of a subtree as a leaf is lower than the cost of the subtree as it is, we collapse it. Generally, we can apply this postprocess to any BVH constructed by an arbitrary method to further improve its quality [12, 43].

[1] Linear Bounding Volume Hierarchy
[2] Hierarchical Linear Bounding Volume Hierarchy

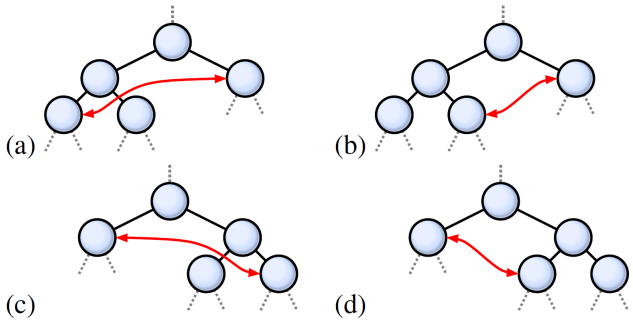


Figure 2.6: *An illustration of four possible tree rotations (courtesy of Kensler [45]).*

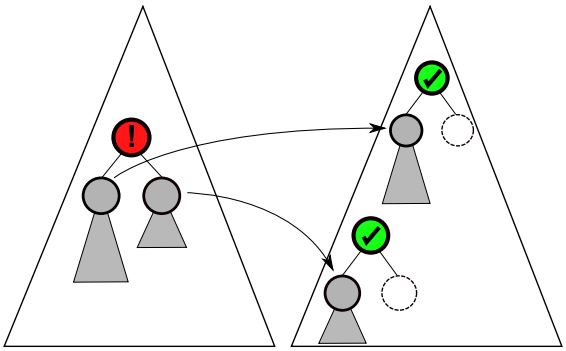


Figure 2.7: *An illustration of the insertion-based optimization (courtesy of Bittner et al. [12]).*

2.2.8 Spatial Splits

The BVH often adapts poorly to scenes containing scene primitives with non-uniform sizes and overlapping bounding boxes, which are impossible to separate by definition. The spatial subdivision structures, which split the space into disjoint cells, excel in such scenes. For example, the kD-tree [8, 36] adapts very well to the geometry with non-uniform sizes; however, both the construction and the traversal are more complicated [73]. Thus, researchers endeavor to fuse advantages of both the kD-tree and BVH. The idea is to relax the restriction that each scene primitive is referenced only once. Using this relaxation, the BVH adapts much better to scenes containing scene primitives with non-uniform sizes. The traversal and the structure itself remain the same. The only problematic part might be the refitting procedure as the leaf nodes are induced by spatial splits and not by scene primitives. Thus, if the geometry changes as it happens in dynamic scenes, we are not able to update bounding boxes while preserving spatial splits.

Two strategies were introduced. The first strategy proposed by Ernst and Greiner [24], Dammertz and Keller [19], and Karras and Aila [43] is to cover large scene primitives by multiple tighter bounding boxes before the construction, which are directly fed into construction algorithms. Note that scene primitives themselves are not split (see Figure 2.8). This approach is simpler and can be easily parallelized [43].

The second strategy proposed by Stich et al. [64] is to allow spatial splits during the top-down construction, which is very similar to the kD-tree construction [37, 81]. However, unlike the piecewise linear function in the case of kD-trees [64], the cost function is piecewise quadratic since the bounding box adapts in all three dimensions. Thus, the authors proposed to use a

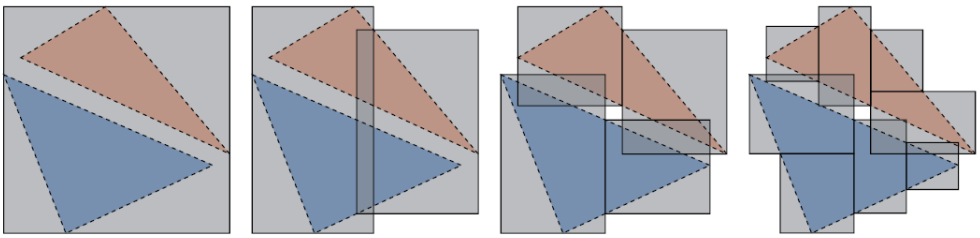


Figure 2.8: *An illustration of the BVH with spatial splits (courtesy of Ernst and Greiner [24]).*

modified version of the binning algorithm. This strategy leads to higher ray tracing performance as it takes into account also the proximity of scene primitives.

2.2.9 Dynamic Scenes

In last two decades, ray tracing has become attractive also for interactive or real-time applications (e.g. computer games), which used to be traditionally based on rasterization. The problem arises when the scene contains dynamic geometry. In each frame, the geometry changes and also the corresponding BVH becomes invalid as the bounding boxes do not enclose the actual geometry (see Figure 2.9). There are two main approaches how to update the BVH. We can refit the bounding boxes, or rebuild the BVH from scratch. Refitting is simpler and faster; however, the BVH may degenerate in time if the changes are significant. Ize et al. [40] and Wald et al. [82] proposed to use refitting while asynchronously rebuilding the BVH from scratch. Yoon et al. [89] and Kopta et al. [46] proposed to optimize the BVH using tree rotations to prevent the bounding boxes degeneration.

Figure 2.9: *An example of a scene with dynamic geometry.*

A typical scenario is that the scene consists of a static background and several dynamic objects. Wald et al. [79] proposed the concept of *two-level BVHs*. A top BVH is built over the individual objects and each object has its BVH. More precisely, the top BVH contains references to individual objects with local transformations which support rigid body animations and instancing. A caveat of the two-level BVH is that it might cause a traversal overhead if the individual objects overlap. This issue was recently addressed by Benthin et al. [7].

2.2.10 Ray Traversal

We also review the state-of-the-art in ray traversal, which is a no less important problem, even though that methods proposed in Chapter 3 are oblivious to a particular traversal algorithm. The basic algorithm is based on a stack. We put the root onto the stack, and then we enter the traversal loop. We pop up a node from the top of the stack and test the ray against the bounding box of the node. If the ray hits the bounding box, we either put both children onto

the stack if the node is interior, or we test the ray against scene primitives in that node if it is a leaf, and eventually update the nearest intersection. We continue until the stack is empty. The traversal scheme is quite simple, and it is the core of parallel traversal algorithms. However, both multi-core CPUs and many-core GPUs have their specific issues. The parallel traversal is simpler than the parallel construction as rays are simply distributed between threads and each thread processes its rays sequentially.

Multi-core CPUs Modern CPUs have the wide vector units. There are two approaches how to use them for the traversal: (1) Using wide-BVHs, a single ray is tested against multiple bounding boxes or scene primitives [78], which might be useful for incoherent rays. However, the construction of wide-BVHs is more complicated. (2) A *ray packet*, i.e. multiple rays are tested against a single bounding box or scene primitive with the packet size equal to the width of vector units (4, 8, or 16) [16]. Ray packets can be further aggregated into larger *ray streams* containing up to thousands of rays [80, 58]. Using ray packets or ray streams is very efficient for *coherent rays*, i.e. rays with similar origins and directions (e.g. primary rays), since they visit the same nodes in the BVH. However, using ray packets or ray streams for incoherent rays might lead to sequential processing in the extreme case. Some rays in a packet or stream may become inactive which leads to SIMD underutilization. Therefore, several reordering and filtering methods were proposed to increase coherency [50, 32, 9, 6].

Many-core GPUs Threads on the GPU are scheduled and executed in *warps*, i.e. groups of 32 threads executing the same code [56]. The problem is *warp divergence*, i.e. different threads in a warp execute different code, where different branches are executed sequentially. The GPU traversal is mostly limited by the memory bandwidth and warp divergence. Aila and Laine [3] proposed an efficient stack-based traversal algorithm identifying two factors causing the warp underutilization. The first factor is that the traversal loop consists of two phases: testing a ray against bounding box and testing the ray against scene primitives in leaves. Switching between these two phases causes the warp divergence. The authors proposed to postpone this switching by storing already found leaves into a buffer showing that even a buffer with a single entry is sufficient. The second factor is that already terminated threads in a warp are inactive. Therefore, authors proposed to use *dynamic fetch*; i.e. if only a few threads are active, the inactive threads fetch new rays. The stack on the GPU is stored in the local memory, so researchers designed various stack-less traversal algorithms [47, 35, 1, 10]. Guthe [34] proposed to use 4-ary BVHs to reduce memory latency. Recently, Ylitie et al. [88] proposed to use compressed 8-ary BVHs achieving the highest performance up to date.

3 Overview of Contributions

This chapter summarizes the main contributions of this thesis. The thesis is submitted as a compilation of five research papers corresponding to the following five sections presented in the chronological order of their publications. The relations among the papers and the goals of the thesis are discussed in Chapter 4. In Section 3.1, we present a method for handling geometric changes in dynamic scenes. In Sections 3.2, 3.3, and 3.4, we present three high-performance BVH construction algorithms: two for many-core GPUs and one for multi-core CPUs. In Section 3.5, we present a parallel insertion-based BVH optimization targeting many-core GPUs.

3.1 T-SAH: Animation Optimized BVH

In dynamic scenes, the geometry changes in every frame invalidating the BVH, which must be updated. We can construct the BVH for one of the keyframes, and during the rendering refit the bounding boxes, which is very fast but bounding boxes might degenerate in time. We can also reconstruct the BVH from scratch in each frame, which in turn might be unnecessarily costly because it does not take into account any temporal coherence.

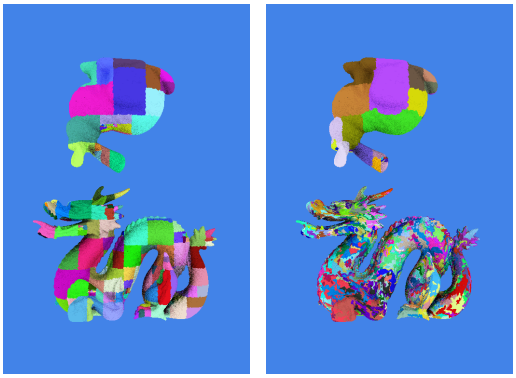


Figure 3.1: *A visualization of a BVH using the SAH (left) and the T-SAH (right). Note how the T-SAH automatically identifies groups of fragments which are coherent through the animation sequence.*

We propose an algorithm based on using a single BVH which is optimized for the whole animation sequence. During the rendering, we use simple refitting of bounding boxes. We extended the cost based on the SAH into the temporal domain (T-SAH) which is defined as a weighted arithmetic mean of costs of given animation frames:

$$c_t(N) = \frac{1}{SA_t(N)} \left[c_T \sum_{N_i} SA_t(N_i) + c_I \sum_{N_l} SA_t(N_l)|N_l| \right], \tag{3.1}$$

$$\widetilde{c}(N) = \frac{\sum_i w_i c_i(N)}{\sum_i w_i}, \tag{3.2}$$

where $SA_t(N)$ is the surface area of node N in time t and w_i is the weight of animation frame i. We plug the T-SAH cost model into the insertion-based optimization algorithm [12] to optimize the BVH for the animation sequence. Note that we use for the optimization only a few representative frames. Using all animation frames would be costly providing a BVH roughly with the same quality. A visualization of a BVH constructed by our method is shown in Figure 3.1. Using a single BVH reduces the per-frame overhead to refitting of bounding boxes. We show that this approach outperforms both the per-frame reconstruction from scratch using HLBVH [29] and the best-over-time BVH with refitting [80] (see Figure 3.2). The method is extremely easy to integrate to any application using BVHs. More details about the method and results can be found in the original paper [15].

3.2 Parallel BVH Construction using k-means

To design new algorithms for parallel architectures, we should not only look for the inspiration in the traditional construction approaches but also try to find completely new directions. We were inspired by the work of Walter et al. [84], who came up with an idea to use agglomerative

1.1× vs SAH+refit 1.5× vs HLBVH	5.4× vs SAH+refit 1.1× vs HLBVH	1.3× vs SAH+refit 2.6× vs HLBVH	1.4× vs SAH+refit 1.6× vs HLBVH

Figure 3.2: *Average speedups of ray tracing for several tested animations using our single T-SAH optimized BVH with refitting compared to the SAH with refitting and per-frame reconstruction using HLBVH.*

clustering for the BVH construction. In this work, we investigate the opposite approach employing divisible clustering in the context of the BVH construction, which is an alternative to the traditional top-down approach based on splitting by axis-aligned planes.

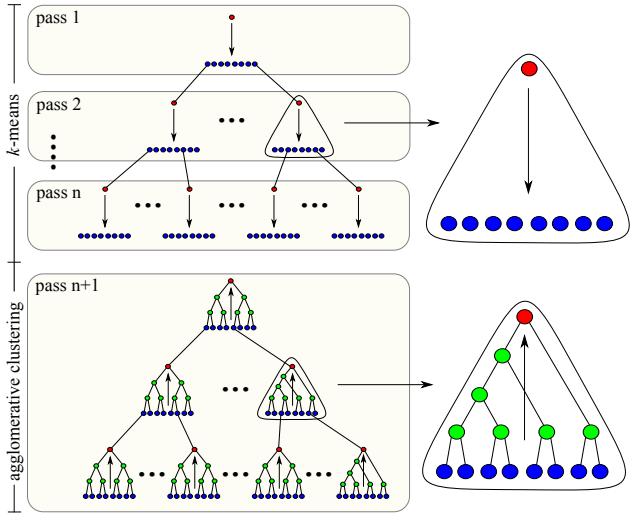


Figure 3.3: *An illustration of the BVH construction using k-means clustering. First, we construct a k-ary BVH by recursively applying k-means clustering. Then, we build intermediate levels by agglomerative clustering.*

We propose a parallel BVH construction algorithm combining both divisible and agglomerative clustering. First, we construct a k-ary BVH by iteratively applying k-means clustering. Second, we construct intermediate levels using agglomerative clustering (see Figure 3.3). A visualization of k-means clustering is shown in Figure 3.4. According to our knowledge, we are the first who applied k-means clustering in the context of the BVH construction. The results show that our algorithm achieves the time-to-image speedup up to 11% with respect to HLBVH [29]. However, it is slightly worse than ATRBVH [21], but still leads to better results in a few cases. Recently, wide-BVHs have become popular not only on CPUs [83] but also on GPUs [88], and thus we believe that our method will be further investigated in this context. More details about the method and results can be found in the original paper [51].

Figure 3.4: *A visualization of the results of k-means clustering ($k = 8$) for the first three passes of the algorithm using five k-means iterations. Triangles belonging to different clusters are shown in different colors.*

3.3 Progressive Hierarchical Refinement

Due to the limited memory and costly CPU-GPU data transfers, using many-core GPUs is not always the best choice, especially if we deal with a large amount of data. In that case, multi-core CPUs are preferable, for which we need efficient BVH construction algorithms.

We propose a parallel BVH construction algorithm using progressive hierarchical refinement (PHR). The algorithm was inspired by the idea of Hunt et al. [39], who used a scene graph structure to accelerate the kD-tree construction. The algorithm starts by constructing an auxiliary BVH using a fast construction algorithm such as LBVH [42]. Then, we find a *cut* in the auxiliary BVH, which is a set of nodes completely separating the root and leaves. The cut is formed by nodes whose surface area is below an adaptive threshold. To find the initial cut, we use a priority queue with a simple test on the surface area. We split the cut into two parts by the sweeping algorithm. We want to keep the sizes of both parts the same as the size of the initial cut. Thus, we refine the threshold taking into account the current depth and replace some nodes of the cut by their children based on the refined threshold (see Figure 3.5). We apply this procedure recursively to build the whole BVH.

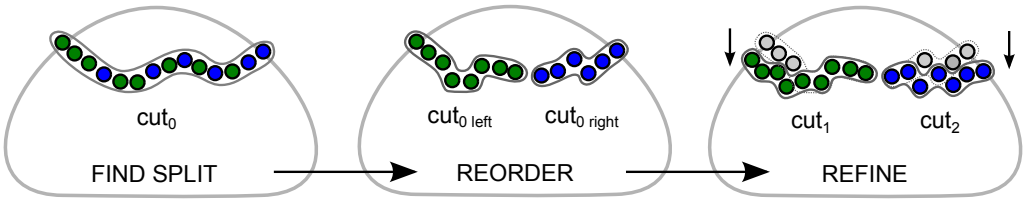
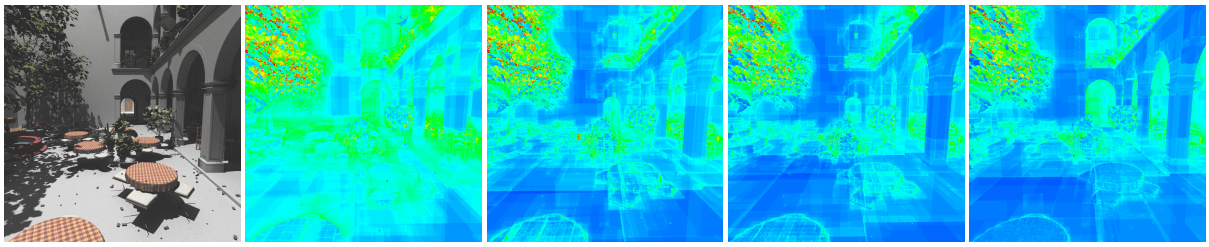


Figure 3.5: *An illustration of the hierarchical cut refinement. First, we form the initial cut based on the threshold (left). Then, we split the cut into two parts by the sweeping algorithm (middle). Finally, we refine the threshold and replace some of the nodes of the cut by their children (right).*

The results show that our algorithm achieves the time-to-image speedup up to 29% with respect to AAC [33] and up to 4% with respect to builders implemented in Intel Embree [83] (see Figure 3.6). In comparison with the strategy proposed by Hunt et al. [39] adapted to the BVH construction, our method achieves the speedup up to 20%. More details about the method and results can be found in the original paper [38].



Embree SAH 769 ms | Our PHR-Fast 472 ms | Our PHR-HQ 977 ms | Embree Fast-Spatial 1950 ms

Figure 3.6: *The San Miguel scene rendered in Embree path tracer [83]. Visualizations of the number of traversal steps for primary rays using BVHs from different builders (the red color corresponds to 100 traversals per ray). Our PHR-Fast method provides* 1.6× *lower build time than Embree SAH, while the PHR-HQ method has* 2× *lower build time than Embree Fast-Spatial.*

3.4 Parallel Locally-Ordered Clustering

Using a GPU is an excellent choice if the data fits into its memory with only a few CPU-GPU transfers. The GPU is a source of tremendous computational power; however, designing algorithms for GPUs might be quite challenging.

We propose a simple yet efficient BVH construction algorithm targeting contemporary GPUs based on parallel locally-ordered clustering (PLOC) inspired by work of Walter et al. [84]. We construct the BVH iteratively in bottom-up fashion merging a batch of cluster pairs in each iteration. The idea is to keep the clusters ordered along the Morton curve. This ordering allows us to identify approximate nearest neighbors very efficiently (see Figure 3.7). We use the distance function defined as a surface area of the bounding box tightly enclosing both clusters. This distance function obeys a non-decreasing property, which tells us that if the nearest neighbors of two clusters mutually correspond, then we know that no better neighbor will emerge in the future. Hence, we can merge all mutually corresponding cluster pairs independently in parallel. The method is GPU-friendly since it does not use any additional data structures such as distance matrix, and the number of atomic operations is minimized. The algorithm is extremely simple and outperforms the state-of-the-art GPU-based builders (see Figure 3.8). The algorithm achieves the time-to-image speedup up to 39% with respect to ATRBVH [21]. We also conducted a comparison with AAC [33] showing that our algorithm produces BVHs of a similar quality about four times faster. Source codes of the method are publicly available at the project site[1]. More details about the method and results can be found in the original paper [52].

3.5 Parallel Reinsertion for BVH Optimization

If we trace a large number of rays, it pays off to optimize a BVH as much as possible since the spent time will be amortized by reusing the BVH many times. The insertion-based optimization proposed by Bittner et al. [12] achieves the best results in terms of BVH quality up to date. However, it is inherently sequential. In the original algorithm, we iteratively select nodes and remove them. Then, we use branch-and-bound search to find the best position for the insertion using the priority queue.

[1] http://dcgi.felk.cvut.cz/projects/ploc

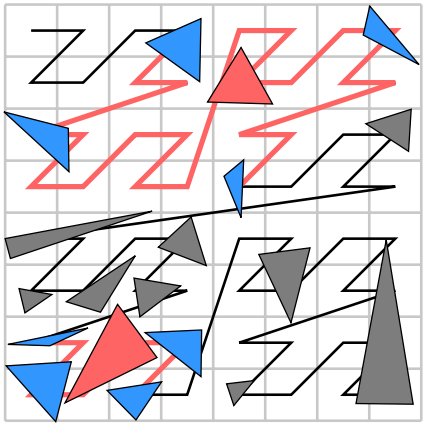


Figure 3.7: *An illustration of the nearest neighbor search for $r = 2$. Two clusters (red triangles) search for their nearest neighbors (blue triangles) in the neighborhood (red curve). Notice how the algorithm adapts to the density of clusters in the neighborhood.*

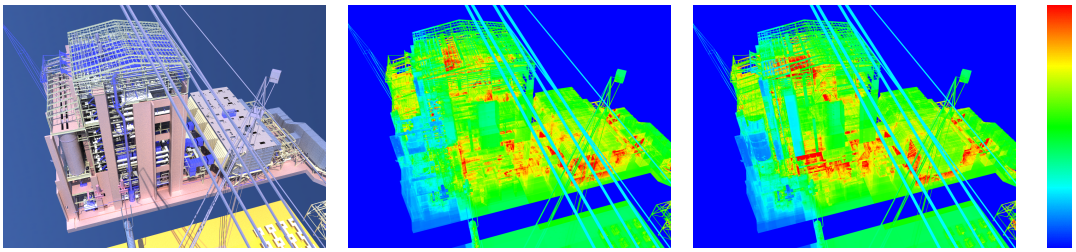


Figure 3.8: *Path tracing of the Powerplant scene (12.8M triangles) using a BVH constructed by our method (left). A visualization of the number of ray intersection operations for our method (middle) and the state-of-the-art ATRBVH [21] (right). The red color corresponds to 325 intersections (both bounding volume and triangle intersections are counted). In this case, our method achieves 32% reduction of build time (210 ms vs. 309 ms) and 17% speedup in the ray tracing performance (88 MRays/s vs. 75 MRays/s) with respect to ATRBVH.*

We propose a parallel insertion-based optimization of BVHs for contemporary many-core GPUs. The key observation is that we do not need to remove the node from the hierarchy to find the best position for the insertion. We start to search for the best position from the original position of the node, which serves as a lower bound for pruning. We successively visit sibling subtrees on the way up to the root using the pre-order traversal with parent links (see Figure 3.10). Using such traversal, all nodes can search for their best positions in parallel. The search procedure is GPU-friendly as we do not use any priority queue or another auxiliary data structure. Conflicts between nodes occur, so we propose a locking scheme based on atomic operations to resolve these conflicts by prioritizing nodes with higher cost reduction. An illustration of the algorithm is depicted in Figure 3.9. The method produces BVHs of higher quality than the state-of-the-art algorithms. In particular, our method achieves the ray tracing speedup up to 12% in comparison with ATRBVH [21] and up to 31% in comparison with PLOC [52]. The proposed algorithm is roughly about two orders of magnitude faster than the original sequential

algorithm. We released the source codes, which are available at the project site[2]. More details about the method and results can be found in the original paper [53].

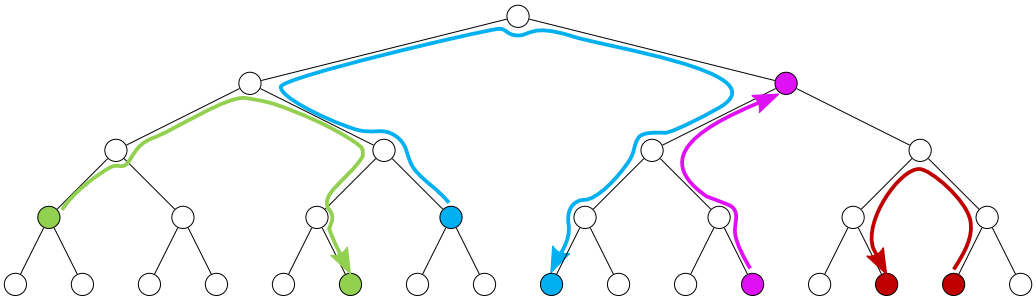


Figure 3.9: *An illustration of our parallel insertion-based BVH optimization. For each node (the input node), we search for the best position leading to the highest reduction of the cost for reinsertion (the output node) in parallel. The input and output nodes together with the corresponding path are highlighted with the same color. We move the nodes to the new positions in parallel while taking care of potential conflicts of these operations.*

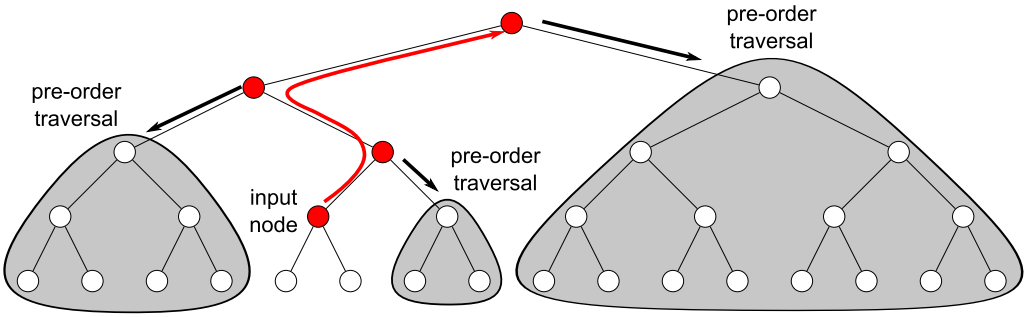


Figure 3.10: *An overview of the search for a new position. We proceed from a given input node up to the root visiting all sibling subtrees using the pre-order traversal with parent links.*

4 Conclusion and Future Work

This thesis presents five novel methods for high-performance ray tracing using BVHs. This chapter summarizes the proposed methods and discusses their possible future extensions.

4.1 Summary

The presented methods cover a wide spectrum of scenarios in the rendering. Here, we put them into a common context defined by the goals stated in Section 1.1.

G.1 Efficient intersection query In a wider perspective, all of the presented methods address this goal. In Section 3.1, we proposed a method for high-performance ray tracing in dynamic scenes. In Section 3.3 and 3.4, we proposed two BVH construction algorithms producing high-quality BVHs very quickly targeting multi-core CPUs and many-core GPUs respectively. However, we see the highest impact on this goal in the method presented in

[2] http://dcgi.felk.cvut.cz/projects/prbvh

Section 3.5. We proposed a parallel optimization algorithm achieving the highest possible BVH quality among the state-of-the-art builders at the cost of slightly higher construction times. Using our method is the best choice for the offline rendering or static content in the online rendering provided the static geometry is known a priori.

G.2 Efficient construction We presented three high-performance BVH construction algorithms. In Section 3.2, we proposed a construction algorithm based on k-means clustering targeting many-core GPUs. In Section 3.3, we proposed a construction algorithm based on a progressively refined cut of an existing BVH targeting multi-core CPUs. Finally, in Section 3.4, we proposed another construction algorithm based on locally-ordered clustering targeting many-core GPUs. All three algorithms are highly parallel and provide a good trade-off between construction times and the ray tracing performance. The algorithms are applicable in the context of the real-time rendering, and outperform other high-performance construction algorithms such as ATRBVH or AAC.

G.3 Temporal coherence In Section 3.1, we proposed a method for handling geometry changes in animated scenes. Since we use a single BVH optimized for the whole animation sequence, updating of the BVH reduces to simple refitting of bounding boxes. The method outperforms the state-of-the-art approaches based either on the per-frame reconstruction from scratch or on the best-over-time BVH with refitting.

4.2 Future Work

We see several possible ways how to extend each of the presented methods and summarize the most promising ones.

A drawback of the T-SAH method presented in Section 3.1 is that if the geometric changes are significant, then even highly optimized single BVH will not be good enough to cope with these changes. We experimented with using two optimized BVHs corresponding to the first and second halves of animation frames. We plan to further investigate employing multiple optimized BVHs. We want to look at the adaptive subdivision of the input animation sequence in order to provide a minimal set of BVHs within a given memory budget and maximize the ray tracing performance.

The bottom-up construction algorithm based on parallel locally-ordered clustering presented in Section 3.4 uses the Morton curve to accelerate the nearest neighbors search. We want to investigate how the algorithm behaves with the extended Morton codes [70], which take into account not only the position but also the size of scene primitives. Thanks to simplicity, we believe that the algorithm is a promising candidate for future extensions and optimizations.

We see a significant potential in the last method for the parallel BVH optimization presented in Section 3.5 since there is a number of avenues how to extend it. The algorithm produces BVHs of the highest possible quality and we believe that we can do even better. The algorithm takes each node and tries to insert it into another node in order to reduce the cost. We can generalize this operation from pairs of nodes to k-tuples of nodes. We take k-tuples and try to reinsert them into each other which might lead to lower costs. Furthermore, we would like to investigate the possibility of incorporating spatial splits, which might accelerate ray tracing even more, especially for scenes containing primitives with non-uniform sizes. The problem is that spatial splits are directly associated with the top-down construction and the incorporation into the algorithm might be challenging. Last, we would like to extend the method for wide-BVHs as

it was shown recently that they achieve excellent results not only on CPUs [83] but also on GPUs [88].

Bibliography

[1] Attila Áfra and László Szirmay-Kalos. Stackless Multi-BVH Traversal for CPU, MIC and GPU Ray Tracing. *Computer Graphics Forum*, 33(1):129–140, 2014.

[2] Timo Aila, Tero Karras, and Samuli Laine. On Quality Metrics of Bounding Volume Hierarchies. In *Proceedings of High-Performance Graphics*, pages 101–108. ACM, 2013.

[3] Timo Aila and Samuli Laine. Understanding the Efficiency of Ray Traversal on GPUs. In *Proceedings of High-Performance Graphics*, pages 145–149, 2009.

[4] Ciprian Apetrei. Fast and Simple Agglomerative LBVH Construction. In *Proceedings of Computer Graphics and Visual Computing*, 2014.

[5] Arthur Appel. Some Techniques for Shading Machine Renderings of Solids. In *Proceedings of Spring Joint Computer Conference*, pages 37–45. ACM, 1968.

[6] Rasmus Barringer and Tomas Akenine-Möller. Dynamic Ray Stream Traversal. *ACM Trans. Graph.*, 33(4):151:1–151:9, 2014.

[7] Carsten Benthin, Sven Woop, Ingo Wald, and Attila Áfra. Improved Two-Level BVHs using Partial Re-Braiding. In *Proceedings of High-Performance Graphics*, 2017.

[8] Jon Louis Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, 18(9):509–517, 1975.

[9] Jacco Bikker. Improving Data Locality for Efficient In-Core Path Tracing. *Computer Graphics Forum*, 31(6):1936–1947, 2012.

[10] Nikolaus Binder and Alexander Keller. Efficient Stackless Hierarchy Traversal on GPUs with Backtracking in Constant Time. In *Proceedings of High-Performance Graphics*, pages 41–50, 2016.

[11] Jiří Bittner. *Hierarchical Techniques for Visibility Computations*. Ph.D. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, 2002.

[12] Jiří Bittner, Michal Hapala, and Vlastimil Havran. Fast Insertion-Based Optimization of Bounding Volume Hierarchies. *Computer Graphics Forum*, 32(1):85–100, 2013.

[13] Jiří Bittner, Michal Hapala, and Vlastimil Havran. Incremental BVH Construction for Ray Tracing. *Computers and Graphics*, 47(1):135–144, 2015.

[14] Jiří Bittner and Vlastimil Havran. RDH: Ray Distribution Heuristics for Construction of Spatial Data Structures. In *Proceedings of Spring Conference on Computer Graphics*, pages 61–67. ACM, 2009.

[15] Jiří Bittner and Daniel Meister. T-SAH: Animation Optimized Bounding Volume Hierarchies. *Computer Graphics Forum (Proceedings of Eurographics)*, 34(2):527–536, 2015.

[16] Solomon Boulos, Dave Edwards, Dylan Lacewell, Joe Kniss, Jan Kautz, Peter Shirley, and ngo Wald. Packet-based Whitted and Distribution Ray Tracing. In *Proceedings of Graphics Interface*, pages 177–184. ACM, 2007.

[17] James Clark. Hierarchical Geometric Models for Visible Surface Algorithms. *CACM*, 19(10):547–554, 1976.

[18] Holger Dammertz, Johannes Hanika, and Alexander Keller. Shallow Bounding Volume Hierarchies for Fast SIMD Ray Tracing of Incoherent Rays. *Computer Graphics Forum*, 27:1225–1233(9), 2008.

[19] Holger Dammertz and Alexander Keller. Edge Volume Heuristic - Robust Triangle Subdivision for Improved BVH Performance. In *Proceedings of Symposium on Interactive Ray Tracing*, pages 155–158, 2008.

[20] Tomáš Davidovič, Jaroslav Křivánek, Miloš Hašan, and Philipp Slusallek. Progressive Light Transport Simulation on the GPU: Survey and Improvements. *ACM Transactions on Graphics*, 33(3):29:1–29:19, 2014.

[21] Leonardo Domingues and Hélio Pedrini. Bounding Volume Hierarchy Optimization through Agglomerative Treelet Restructuring. In *Proceedings of High-Performance Graphics*, pages 13–20, 2015.

[22] Philip Dutre, Kavita Bala, Philippe Bekaert, and Peter Shirley. *Advanced Global Illumination.* AK Peters Ltd, 2006.

[23] Christer Ericson. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology) (The Morgan Kaufmann Series in Interactive 3D Technology).* Morgan Kaufmann Publishers Inc., 2004.

[24] Manfred Ernst and Gunther Greiner. Early Split Clipping for Bounding Volume Hierarchies. In *Proceedings of Symposium on Interactive Ray Tracing*, pages 73–78, 2007.

[25] Manfred Ernst and Gunther Greiner. Multi bounding Volume Hierarchies. In *Proceedings of Symposium on Interactive Ray Tracing*, pages 35–40, 2008.

[26] Bartosz Fabianowski, Colin Fowler, and John Dingliana. A Cost Metric for Scene-Interior Ray Origins. pages 49–52, 2009.

[27] Akira Fujimoto, Takayuki Tanaka, and Kansei Iwata. Tutorial: Computer Graphics; Image Synthesis. chapter ARTS: Accelerated Ray-tracing System, pages 148–159. Computer Science Press, Inc., 1988.

[28] Per Ganestam, Rasmus. Barringer, Michael Doggett, and Thomas Akenine-Möller. Bonsai: Rapid Bounding Volume Hierarchy Generation using Mini Trees. *Journal of Computer Graphics Techniques*, 4(3):23–42, 2015.

[29] Kirill Garanzha, Jacopo Pantaleoni, and David McAllister. Simpler and Faster HLBVH with Work Queues. In *Proceedings of High-Performance Graphics*, pages 59–64, 2011.

[30] Andrew Glassner. Tutorial: Computer Graphics; Image Synthesis. chapter Space Subdivision for Fast Ray Tracing, pages 160–167. Computer Science Press, Inc., 1988.

[31] Jeffrey Goldsmith and John Salmon. Automatic Creation of Object Hierarchies for Ray Tracing. *Comput Graphics and Applications*, 7(5):14–20, 1987.

[32] Christiaan Gribble and Karthik Ramani. Coherent Ray Tracing via Stream Filtering. In *Proceeding of Symposium on Interactive Ray Tracing*, pages 59–66, 2008.

[33] Yan Gu, Yong He, Kayvon Fatahalian, and Guy Blelloch. Efficient BVH Construction via Approximate Agglomerative Clustering. In *Proceedings of High-Performance Graphics*, pages 81–88, 2013.

[34] Michael Guthe. Latency Considerations of Depth-first GPU Ray Tracing. In *Proceedings of Eurographics (Short Papers)*. The Eurographics Association, 2014.

[35] Michal Hapala, Tomáš Davidovič, Ingo Wald, Vlastimil Havran, and Philipp Slusallek. Efficient Stack-less BVH Traversal for Ray Tracing. In *Proceedings of Spring Conference on Computer Graphics*, pages 7–12. ACM, 2013.

[36] Vlastimil Havran. *Heuristic Ray Shooting Algorithms*. Ph.D. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, 2000.

[37] Vlastimil Havran, Robert Herzog, and Hans-Peter Seidel. On the Fast Construction of Spatial Data Structures for Ray Tracing. In *Proceedings of Symposium on Interactive Ray Tracing 2006*, pages 71–80, 2006.

[38] Jakub Hendrich, Daniel Meister, and Jiří Bittner. Parallel BVH Construction Using Progressive Hierarchical Refinement. *Computer Graphics Forum (Proceedings of Eurographics)*, 36(2):487–494, 2017.

[39] Warren Hunt, William Mark, and Don Fussell. Fast and Lazy Build of Acceleration Structures from Scene Hierarchies. In *Proceedings of Symposium on Interactive Ray Tracing*, pages 47–54, 2007.

[40] Thiago Ize, Ingo Wald, and Steven Parker. Asynchronous BVH Construction for Ray Tracing Dynamic Scenes on Parallel Multi-Core Architectures. In *Proceedings of Symposium on Parallel Graphics and Visualization*, pages 101–108, 2007.

[41] James Kajiya. The Rendering Equation. *SIGGRAPH Computer Graphics*, 20(4):143–150, 1986.

[42] Tero Karras. Maximizing Parallelism in the Construction of BVHs, Octrees, and k-d Trees. In *Proceedings of High-Performance Graphics*, pages 33–37, 2012.

[43] Tero Karras and Timo Aila. Fast Parallel Construction of High-Quality Bounding Volume Hierarchies. In *Proceedings of High-Performance Graphics*, pages 89–100. ACM, 2013.

[44] Csaba Kelemen, László Szirmay-Kalos, György Antal, and Ferenc Csonka. A Simple and Robust Mutation Strategy for the Metropolis Light Transport Algorithm. *Computer Graphics Forum*, 21(3):531–540, 2002.

[45] Andrew Kensler. Tree Rotations for Improving Bounding Volume Hierarchies. In *Proceedings of Symposium on Interactive Ray Tracing*, pages 73–76, 2008.

[46] Daniel Kopta, Thiago Ize, Josef Spjut, Erik Brunvand, Al Davis, and Andrew Kensler. Fast, Effective BVH Updates for Animated Scenes. In *Proceedings of Symposium on Interactive 3D Graphics and Games*, pages 197–204, 2012.

[47] Samuli Laine. Restart Trail for Stackless BVH Traversal. In *Proceedings of High-Performance Graphics*, pages 107–111. Eurographics Association, 2010.

[48] Christian Lauterbach, Michael Garland, Shubhabrata Sengupta, David Luebke, and Dinesh Manocha. Fast BVH Construction on GPUs. *Computer Graphics Forum*, 28(2):375–384, 2009.

[49] David MacDonald and Kellogg Booth. Heuristics for Ray Tracing Using Space Subdivision. *Visual Computer*, 6(3):153–65, 1990.

[50] Erik Mansson, Jacob Munkberg, and Tomas Akenine-Moller. Deep Coherent Ray Tracing. In *Proceedings of Symposium on Interactive Ray Tracing*, pages 79–85. IEEE Computer Society, 2007.

[51] Daniel Meister and Jiří Bittner. Parallel BVH Construction Using k-means Clustering. *Visual Computer (Proceedings of Computer Graphics International)*, 32(6-8):977–987, 2016.

[52] Daniel Meister and Jiří Bittner. Parallel Locally-Ordered Clustering for Bounding Volume Hierarchy Construction. *IEEE Transactions on Visualization and Computer Graphics*, 24(3):1345–1353, 2018.

[53] Daniel Meister and Jiří Bittner. Parallel Reinsertion for Bounding Volume Hierarchy Optimization. *Computer Graphics Forum (Proceedings of Eurographics)*, 37(2):463–473, 2018.

[54] Tomas Möller and Ben Trumbore. Fast, Minimum Storage Ray-triangle Intersection. *Journal of Graphics Tools*, 2(1):21–28, 1997.

[55] Guy Morton. A Computer Oriented Geodetic Database and a New Technique in File Sequencing. Technical report, 1966.

[56] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable Parallel Programming with CUDA. *Queue*, 6(2):40–53, 2008.

[57] Ola Olsson. Constructing High-Quality Bounding Volume Hierarchies for N-Body Computation Using the Acceptance Volume Heuristic. *Astronomy and Computing*, 22:1–8, 2018.

[58] Ryan Overbeck, Ravi Ramamoorthi, and William Mark. Large Ray Packets for Real-Time Whitted Ray Tracing. In *Proceedings of Symposium on Interactive Ray Tracing*, pages 41–48, 2008.

[59] Jacopo Pantaleoni and David Luebke. HLBVH: Hierarchical LBVH Construction for Real-Time Ray Tracing of Dynamic Geometry. In *Proceedings of High-Performance Graphics*, pages 87–95, 2010.

[60] Steven Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. OptiX: A General Purpose Ray Tracing Engine. *ACM Transactions on Graphics*, 29(4):66:1–66:13, 2010.

[61] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., 2004.

[62] Stefan Popov, Iliyan Georgiev, Rossen Dimov, and Philipp Slusallek. Object Partitioning Considered Harmful: Space Subdivision for BVHs. In *Proceedings of High-Performance Graphics*, pages 15–22, 2009.

[63] Steven Rubin and Turner Whitted. A 3-dimensional Representation for Fast Rendering of Complex Scenes. *SIGGRAPH Computer Graphics*, 14(3):110–116, 1980.

[64] Martin Stich, Heiko Friedrich, and Andreas Dietrich. Spatial Splits in Bounding Volume Hierarchies. In *Proceedings of the High-Performance Graphics*, pages 7–13. ACM, 2009.

[65] László Szirmay-Kalos and Gabor Márton. Worst-case Versus Average Case Complexity of Ray-shooting. *Computing*, 61(2):103–131, 1998.

[66] John Tsakok. Faster Incoherent Rays: Multi-BVH Ray Stream Tracing. In *Proceedings of High-Performance Graphics*, pages 151–158, 2009.

[67] Kostas Vardis, Andreas Vasilakis, and Georgios Papaioannou. A Multiview and Multilayer Approach for Interactive Ray Tracing. In *Proceedings of Symposium on Interactive 3D Graphics and Games*, pages 171–178. ACM, 2016.

[68] Kostas Vardis, Andreas Vasilakis, and Georgios Papaioannou. DIRT: Deferred Image-based Ray Tracing. In *Proceedings of High-Performance Graphics*, pages 63–73. Eurographics Association, 2016.

[69] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, 1998. AAI9837162.

[70] Marek Vinkler, Jiří Bittner, and Vlastimil Havran. Extended Morton Codes for High Performance Bounding Volume Hierarchy Construction. In *Proceedings of High-Performance Graphics*, 2017.

[71] Marek Vinkler, Jiří Bittner, Vlastimil Havran, and Michal Hapala. Massively Parallel Hierarchical Scene Processing with Applications in Rendering. *Computer Graphics Forum*, 32(8):13–25, 2013.

[72] Marek Vinkler and Vlastimil Havran. Register Efficient Dynamic Memory Allocator for GPUs. *Computer Graphics Forum*, 34(8):143–154, 2015.

[73] Marek Vinkler, Vlastimil Havran, and Jiří Bittner. Performance Comparison of Bounding Volume Hierarchies and Kd-Trees for GPU Ray Tracing. *Computer Graphics Forum*, 2016.

[74] Marek Vinkler, Vlastimil Havran, Jiří Bittner, and Jiří Sochor. Parallel On-Demand Hierarchy Construction on Contemporary GPUs. *IEEE Transactions on Visualization and Computer Graphics*, 22(99):1886–1898, 2016.

[75] Marek Vinkler, Vlastimil Havran, and Jiří Sochor. Visibility Driven BVH Build Up Algorithm for Ray Tracing. *Computers and Graphics*, 36(4):283–296, 2012.

[76] Ingo Wald. On Fast Construction of SAH-based Bounding Volume Hierarchies. In *Proceedings of Symposium on Interactive Ray Tracing*, pages 33–40, 2007.

[77] Ingo Wald. Fast Construction of SAH BVHs on the Intel Many Integrated Core (MIC) Architecture. *IEEE Transactions on Visualization and Computer Graphics*, 18(1):47–57, 2012.

[78] Ingo Wald, Carsten Benthin, and Solomon Boulos. Getting Rid of Packets - Efficient SIMD Single-Ray Traversal using Multi-Branching BVHs. In *Symposium on Interactive Ray Tracing*, pages 49–57, 2008.

[79] Ingo Wald, Carsten Benthin, and Philipp Slusallek. Distributed Interactive Ray Tracing of Dynamic Scenes. In *Proceedings of Symposium on Parallel and Large-Data Visualization and Graphics*, pages 77–86, 2003.

[80] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray Tracing Deformable Scenes Using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics*, 26(1), 2007.

[81] Ingo Wald and Vlastimil Havran. On building fast KD-trees for ray tracing, and on doing that in O(N log N). In *Proceedings of Symposium on Interactive Ray Tracing*, pages 61–69, 2006.

[82] Ingo Wald, Thiago Ize, and Steven Parker. Fast, Parallel, and Asynchronous Construction of BVHs for Ray Tracing Animated Scenes. *Computers and Graphics*, 32(1):3–13, 2008.

[83] Ingo Wald, Sven Woop, Carsten Benthin, Gregory Johnson, and Manfred Ernst. Embree: A Kernel Framework for Efficient CPU Ray Tracing. *ACM Transactions on Graphics*, 33, 2014.

[84] Bruce Walter, Kavita Bala, Milind Kulkarni, and Keshav Pingali. Fast Agglomerative Clustering for Rendering. In *Proceedings of Symposium on Interactive Ray Tracing*, pages 81–86, 2008.

[85] Hank Weghorst, Gary Hooper, and Donald Greenberg. Improved Computational Methods for Ray Tracing. *ACM Transactions on Graphics*, 3(1):52–69, 1984.

[86] Turner Whitted. An Improved Illumination Model for Shaded Display. *Communications of the ACM*, 23(6):343–349, 1980.

[87] Sven Woop, Attila Áfra, and Carsten Benthin. STBVH: A Spatial-temporal BVH for Efficient Multi-segment Motion Blur. In *Proceedings of High-Performance Graphics*, pages 8:1–8:8. ACM, 2017.

[88] Henri Ylitie, Tero Karras, and Samuli Laine. Efficient Incoherent Ray Traversal on GPUs Through Compressed Wide BVHs. In *Proceedings of High-Performance Graphics*, pages 4:1–4:13, 2017.

[89] Sung-Eui Yoon, Sean Curtis, and Dinesh Manocha. Ray Tracing Dynamic Scenes using Selective Restructuring. In *Proceedings of Eurographics Symposium on Rendering*, pages 73–84, 2007.

A Author's Publications

The following research papers were co-authored by the author of the thesis and published in journals with impact factor, and all are related to the thesis.

Jiří Bittner and Daniel Meister. T-SAH: Animation Optimized Bounding Volume Hierarchies. *Computer Graphics Forum (Proceedings of Eurographics)*, 34(2):527–536, 2015 (IF = 1.542)

Cited in:

Kostas Vardis, Andreas Vasilakis, and Georgios Papaioannou. DIRT: Deferred Image-based Ray Tracing. In *Proceedings of High-Performance Graphics*, pages 63–73. Eurographics Association, 2016

Kostas Vardis, Andreas Vasilakis, and Georgios Papaioannou. A Multiview and Multilayer Approach for Interactive Ray Tracing. In *Proceedings of Symposium on Interactive 3D Graphics and Games*, pages 171–178. ACM, 2016

Sven Woop, Attila Áfra, and Carsten Benthin. STBVH: A Spatial-temporal BVH for Efficient Multi-segment Motion Blur. In *Proceedings of High-Performance Graphics*, pages 8:1–8:8. ACM, 2017

Daniel Meister and Jiří Bittner. Parallel BVH Construction Using k-means Clustering. *Visual Computer (Proceedings of Computer Graphics International)*, 32(6-8):977–987, 2016 (IF = 1.468)

Cited in:

Ola Olsson. Constructing High-Quality Bounding Volume Hierarchies for N-Body Computation Using the Acceptance Volume Heuristic. *Astronomy and Computing*, 22:1–8, 2018

Jakub Hendrich, Daniel Meister, and Jiří Bittner. Parallel BVH Construction Using Progressive Hierarchical Refinement. *Computer Graphics Forum (Proceedings of Eurographics)*, 36(2):487–494, 2017 (IF $\sim$ 1.6)

Cited in:

Carsten Benthin, Sven Woop, Ingo Wald, and Attila Áfra. Improved Two-Level BVHs using Partial Re-Braiding. In *Proceedings of High-Performance Graphics*, 2017

Daniel Meister and Jiří Bittner. Parallel Locally-Ordered Clustering for Bounding Volume Hierarchy Construction. *IEEE Transactions on Visualization and Computer Graphics*, 24(3):1345–1353, 2018 (IF $\sim$ 2.8)

Daniel Meister and Jiří Bittner. Parallel Reinsertion for Bounding Volume Hierarchy Optimization. *Computer Graphics Forum (Proceedings of Eurographics)*, 37(2):463–473, 2018 (IF $\sim$ 1.6)

B Authorship Contribution Statement

This statement describes the specific contributions of the author of this thesis to the presented publications.

T-SAH: Animation Optimized Bounding Volume Hierarchies (Bittner 60%, Meister 40%) I extended the Aila's GPU framework [3] to support animated scenes, implemented reference methods, performed the final evaluation, created the accompanying video, created the main table and plots, and rendered all images in the paper.

Parallel BVH Construction using k-means Clustering (Meister 70%, Bittner 30%) I came with the initial idea, implemented the prototype, did all experiments, performed the final evaluation, wrote most of the paper, created almost all figures, and created the accompanying video.

Parallel BVH Construction using Progressive Hierarchical Refinement (Hendrich 70 %, Meister 15%, Bittner 15%) I integrated the algorithm into Embree [60], where I performed the final evaluation. In particular, I created the main table and rendered all images in the paper.

Parallel Locally-Ordered Clustering for Bounding Volume Hierarchy Construction (Meister 70%, Bittner 30%) I came with the initial idea, implemented the prototype, did all experiments, performed the final evaluation, wrote most of the paper, created almost all figures, and created the accompanying video.

Parallel Reinsertion for Bounding Volume Hierarchy Optimization (Meister 70%, Bittner 30%) I came with the initial idea, implemented the prototype, did all experiments, performed the final evaluation, wrote most of the paper, and created all figures.

Daniel Meister

* 4th June, 1989

Education

Czech Technical University in Prague:

2014 – present	Ph.D. in Information Science and Computer Engineering
2012 – 2014	M.Sc. in Computer Graphics and Interaction
2009 – 2012	B.Sc. in Software Engineering

Work Experience

2014/10 – 2017/3	External Developer (Interactive Rendering System), Škoda Auto
2017/11 – present	Researcher, Toyota Research Lab, CTU in Prague

Computer Skills

C/C++, CUDA, OpenGL, SIMD, Matlab, Python, LaTeX

Research Interests

Data Structures for Ray Tracing, Real-Time Ray Tracing, GPGPU, Parallel Computing, Global Illumination

Professional Visits Abroad

2017	National Institute of Informatics, Japan (5 months)
2014	Vienna University of Technology, Austria (1 month)

Projects

- 2014 – 2017 *Development Adaptive Interactive System for Increasing Safety of Vehicle Crew and its Use for Evaluation of Pavement Surface Characteristics* (TA04031769), Technology Agency of the Czech Republic, Project External Team Member
- 2014 – 2015 *Optimal Algorithms for Image Synthesis* (GAP202/12/2413), The Czech Science Foundation, Project Team Member
- 2013 – 2014 *Global Illumination for Augmented Reality in General Environments* (GAP202/11/1883), The Czech Science Foundation, Project Team Member

Languages

Czech	Native Language
English	Fluent
Japanese	Intermediate (JLPT N3)
French	Basic Knowledge
Spanish	Basic Knowledge

Professional Society Membership

Upsilon Pi Epsilon Honor Society