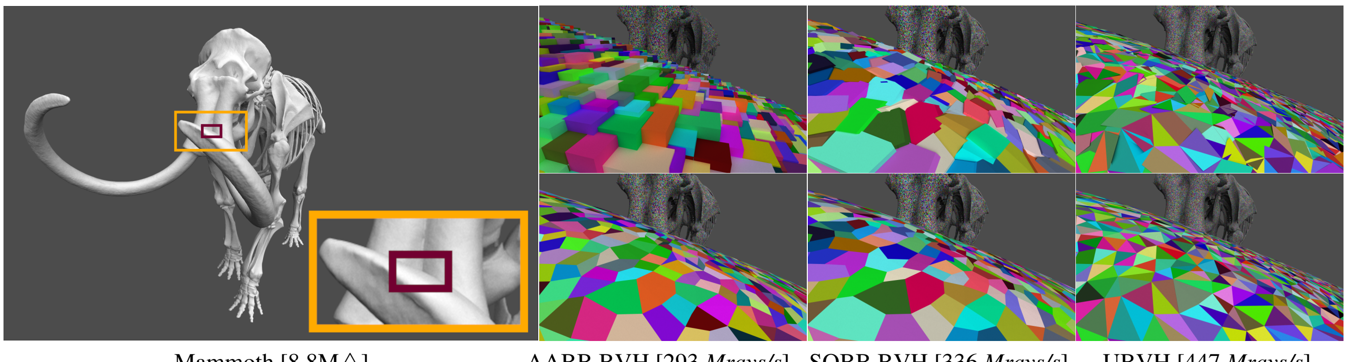
High-Performance Graphics 2025
A. Knoll and C. Peters
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 44 (2025), Number 8

UBVH: Unified Bounding Volume and Scene Geometry Representation for Ray Tracing

M. Káčerik and J. Bittner

Czech Technical University in Prague, Faculty of Electrical Engineering, Czechia



Mammoth [8.8M△] AABB BVH [293 *Mrays/s*] SOBB BVH [336 *Mrays/s*] UBVH [447 *Mrays/s*]

Figure 1: *Path traced image of the Mammoth scene with 8.8M triangles (left). Visualization of the leaf bounding volumes (top row) and scene triangles (bottom row) on the left tusk. Triangles stored in the same leaf node share the color with the corresponding bounding volume. Traditionally, the bounding volumes and triangles have to be tested for intersection separately, while for UBVH the test is practically identical, yielding an efficient traversal algorithm. The numbers in brackets show the ray tracing speed for incoherent secondary rays*

.

Abstract

Bounding volume hierarchies (BVHs) are currently the most common data structure used to accelerate ray tracing. The existing BVH methods distinguish between the bounding volume representation associated with the interior BVH nodes and the scene geometry representation associated with leaf nodes. We propose a new method that unifies the representation of bounding volumes and triangular scene geometry. Our unified representation builds on skewed oriented bounding boxes (SOBB) that yield tight bounds for interior nodes and precise representation for triangles in the leaf nodes. This innovation allows to streamline the conventional massively parallel BVH traversal, as there is no need to switch between testing for ray intersection in interior nodes and leaf nodes. The results show that the proposed method accelerates ray tracing of incoherent rays between 1.2x–11.8x over the AABB BVH, 1.4x-4.2x over the 14*-DOP BVH, 1.1x-2.0x over the OBB BVH, and by 1.1x–1.7x over the SOBB BVH.*

CCS Concepts

• ***Computing methodologies*** → ***Ray tracing;*** *Massively parallel algorithms;*

1. Introduction

Traditionally, we think of a *Bounding Volume Hierarchy* (BVH) as a tree consisting of interior nodes representing the bounding volumes and leaf nodes pointing to the geometric primitives (as illustrated in Fig. 2, left). The separation of bounding volumes and the actual scene geometry naturally leads to the two-stage tree traversal. To determine the ray-geometry intersection, in the first stage the ray is repeatedly tested for the intersection with the bounding volumes of interior nodes. Upon reaching the leaf node, in the second stage, the ray is tested using a different routine to find the geometry intersection. This complicates the traversal algorithm, especially in the massively parallel context, where abrupt switches of the context lead to thread execution divergence and hardware under-utilization.

Unifying the representation of bounding volumes and geometry

© 2025 The Author(s). Computer Graphics Forum published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.
This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

(as depicted in Fig. 2, right) opens the possibility of merging the two previously distinct traversal stages while utilizing an identical intersection test for the bounding volumes of the hierarchy and the triangular scene geometry. This increases the overall efficiency of the traversal in terms of hardware utilization as well as the total number of evaluated intersection tests.

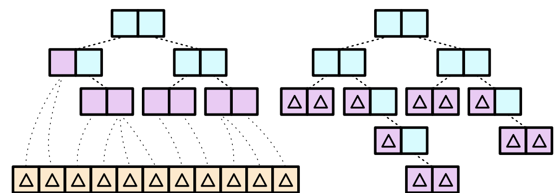


Figure 2: *Traditional BVH stores bounding volumes in interior nodes of the hierarchy and triangles are referenced in the leaf nodes (left). The UBVH stores the triangles directly inside the hierarchy in the same way as the bounding volumes (right).*

Our main contributions are:

- Unified BVH (UBVH), a data structure that unifies the representation of bounding volumes and triangular geometry.
- Efficient traversal algorithm for the UBVH, using identical intersection tests for both the interior nodes and leaf nodes.
- Efficient way of encoding triangle pairs in the UBVH and an algorithm to find such pairs.
- Verification of the proposed methods and their comparison with AABB, *14*-DOP, OBB, and SOBB BVHs on representative scenes.

2. Related Work

A long history of research on the bounding volume hierarchies in the context of ray tracing was recently surveyed by Meister et al. [MOB*21]. BVH construction process is often guided by a heuristic that estimates its performance. Most state-of-the-art builders attempt to minimize the *surface area heuristic* (SAH) [GS87; AKL13]. For our line of research, the most relevant are bottom-up builders, that by design construct trees with leaves of one primitive. They were brought to GPUs by Meister and Bittner with *parallel locally-ordered clustering* (PLOC) [MB18] and later refined by Benthin et al. as *PLOC++* [BDTD22] and *H-PLOC* [BMB*24].

Dominating bounding volume for ray tracing BVHs is an *axis-aligned bounding box* (AABB), mainly due to its simplicity. However, it is well-known that it provides a loose fit to many geometric configurations, leading to substantial overlaps of the volumes in the hierarchy. Furthermore, in such cases, it also generates many unnecessary intersection tests compared to tighter bounding volumes with smaller surface areas. A method to find an optimal *oriented bounding box* (OBB) was presented by O'Rourke [ORo85], but it is too slow for a practical workload. Multiple approximate OBB methods were proposed [Got00; CGM11]. The *di-tetrahedron OBB algorithm* (DiTO) by Larsson and Källberg [LK11] was recently parallelized by Vitsas et al. [VEPG23] for fast OBB hierarchy construction. For OBB estimation, DiTO uses an intermediate representation of the bounding volume in the form of a *discrete orientation polytope* (k-DOP) [KZ97; KHM*98; SVCM23]. Recently, Káčerik and Bittner [KB24] presented a fast method to construct a SAH-optimized *14*-DOP hierarchy and later proposed using the intermediate k-DOP representation for a fast construction of a hierarchy of *skewed oriented bounding boxes* (SOBB) [KB25]. In different context, Calderon and Boubekeur proposed mesh-based bounding proxies [CB17] which provide a tight fit to the underlying geometry, but their potential usage for ray tracing is yet unclear.

Instead of replacing the AABBs to achieve tighter bounding, the geometry of the scene could be split into smaller parts, which can be bounded by their own small AABBs [KA13], or utilized for a top-down BVH construction with spatial splits [FLPE16].

Yet another approach is to optimize the memory layout and memory access patterns to efficiently map the problem to modern hardware. Wald et al. [WBB08] presented the way to convert a binary BVH to a wide BVH. Later, Ylitie et al. showed how to do the conversion in a SAH-optimal fashion and presented a compressed wide layout [YKL17]. Efficient traversal of the compressed wide BVHs was addressed by Vaidyanathan et al. [VWB19]. Meister and Bittner conducted a thorough comparison of several state-of-the-art GPU-oriented BVH methods [MB22].

To reduce the memory consumption of the geometry referenced by the BVH, *ray-strips* by Lauterbach et al. [LYM07] exploit the connectivity of the triangles followed by SAH-aware variant [LYTM08]. Segovia and Ernst [SE10] presented *hierarchical mesh quantization*, where they were able to fit the geometry directly to a BVH node by compressing the triangle vertices as local offsets to the bounding planes. *No-Memory BVH* by Eisemann et al. [EBM12] inverted the approach by implicitly storing the bounding volume as reordered scene triangles. For parametric models such as Bézier patches or curves tight bounding volumes can be constructed directly from the parametric representation and used as a geometry proxy for intersection test [PK87; BK18].

As far as we know, there was no previous work aiming to unify the representation of bounding volumes and scene geometry for BVHs or directly use compressed representation of triangle pairs for ray intersection evaluation.

3. Unified Representation

Our work builds on the recently introduced skewed oriented bounding boxes (SOBBs) [KB25]. SOBBs were originally proposed as a general bounding volume for BVHs to accelerate ray tracing. We extend this idea further by observing that SOBBs have great potential to precisely represent triangular geometry.

In this section, we briefly review the most important properties of SOBBs and then present the way of storing triangles as SOBBs together with a cheap verification test to determine a ray-triangle intersection for a positive ray-SOBB intersection.

3.1. Skewed Oriented Bounding Box

SOBB generalizes a well-known oriented bounding box by relaxing the orthogonality constraint of its basis. Therefore, SOBB is a

© 2025 The Author(s).
Computer Graphics Forum published by Eurographics and John Wiley & Sons Ltd.

parallelepiped whose faces are three pairs of congruent parallelograms. For the purpose of an efficient ray-SOBB intersection test, the authors proposed representing the SOBB as an intersection of three slabs, where each slab **s** consists of a normal **n** defining the slab plane together with minimal (d_{min}) and maximal (d_{max}) projections along this normal, defining the slab boundaries. Each slab is then encoded as a four dimensional vector:

$$\mathbf{s} = \begin{bmatrix} s_x & s_y & s_z & s_d \end{bmatrix}^T = \frac{\begin{bmatrix} n_x & n_y & n_z & d_{min} \end{bmatrix}^T}{\max(d_{max} - d_{min}, \epsilon)}. \quad (1)$$

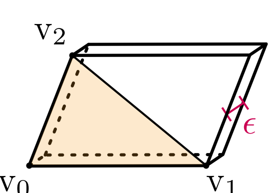
The ϵ is used as a minimal width of the slab to prevent division by zero when encoding the SOBB (we used $\epsilon = 10^{-7}$).

The SOBBs are able to provide tight bounds of the contained geometry (possibly even tighter than the OBBs). At the same time, the k-DOP based algorithm for tight SOBB computation [KB25] is significantly faster than the the ditetrahedron-based state-of-the-art algorithm for OBBs [VEPG23].

3.2. Triangle as SOBB

Any triangle can be expanded to a parallelogram such that its diagonal divides the parallelogram into two congruent triangles. This parallelogram can then serve as a face of a parallelepiped, forming an SOBB. If the SOBB is constructed with the width of ϵ, a necessity of the slab-based representation, it is also the tightest possible non-degenerate bounding SOBB for the given triangle (see Fig. 3, left). We call the resulting BVH where all nodes are encoded as SOBBs and leaf SOBBs encode individual scene triangles *unified BVH* (UBVH).

During ray tracing, we can use a standard slab-based intersection test to find a ray-SOBB intersection in the exact same way as we do for any other SOBB in the hierarchy. However, once we determine an intersection with the SOBB representing the triangle, we have to perform an additional check to determine whether the actual ray-triangle intersection occurred.



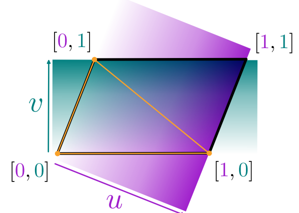


Figure 3: *Left: a triangle placed on the surface of a tight bounding SOBB with the width of ϵ. Right: slab relative coordinates over the surface of the triangle. These coordinates grow from 0 to 1 in the span of the slab.*

We utilize a property of the SOBB representation that directly provides the relative coordinates of the intersection point within the slab. This follows from the fact that the transformation described in Eq. 1 scales the world space so that the width of the slab becomes a unit length, with ϵ ensuring that the slab does not collapse into a plane. The slab **s** contains all the points **p** that satisfy the inequality:

$$u = \mathbf{p} \cdot \begin{bmatrix} s_x & s_y & s_z \end{bmatrix}^T - s_d \quad (2)$$

$$\mathbf{p} \in \mathbf{s} \iff 0 \leq u \leq 1. \quad (3)$$

The value u obtained by adjusting the scalar projection by s_d represents a relative distance from the lower limit of the slab, and we will refer to it as a slab relative coordinate. The slab triplet then defines a space with slab relative coordinates u, v, and w, in which all points lying on the SOBB surface have at least one of these coordinates equal to 0 or 1, while the other two coordinates, assuming that they hold the inequality above, describe a relative position within the corresponding parallelogram.

We can verify that the ray-triangle intersection happened simply by projecting the ray-SOBB intersection point **p** to the SOBB unit space and evaluating, whether the position corresponds to the known position of the triangle T within the parallelogram. We construct the SOBB so that the triangle is always aligned with $w = 0$ plane and therefore:

$$\mathbf{p} \in T \iff u + v \leq 1. \quad (4)$$

See Fig. 3, right, for an illustration of this equation. Note that the relative slab coordinates are closely related to the barycentric coordinates of the enclosed triangle. With respect to layout shown in Fig. 3, barycentric coordinates λ_0, λ_1, and λ_2 corresponding to vertices v_0, v_1, v_2 can be computed trivially:

$$\lambda_1 = 1 - u, \quad \lambda_2 = 1 - v, \quad \lambda_0 = 1 - \lambda_1 - \lambda_2. \quad (5)$$

3.3. Triangle pair as SOBB

Working with pairs of triangles sharing a common edge instead of singular triangles is known to provide multiple benefits: reduced memory footprint, faster bottom-up BVH construction due to the reduced number of clusters, and a potentially more efficient intersection test [WWB*14; BMB*24]. We adopt this idea for our UBVH representation and encode suitable triangle pairs as a single SOBB. We call the resulting data structure *extended UBVH*.

We distinguish three configurations of triangle pairs with a common edge: (1) coplanar triangles with central symmetry over the midpoint of the shared edge, forming a parallelogram, (2) arbitrary coplanar triangles, and (3) butterfly formation of two noncoplanar triangles, resembling the shape of wings.

3.3.1. Coplanar symmetric pair

The case of central symmetry over the midpoint of the shared edge is a straightforward extension of the single triangle approach, as depicted in Fig. 4, left. If inequality $u + v \leq 1$ is true, the point p belongs to the first triangle (orange); otherwise, it belongs to the second triangle (cyan).

3.3.2. Coplanar asymmetric pair

Although arbitrary pair of coplanar triangles can always be bounded by a parallelogram (see Fig. 4, right), there is no obvious way to determine the ray-triangle intersection using only the slab relative coordinates (i.e., without additional information). For

© 2025 The Author(s).

this reason, we treat this pair configuration as two separate triangles with separate SOBB representation.

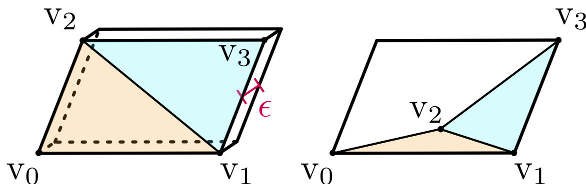


Figure 4: *Left: a coplanar triangle pair with central symmetry over the edge midpoint, placed on the surface of a tight bounding SOBB with the width of ϵ. Right: a coplanar triangle pair with no symmetry bounded by a parallelogram.*

3.3.3. Butterfly pair

Any noncoplanar pair of triangles can be encoded on the surface of an SOBB as shown in Fig. 5. Also, with a careful approach, the ray-triangle intersection can be determined directly from the relative slab coordinates, similarly to the case of a single triangle.

Depending on the butterfly shape, resulting SOBB might provide a poor fit for the pair, including a lot of empty space. However, due to the implicit clipping of hierarchies of oriented bounding volumes [KB25] this problem is mitigated. This enlarged bounding volume is only used to encode the triangle pair and is not used to compute the parent bounding volumes (these are fitted independently as we discuss in Section 4.1.1). The empty space is thus effectively clipped by the parent bounding volumes. This observation is supported by measurements that show a very low number of ray-triangle intersection tests for the proposed representation.

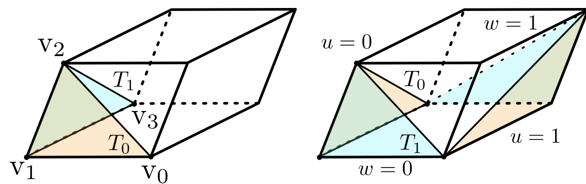


Figure 5: *Left: a nonplanar, convex triangle pair, placed on the surface of a tight bounding SOBB. Right: possible placements of a concave pair, either with flipped order or on the SOBB backside.*

Convex and concave triangle pair configuration. Assuming outward pointing normal vectors, the relative position of the triangles as shown in Fig. 5, left, is what we consider a convex configuration. Were the normals pointing inwards, in a so called concave configuration, placing the triangles in the same relative position within the SOBB would lead to an inconsistency between the configurations. We identified two ways of addressing this issue: either the concave pair is encoded on the other side of the SOBB, or the triangles are flipped, maintaining the same relative SOBB placement with different triangle order, as shown in Fig. 5, right. We implemented the second option, as it allows to share the same intersection test for both convex and concave configurations.

Intersection test. We construct the SOBB in such a way that the first triangle T_0 occupies the face with the relative slab coordinate $w = 0$ and the other triangle T_1 occupies the face with $u = 0$. Similarly to a single triangle test, valid ray-SOBB intersection point p intersects one of the triangles if following conditions are met:

$$\begin{aligned} \mathbf{p} \in T_0 &\iff w = 0 \wedge u + v \leq 1 \\ \mathbf{p} \in T_1 &\iff u = 0 \wedge v + w \leq 1. \end{aligned} \tag{6}$$

3.3.4. Barycentric coordinates for triangle pairs

As we do not impose constraints on the common edge of a pair of triangles, the barycentric coordinates can no longer be computed as trivially as for the single triangle with consistent relative position of triangle vertices and SOBB slabs. Although the mapping between the slab coordinates and the vertices of triangles could be maintained, we decided to simply recompute the barycentric coordinates in the shading phase of the ray tracing pipeline.

3.4. Precision

Encoding triangles on the SOBB surface comes with potential precision issues when encoding single triangles and near-planar butterfly pairs.

Single triangles. Unlike in Eq. 6 for butterflies, in Eq. 4 for triangles we do not explicitly consider the coordinate w. For a single triangle, the w corresponds to the ϵ wide slab, where the values 0 and 1 correspond to the front and back sides of the SOBB (see Fig. 3, left, for reference). As the triangle is placed on the SOBB front side ($w = 0$), disregarding w slightly distorts the resulting intersection for rays coming from the back due to the ϵ difference, but in turn simplifies the intersection test. We used this faster intersection test as it did not yield visible artifacts in our test scenes. For an accurate handling of backface hits, it is possible to explicitly ignore the hits on the back side of the SOBB.

Near-planar butterflies. With finite precision, butterflies with almost identical normals can introduce numerical issues in the uvw space due to extremely thin and nearly parallel SOBB slabs. Therefore, we used a fixed threshold angle of $5 \cdot 10^{-4}$ rad, below which the pair is dismissed and the triangles are considered separately.

4. BVH Construction and Traversal

Our UBVH structure is constructed by transforming the bounding volumes of an existing AABB hierarchy, similar to the way a SOBB BVH is built [KB25]. The main differences are the assembly of the leaf nodes and the approach to BVH traversal.

4.1. BVH construction

First, we construct the AABB BVH with a single primitive per leaf and proceed with the bottom-up BVH transformation to obtain SOBB for each BVH node.

4.1.1. Transformation to UBVH

For each leaf, we retrieve the associated triangle (or pair of triangles) and encode it on the surface of a SOBB, as described in Sec. 3. Then we continue the bottom-up pass, merging child nodes

© 2025 The Author(s).

at each step, using an intermediate k-DOP representation to estimate the merged SOBB in the same fashion as the SOBB BVH is constructed [KB25].

Leaf nodes are explicitly positioned before the interior nodes in the currently processed node's child list. Due to sequential evaluation of ray-child intersection tests (explained in detail in Sec. 4.2), early evaluation of the ray-triangle intersection that yields an intersection has the potential to cull the more distant subtrees rooted in the following interior nodes. This potential is even greater for wide BVHs with more child nodes.

4.1.2. Triangle pairs discovery

Standard UBVH requires one primitive per leaf node, while extended UBVH can also handle specific triangle pairs with a shared edge, as described in Sec. 3.3. As scenes typically consist of a large number of neighboring triangles, this approach can provide up to 50% reduction of primitives and thus the associated memory usage. However, this process can be costly, and its efficiency is related to the structure of a given scene.

Finding suitable triangle pairs can be approached in many ways with varying trade-offs. We propose two pair discovery algorithms: (1) a simple greedy parallel algorithm that attempts to merge a triangle with its direct neighbors in the input triangle array and (2) more involved pairing based on parallel locally-ordered clustering (PLOC) [MB18].

Simple greedy pairs. In parallel, each triangle attempts to merge with its successor, accepting valid pairs in a greedy manner using atomic instructions. This approach is strongly tied to the memory layout of the input scene representation and could require topological layout optimization to be successful.

PLOC-based pairing. This approach incorporates the spatial proximity of the triangles and also enables SAH-driven pairing, beneficial for construction of efficient hierarchy. It resembles a standard PLOC iteration [MB18], where the set radius of candidates ordered by the Morton space-filling curve are considered for merging, with two major differences: (1) the *distance()* function returns the value ∞ for candidates without a shared edge, effectively disallowing their merge, and (2) successfully merged pairs are excluded from the subsequent iterations, maintaining only clusters of up to two primitives. The resulting set of leaves is then passed to a standard bottom-up construction algorithm.

4.2. BVH Traversal

Most contemporary massively parallel traversal algorithms are based on the so-called *while-while* design with persistent threads by Aila and Laine [AL09]. One loop evaluates bounding volume intersections, while the other evaluates scene geometry intersections in the BVH leaf nodes. Each of these loops fetches its own data according to the bounding volume and scene geometry representation. The *while-while* design strives to optimize the switches between these two contexts to minimize thread divergence and maximize hardware utilization.

The fundamental difference of our algorithm lies in the fact that we require only a single loop, as due to our UBVH data structure

ALGORITHM 1: Pseudocode of the traversal of a single ray using the N-wide UBVH.

```
ray ← FetchRay()
node ← FetchNode(root_id)
stack ← ∅
result ← INVALID
repeat
    t_mins ← [BIG_FLOAT, ...]
    c_ids ← [INVALID_VALUE, ...]
    node_id ← INVALID_VALUE
    // intersect N-wide node
    for i ← 0 to N do
        result ← Intersect(node.child_sobb[i])
        if intersection found then
            if child is leaf then
                CheckTriangleHit()
            else
                t_mins[i] ← t
                c_ids[i] ← node.child_id[i]
                if child is closest by intersection distance then
                    node_id ← node.child_id[i]
    if node_id is INVALID_VALUE then
        node_id ← StackPop(stack)
    // traversal will continue
    if node_id is valid id then
        node ← FetchNode(node_id)
        Sort(t_mins, c_ids)
        StackPushBackwards(stack, c_ids[i])
until stack ≠ ∅
StoreIntersection(result)
```

we are able to evaluate both intersections equivalently. Hence, we do not need to consider the context switches inherent to traditional BVHs. Our additional ray-triangle intersection check is very fast, does not require additional data to be fetched from memory, and it is executed very rarely, as will be shown in the results. The pseudocode of the single ray traversal is provided in Alg. 1.

The stack-based top-down traversal starts with the BVH root node and continuously evaluates intersections with all child nodes. If the child is a leaf, the additional fast triangle intersection verification test is performed (according to Eq. 4 or Eq. 6) and the valid intersection is stored locally. Additionally, we keep track of the closest non-leaf child as a candidate for the next iteration. If none is found, we try to get the next node from the stack. If the traversal should continue, we immediately retrieve the next node from the main memory, followed by a distance-based sort using a sorting network [Bat68; Gut14], hiding the computation in the latency of the memory access. Finally, the sorted nodes are pushed to stack in far→near order.

Triangle intersection verification. If a leaf node is found, the ray-triangle intersection test is evaluated immediately. Here, it is important to realize that our test is performed on top of a ray-SOBB intersection that necessarily produces two hit points for a successful hit of a non-degenerate SOBB. For the cases of a single triangle and a parallelogram, we neglect the far intersection point due to the thickness of the SOBB being ϵ (see the discussion in Section 3.4). However, doing this also for butterflies might produce

© 2025 The Author(s).
Computer Graphics Forum published by Eurographics and John Wiley & Sons Ltd.

potentially incorrect results due to unexpected back-face culling or self-shadowing. Hence, if we care about the triangle backside and the triangle hit could not be found with the near point, the far point must also be considered. To avoid numerical issues with leaf SOBB boundaries in the *uvw* space, the coordinate corresponding to intersected face of SOBB is explicitly set to 0 or 1 when an intersection with the SOBB is found. This is done using knowledge of the slab planes that yield the nearest and farthest intersection.

Dynamic ray fetch. The behavior of our algorithm is consistent with the findings of Ylitie et al. [YKL17] where dynamic ray fetch provides practically no real benefit for a binary BVH, but wide layouts benefit significantly.

5. Results

The presented method has been prototyped and evaluated in a Vulkan-based framework and its implementation is available online: `https://dcgi.fel.cvut.cz/projects/ubvh/`. All utilized BVHs are constructed on the GPU in the following series of steps:

Baseline BVH. 2-wide AABB BVH is constructed using a single kernel launch variant of the PLOC++ algorithm [BDTD22] and also serves as a starting point for all transformed BVHs. Initial AABB clusters are formed using either scene triangles or triangle pairs for the extended UBVH and the search radius is 16.

Collapsing. When appropriate, BVH is optimized with parallel SAH-based subtree collapsing [MB18], using SAH cost constants $c_i = 2$, $c_t = 3$ for the AABB BVH and $c_i = 2$, $c_t = 4.5$ for the other bounding volumes. The maximum leaf size is set to 8.

Transformation. The corresponding bounding volume transformation algorithm is used to obtain *14*-DOP BVH [KB24], OBB BVH [VEPG23], SOBB BVH [KB25], and UBVH. For SOBB and UBVH cases, we use a *64*-DOP for the intermediate representation during the process.

Layout optimization. Lastly, the hierarchy is packed to a final, traversal layout. We utilize 4-wide layouts, as they consistently outperform their binary counterpart in all our tested scenarios. For conversion, we used parallel conversion introduced by Benthin et al. [BMB*24]. Specialized compression is not applied for any of the methods.

As a reference, we constructed 4-wide AABB BVH, 4-wide *14*-DOP BVH, 4-wide OBB BVH, 4-wide SOBB BVH, and also 4-wide SOBB BVH without subtree collapsing (denoted SOBB*). Our proposed method is evaluated in the form of the 4-wide UBVH and the 4-wide extended UBVH. The topology of SOBB* matches the UBVH; these two data structures differ only in the content of the leaf nodes, i.e. the SOBB shape.

For performance evaluation, we used eight scenes of geometric complexity varying between 1 and 13 million triangles. For each scene, we rendered about eight representative views (including far and close ones) at a resolution of 1920x1080 pixels. We used naive path tracing (i.e., no next event estimation, no Russian roulette) with eight samples per pixel, and we treated all surfaces as Lambertian. We used up to eight bounces per path which created sufficiently incoherent workload for interior-like scenes. The test was performed on a PC with an Intel i9-10900X, 128GB RAM, and an NVIDIA RTX 3080Ti with 12GB of VRAM.

5.1. Measurements

To evaluate the methods tested, we report the average number of primitives per leaf node, the total build time in milliseconds, and total memory requirements for storing the BVH and triangles in an optimized layout. To assess the traversal performance, we report the following statistics: average ray-bounding volume intersection tests per ray, average ray-triangle intersection tests per ray (both consider all rays traced regardless of their type), trace speed of primary rays, trace speed of secondary rays (both in millions of rays per second), and finally the total average time for frame for a given rendering scenario. The results are summarized in Tab. 1 and we discuss them below.

5.1.1. Build times

The construction of our UBVH structures is on average 3.5x slower than construction of 4-wide AABB BVH and 2.1x slower than the one of 4-wide SOBB BVH. The main bottleneck is the transformation of AABBs to SOBBs. Although the algorithm is exactly the same for our UBVH as for the SOBB BVH, unlike the collapsed SOBB BVH we have to process significantly more nodes to maintain full hierarchy with leaves of size one (or two). This is even more pronounced for the uncollapsed SOBB BVH, which has expensive leaf transformation and is on average 1.2x slower than UBVH.

The extended UBVH shows that the overhead of the pair discovery algorithm can be completely recovered already when only a handful of pairs is found (such as for the *Crown* scene), and with higher number of pairs the total build time can be significantly reduced (such as for the *Powerplant* or *Sheep* scenes) compared to the baseline UBVH.

5.1.2. Memory consumption

For conventional BVHs, memory consumption can be divided into two parts. One is consumed by the hierarchy itself, and the other is consumed by the triangles, typically transformed into a representation favorable for the fast and precise intersection test. Our unified representation has a larger memory footprint for bounding volumes than AABB (12 floats for SOBB vs 6 floats for AABB) and is equivalent with OBBs and SOBBs.

For triangles referenced from the traditional BVHs, we use a layout of 12 floats per triangle required for Woop's triangle intersection test [Woo04; AL09]. Triangles in UBVH also use 12 floats per triangle as they are encoded as SOBBs. In extended UBVH a triangle pair is also represented with 12 floats (SOBB) and two bits identifying the particular triangle pair configuration which are stored inside the primitive ID field.

The standard UBVH consumes on average 1.5x more memory than a 4-wide AABB BVH and 1.4x more memory than a 4-wide SOBB BVH. Consumption of an extended UBVH representation is directly tied to the number of triangle pairs in the hierarchy, consuming 0.5x the UBVH memory in an ideal case up to exactly

© 2025 The Author(s).

Table 1: *Performance comparison of UBVH and extended UBVH methods to the reference methods: 4-wide BVHs of AABBs,* 14*-DOPs, OBBs, and SOBBs. The star symbol denotes a BVH construction with no subtree collapsing. The italic font for the average triangle intersections for UBVH methods signifies that, unlike the references, most of the price of these tests is already covered in the bounding volume intersection.*

Scene #triangles	Bounding volume	Leaf size	Build time (ms) ↓		Memory req. (MB) ↓		Avg. tested BVs per ray ↓		Avg. tested △ per ray ↓		Primary (MRps) ↑		Secondary (MRps) ↑		Avg. frame time (ms) ↓	
San Miguel [10.0M]	AABB	2.7	**37**	**(1.00)**	677	(1.00)	141.7	(1.00)	13.3	(1.00)	**618**	**(1.00)**	207	(1.00)	2188	(1.00)
	14-DOP	3.6	53	(1.42)	794	(1.17)	**113.3**	**(0.80)**	10.2	(0.76)	513	(0.83)	161	(0.78)	2810	(1.28)
	OBB	3.7	83	(2.24)	740	(1.09)	133.4	(0.94)	10.6	(0.80)	392	(0.63)	145	(0.70)	3141	(1.44)
	SOBB	3.7	63	(1.69)	740	(1.09)	131.2	(0.93)	11.6	(0.87)	408	(0.66)	195	(0.94)	2374	(1.09)
	SOBB*	1.0	185	(4.97)	1494	(2.21)	142.7	(1.01)	3.7	(0.28)	366	(0.59)	191	(0.92)	2436	(1.11)
	UBVH	1.0	172	(4.62)	1015	(1.50)	142.4	(1.00)	*2.3*	*(0.17)*	506	(0.82)	210	(1.02)	2179	(1.00)
	UBVH ext.	1.5	132	(3.56)	**666**	**(0.98)**	132.1	(0.93)	***2.0***	***(0.15)***	605	(0.98)	**242**	**(1.17)**	**1893**	**(0.87)**
Crown [3.5M]	AABB	2.5	**16**	**(1.00)**	**247**	**(1.00)**	111.5	(1.00)	16.7	(1.00)	**1214**	**(1.00)**	235	(1.00)	1220	(1.00)
	14-DOP	3.4	21	(1.32)	289	(1.17)	**88.1**	**(0.79)**	12.4	(0.74)	902	(0.74)	207	(0.88)	1400	(1.15)
	OBB	3.6	32	(2.03)	269	(1.09)	100.7	(0.90)	6.4	(0.39)	810	(0.67)	235	(1.00)	1254	(1.03)
	SOBB	3.6	24	(1.54)	269	(1.09)	101.8	(0.91)	10.0	(0.60)	812	(0.67)	283	(1.20)	1060	(0.87)
	SOBB*	1.0	60	(3.85)	532	(2.15)	111.8	(1.00)	3.5	(0.21)	708	(0.58)	252	(1.07)	1194	(0.98)
	UBVH	1.0	53	(3.41)	362	(1.47)	111.8	(1.00)	***1.5***	***(0.09)***	863	(0.71)	296	(1.26)	1011	(0.83)
	UBVH ext.	1.1	52	(3.35)	312	(1.26)	108.0	(0.97)	*1.6*	*(0.10)*	988	(0.81)	**326**	**(1.39)**	**915**	**(0.75)**
Bistro ext. [2.8M]	AABB	2.6	**17**	**(1.00)**	**193**	**(1.00)**	158.6	(1.00)	37.5	(1.00)	**634**	**(1.00)**	132	(1.00)	2839	(1.00)
	14-DOP	3.4	23	(1.34)	227	(1.18)	**121.4**	**(0.76)**	22.0	(0.59)	576	(0.91)	122	(0.93)	3069	(1.08)
	OBB	3.6	32	(1.84)	212	(1.10)	134.8	(0.85)	12.6	(0.34)	496	(0.78)	143	(1.08)	2665	(0.94)
	SOBB	3.6	27	(1.55)	212	(1.10)	137.1	(0.86)	17.0	(0.46)	488	(0.77)	174	(1.32)	2220	(0.78)
	SOBB*	1.0	61	(3.56)	421	(2.19)	153.2	(0.97)	6.0	(0.16)	416	(0.66)	162	(1.23)	2387	(0.84)
	UBVH	1.0	59	(3.40)	285	(1.48)	152.9	(0.96)	***2.1***	***(0.06)***	552	(0.87)	193	(1.46)	1993	(0.70)
	UBVH ext.	1.4	50	(2.87)	202	(1.05)	153.9	(0.97)	*2.3*	*(0.06)*	578	(0.91)	**196**	**(1.49)**	**1965**	**(0.69)**
Mammoth [8.8M]	AABB	2.3	**35**	**(1.00)**	624	(1.00)	67.4	(1.00)	2.8	(1.00)	1360	(1.00)	293	(1.00)	264	(1.00)
	14-DOP	3.2	51	(1.46)	745	(1.19)	**50.9**	**(0.75)**	2.4	(0.85)	1146	(0.84)	273	(0.93)	291	(1.10)
	OBB	3.3	81	(2.35)	692	(1.11)	56.9	(0.84)	4.0	(1.41)	920	(0.68)	234	(0.80)	345	(1.31)
	SOBB	3.3	58	(1.68)	692	(1.11)	56.2	(0.83)	2.5	(0.89)	1203	(0.88)	336	(1.15)	247	(0.94)
	SOBB*	1.0	137	(3.96)	1310	(2.10)	58.8	(0.87)	0.9	(0.31)	1186	(0.87)	342	(1.17)	246	(0.93)
	UBVH	1.0	127	(3.66)	888	(1.42)	58.8	(0.87)	***0.6***	***(0.22)***	1446	(1.06)	416	(1.42)	202	(0.77)
	UBVH ext.	1.7	92	(2.65)	**536**	**(0.86)**	58.5	(0.87)	*0.7*	*(0.26)*	**1508**	**(1.11)**	**447**	**(1.52)**	**190**	**(0.72)**
Powerplant [12.8M]	AABB	2.6	**40**	**(1.00)**	880	(1.00)	144.3	(1.00)	26.7	(1.00)	**784**	**(1.00)**	178	(1.00)	1736	(1.00)
	14-DOP	3.7	58	(1.45)	1016	(1.16)	**127.4**	**(0.88)**	12.7	(0.48)	546	(0.70)	150	(0.84)	2086	(1.20)
	OBB	3.9	87	(2.15)	944	(1.07)	157.4	(1.09)	16.5	(0.62)	441	(0.56)	140	(0.79)	2243	(1.29)
	SOBB	3.9	75	(1.87)	944	(1.07)	137.5	(0.95)	14.6	(0.55)	568	(0.72)	203	(1.14)	1542	(0.89)
	SOBB*	1.0	223	(5.53)	1928	(2.19)	150.4	(1.04)	5.2	(0.19)	445	(0.57)	176	(0.99)	1814	(1.04)
	UBVH	1.0	208	(5.17)	1315	(1.50)	150.0	(1.04)	*2.5*	*(0.10)*	582	(0.74)	244	(1.37)	1318	(0.76)
	UBVH ext.	1.6	161	(4.00)	**840**	**(0.95)**	151.2	(1.05)	***2.3***	***(0.09)***	558	(0.71)	**276**	**(1.55)**	**1185**	**(0.68)**
Hairball [2.9M]	AABB	4.4	**11**	**(1.00)**	**172**	**(1.00)**	165.4	(1.00)	45.8	(1.00)	523	(1.00)	89	(1.00)	3494	(1.00)
	14-DOP	5.1	15	(1.38)	203	(1.18)	**130.6**	**(0.79)**	26.7	(0.58)	439	(0.84)	107	(1.20)	2975	(0.85)
	OBB	5.2	22	(1.99)	194	(1.13)	147.6	(0.89)	12.7	(0.28)	411	(0.79)	128	(1.43)	2530	(0.72)
	SOBB	5.2	18	(1.64)	194	(1.13)	147.9	(0.89)	19.8	(0.43)	432	(0.83)	147	(1.65)	2214	(0.63)
	SOBB*	1.0	46	(4.15)	427	(2.48)	165.8	(1.00)	4.3	(0.09)	326	(0.62)	136	(1.52)	2452	(0.70)
	UBVH	1.0	43	(3.91)	288	(1.68)	165.7	(1.00)	*2.1*	*(0.05)*	535	(1.02)	160	(1.79)	2016	(0.58)
	UBVH ext.	1.7	29	(2.63)	180	(1.05)	159.7	(0.97)	***2.0***	***(0.04)***	**548**	**(1.05)**	**177**	**(1.98)**	**1845**	**(0.53)**
Bistro int. [1.0M]	AABB	2.5	**5.2**	**(1.00)**	72	(1.00)	123.8	(1.00)	63.5	(1.00)	**859**	**(1.00)**	163	(1.00)	2963	(1.00)
	14-DOP	3.5	7.2	(1.39)	84	(1.16)	**96.8**	**(0.78)**	34.6	(0.55)	554	(0.64)	144	(0.88)	3400	(1.15)
	OBB	3.6	9.8	(1.91)	78	(1.09)	100.2	(0.81)	8.7	(0.14)	570	(0.66)	227	(1.39)	2197	(0.74)
	SOBB	3.6	8.8	(1.71)	78	(1.09)	102.6	(0.83)	14.9	(0.23)	636	(0.74)	256	(1.56)	1955	(0.66)
	SOBB*	1.0	21	(4.17)	156	(2.17)	116.9	(0.94)	7.6	(0.12)	520	(0.60)	224	(1.37)	2243	(0.76)
	UBVH	1.0	20	(3.92)	106	(1.47)	116.4	(0.94)	***2.0***	***(0.03)***	623	(0.73)	378	(2.31)	1362	(0.46)
	UBVH ext.	1.5	16	(3.18)	**70**	**(0.97)**	108.6	(0.88)	*2.1*	*(0.03)*	722	(0.84)	**432**	**(2.64)**	**1192**	**(0.40)**
Sheep [10.7M]	AABB	4.3	**52**	**(1.00)**	**644**	**(1.00)**	**245.1**	**(1.00)**	290.9	(1.00)	13	(1.00)	3.8	(1.00)	61363	(1.00)
	14-DOP	5.0	68	(1.29)	760	(1.18)	331.8	(1.35)	258.0	(0.89)	32	(2.46)	11	(2.84)	21863	(0.36)
	OBB	4.8	106	(2.02)	731	(1.14)	249.4	(1.02)	135.1	(0.46)	78	(5.97)	32	(8.44)	7522	(0.12)
	SOBB	4.8	80	(1.52)	738	(1.15)	249.1	(1.02)	121.3	(0.42)	100	(7.65)	37	(9.72)	6455	(0.11)
	SOBB*	1.0	187	(3.57)	1602	(2.49)	380.0	(1.55)	85.0	(0.29)	48	(3.67)	24	(6.20)	10454	(0.17)
	UBVH	1.0	172	(3.29)	1086	(1.69)	379.8	(1.55)	*3.4*	*(0.01)*	128	(9.70)	34	(8.94)	6817	(0.11)
	UBVH ext.	1.5	124	(2.38)	720	(1.12)	334.3	(1.36)	***3.3***	***(0.01)***	**168**	**(12.76)**	**45**	**(11.82)**	**5171**	**(0.08)**

© 2025 The Author(s).

the same amount in the worst case. Our extended UBVH pairing achieved about 1.5x reduction in memory consumption compared to standard UBVH and is roughly equivalent to AABB BVH, despite the larger memory footprint of the bounding volume representation. Furthermore, with the exception of an underwhelming pairing in the *Crown* scene, the extended UBVH consumes roughly equivalent amount of memory as the AABB BVH and less than other reference BVHs (using 12 float triangle representaion for fast intersection test [AL09]).

5.1.3. Average number of intersection tests

Due to the tight fit of SOBBs, our UBVH on average requires approximately the same number of the bounding volume intersections compared to the baseline AABB, despite the fact that the baseline BVH with collapsed subtrees is shallower. An exception is the *Sheep* scene, where the subtree collapsing significantly reduces the number of nodes in the tree. However, this metric should be considered together with the average ray-triangle intersections.

In UBVH, most of the work necessary for the ray-triangle intersection is already covered by the ray-SOBB intersection. The ray-triangle intersection verification itself is cheap and much more efficient than the ray-triangle intersection used in other methods. The practical cost of the UBVH ray-triangle tests is reduced even more as they require no additional data to be fetched. Thus, the numbers in the related column in Tab. 1 are in italics to indicate that although they represent the same idea, they are not directly comparable to the rest of the values. In addition, the deeper hierarchy of UBVH with leaf bounding volumes directly fit to triangles leads to a very low number of ray-triangle intersection verifications.

5.1.4. Ray tracing speed

We first discuss results for all scenes excluding the *Sheep* scene, which has a dedicated section below. Our results are consistent with findings for SOBB BVHs [KB25], that tight bounding volumes with more data and more expensive intersection test tend to degrade the performance of highly coherent primary rays, unless the structure of the scene is complex enough (e.g., the *Hairball* and *Mammoth* scenes). Still, UBVH is, on average, 1.1x faster and extended UBVH is about 1.2x faster than the 4-wide SOBB BVH for primary rays.

On the other hand, incoherent secondary rays benefit significantly from reduced overlaps of tight bounding volumes, with UBVH being on average about 1.5x and extended UBVH about 1.7x faster than the baseline AABB BVH. UBVH also outperforms SOBB BVH with 1.2x and 1.3x faster trace speeds of secondary rays, respectively.

It is known that for path tracing, majority of time is spent on the traversal of secondary rays (over 90% in most of our test scenarios). Thus, the overall rendering performance is strongly correlated with secondary rays performance. The UBVH resulted in the best frame times for all scenes tested.

Sheep scene. Scenes such as *Sheep* are a common outlier used specifically in the research of tight bounding volumes [WMZ*20; VEPG23; KB24; KB25]. Here, tubular shapes tesselated from curves simulating fur form a dense, massively overlapping structure of long, thin, and arbitrarily oriented primitives. In addition to the utilization of tightly fitting SOBBs, the great success of our extended UBVH is due to a huge number of triangle pairs in the form of parallelograms and butterflies (depending on the algorithm) that can be exploited by our method.

5.1.5. Triangle pairs

The extended UBVH results presented in Tab. 1 were obtained with triangle pairs discovered by 4 iterations of the PLOC-based pairing algorithm with a radius of 8, empirically proven to be reasonable default values, although the parameters could be tuned adaptively per scene for best results. The success rate and pair distribution of the settings chosen is expressed as the reduction in the total number of BVH nodes in Fig. 6.

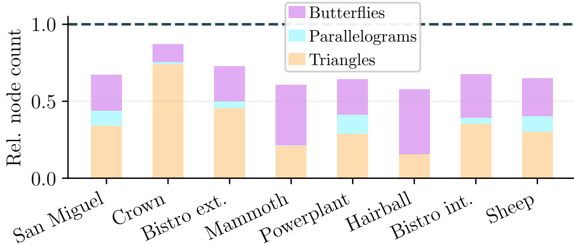


Figure 6: *Relative reduction of primitives using our PLOC-based algorithm for discovery of triangle pairs with a common edge.*

We also tested the greedy pairing algorithm as described in 4.1.2 and report the results in Tab 2. In the top part, we provide relative average values over the scenes except the *Sheep*, which is handled separately. Despite the fact that the algorithm is simpler and faster on its own, it also typically discovers fewer pairs, and consequently, the total build time is overwhelmed by the transformation cost of more nodes, leading also to increased memory requirements and slower trace times.

The exception is the *Sheep* scene, where due to the shape and algorithmically produced topological layout allows the greedy algorithm to discover more parallelogram pairs faster, leading to smaller memory footprint and increased traversal performance.

Table 2: *Relative comparison of the PLOC-based and greedy triangle pairing algorithms introduced in Sec. 4.1.2. Greedy algorithm performs on average worse, however with favorable scene topology it can outperform the PLOC, as proven by the* Sheep *scene.*

Pairing method	Leaf size	Build time	Memory	Secondary RT
avg. w/o Sheep				
PLOC pairs	1.50	1.00	1.00	1.00
Greedy pairs	1.21	1.21	1.23	0.95
Sheep				
PLOC pairs	1.50	1.00	1.00	1.00
Greedy pairs	1.70	0.87	0.92	1.09

© 2025 The Author(s).
Computer Graphics Forum published by Eurographics and John Wiley & Sons Ltd.

5.2. Discussion and Limitations

Traversal. Our UBVH traversal pipeline is limited by global memory access latency. This is partially hidden by the full sorting of children nodes, but another meaningful task, such as stack sanitation, could be investigated. We currently fetch the entire uncompressed wide BVH node at once, creating pressure on the register allocation. Our experiments with 8-wide nodes in this fashion were strongly hindered. Other approaches could attempt to efficiently interleave partial node fetches and intersection computations with precomputed child node ordering, for example, as proposed by Ylitie et al. [YKL17]. Additionally, a shared memory could be utilized to partially cover the traversal stack for faster memory access.

Numerical stability. Ray tracing, as any other computation in finite precision, is prone to floating point errors. We use world-space ϵ-thresholds for computing SOBB bounds and *uvw* parametric space ϵ-thresholds for the triangle intersection verification. Although we do not observe precision related artifacts in tested scenes, to provably achieve watertightness of the SOBB-based triangle intersection, our method would need to be extended as done by Woop et al. for triangles [WBW13].

Compression. State-of-the-art GPU-oriented ray tracing implementations, such as the work of Ylitie et al. [YKL17], utilize axis-aligned bounding boxes with reduced precision organized in a wide compressed BVH node layout. Due to unified representation of bounding volumes and scene geometry, traditional lossy compression methods such as bounding volume quantization are not trivially applicable to UBVH, as they will also be applied to the geometry itself. As we do not use compressed layout for any of the evaluated methods, a comparison with such an approach is missing. We acknowledge that in specific cases it might outperform our implementation.

6. Conclusion and Future Work

We proposed a unified representation of bounding volumes and triangular geometry for BVHs based on skewed oriented bounding boxes. This unified representation allows for a simplified design of the BVH traversal algorithm for ray tracing, leading to a considerable acceleration of traversal of incoherent rays. We also presented a method for efficient representation of triangle pairs, leading to reduced memory consumption and further accelerating ray tracing. For tracing incoherent rays, the UBVH methods outperformed traditional AABB BVHs as well as SOBB, 14-DOP, and OBB BVHs in all tested scenes.

There are numerous possible extensions of our work. Designing a compressed SOBB representation and improving the algorithm for determining triangle pairs would yield an even lower memory footprint and faster trace times. Optimizing BVH topology directly for the SOBBs would likely further improve the traversal performance. We also believe that our method could serve as a basis for designing a novel hardware architecture for ray tracing polygonal scenes.

Acknowledgments

This work was supported by the Research Center for Informatics (CZ.02.1.01/0.0/0.0/16_019/0000765) and by the Grant Agency of the Czech Technical University in Prague (SGS25/150/OHK3/3T/13). The *Crown* scene is courtesy of Martin Lubich, the *Sheep* scene is part of the Blender Foundation's demo scene repository, and the *Mammoth* scene is part of the Smithsonian Institution digitized 3D collection. The rest of the scenes come from Morgan McGuire's Computer Graphics Archive [McG17]. Open Access funding enabled and organized by CzechELib.

References

[AKL13] AILA, TIMO, KARRAS, TERO, and LAINE, SAMULI. "On quality metrics of bounding volume hierarchies". *Proceedings of the 5th High-Performance Graphics Conference*. HPG '13. Anaheim, California: Association for Computing Machinery, 2013, 101–107 2.

[AL09] AILA, TIMO and LAINE, SAMULI. "Understanding the Efficiency of Ray Traversal on GPUs". *Proceedings of the Conference on High Performance Graphics 2009*. HPG '09. New Orleans, Louisiana: Association for Computing Machinery, 2009, 145–149 5, 6, 8.

[Bat68] BATCHER, KENNETH E. "Sorting networks and their applications". *Proceedings of the April 30–May 2, 1968, spring joint computer conference*. 1968, 307–314 5.

[BDTD22] BENTHIN, CARSTEN, DRABINSKI, RADOSLAW, TESSARI, LORENZO, and DITTEBRANDT, ADDIS. "PLOC++: Parallel Locally-Ordered Clustering for Bounding Volume Hierarchy Construction Revisited". *Proc. ACM Comput. Graph. Interact. Tech.* 5.3 (July 2022) 2, 6.

[BK18] BINDER, NIKOLAUS and KELLER, ALEXANDER. "Fast, high precision ray/fiber intersection using tight, disjoint bounding volumes". *ACM SIGGRAPH 2018 Talks*. SIGGRAPH '18. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2018 2.

[BMB*24] BENTHIN, CARSTEN, MEISTER, DANIEL, BARCZAK, JOSHUA, et al. "H-PLOC: Hierarchical Parallel Locally-Ordered Clustering for Bounding Volume Hierarchy Construction". *Proc. ACM Comput. Graph. Interact. Tech.* 7.3 (Aug. 2024) 2, 3, 6.

[CB17] CALDERON, STÉPHANE and BOUBEKEUR, TAMY. "Bounding proxies for shape approximation". *ACM Trans. Graph.* 36.4 (2017), 57–1 2.

[CGM11] CHANG, CHIA-TCHE, GORISSEN, BASTIEN, and MELCHIOR, SAMUEL. "Fast oriented bounding box optimization on the rotation group $SO(3,\mathcal{R})$". *ACM Transactions on Graphics (TOG)* 30.5 (2011), 1–16 2.

[EBM12] EISEMANN, MARTIN, BAUSZAT, PABLO, and MAGNOR, MARCUS. *Implicit Object Space Partitioning: The No-Memory BVH*. Tech. rep. 16. Computer Graphics Lab, TU Braunschweig, Jan. 2012 2.

[FLPE16] FUETTERLING, V., LOJEWSKI, C., PFREUNDT, F.-J., and EBERT, A. "Parallel spatial splits in bounding volume hierarchies". *Proceedings of the 16th Eurographics Symposium on Parallel Graphics and Visualization*. EGPGV '16. Groningen, The Netherlands: Eurographics Association, 2016, 21–30 2.

[Got00] GOTTSCHALK, STEFAN ARIC. *Collision queries using oriented bounding boxes*. The University of North Carolina at Chapel Hill, 2000 2.

[GS87] GOLDSMITH, JEFFREY and SALMON, JOHN. "Automatic Creation of Object Hierarchies for Ray Tracing". *IEEE Computer Graphics and Applications* 7.5 (May 1987), 14–20 2.

[Gut14] GUTHE, MICHAEL. "Latency Considerations of Depth-first GPU Ray Tracing". *Eurographics 2014 - Short Papers*. Ed. by GALIN, ERIC and WAND, MICHAEL. The Eurographics Association, 2014 5.

[KA13] KARRAS, TERO and AILA, TIMO. "Fast parallel construction of high-quality bounding volume hierarchies". HPG '13. Anaheim, California: Association for Computing Machinery, 2013, 89–99 2.

© 2025 The Author(s).

[KB24] KÁČERIK, MARTIN and BITTNER, JIŘÍ. "SAH-Optimized k-DOP Hierarchies for Ray Tracing". *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 7.3 (2024), 1–16 2, 6, 8.

[KB25] KÁČERIK, M. and BITTNER, J. "SOBB: Skewed Oriented Bounding Boxes for Ray Tracing". *Computer Graphics Forum* 44.2 (2025) 2–6, 8.

[KHM*98] KLOSOWSKI, JAMES T, HELD, MARTIN, MITCHELL, JOSEPH SB, et al. "Efficient collision detection using bounding volume hierarchies of k-DOPs". *IEEE transactions on Visualization and Computer Graphics* 4.1 (1998), 21–36 2.

[KZ97] KONEČNÝ, PETR and ZIKAN, KAREL. "Lower bound of distance in 3d". *Proceedings Winter School of Computer Graphics(WCSG '97)* 3 (1997), 640–649 2.

[LK11] LARSSON, THOMAS and KÄLLBERG, LINUS. "Fast computation of tight-fitting oriented bounding boxes". *Game Engine Gems 2* (2011), 1 2.

[LYM07] LAUTERBACH, CHRISTIAN, YOON, SUNG-EUI, and MANOCHA, DINESH. "Ray-Strips: A Compact Mesh Representation for Interactive Ray Tracing". *2007 IEEE Symposium on Interactive Ray Tracing*. 2007, 19–26 2.

[LYTM08] LAUTERBACH, CHRISTIAN, YOON, SUNG-EUI, TANG, MING, and MANOCHA, DINESH. "ReduceM: Interactive and Memory Efficient Ray Tracing of Large Models". *Computer Graphics Forum* 27.4 (2008), 1313–1321 2.

[MB18] MEISTER, DANIEL and BITTNER, JIŘÍ. "Parallel locally-ordered clustering for bounding volume hierarchy construction". *IEEE transactions on visualization and computer graphics* 24.3 (2018), 1345–1353 2, 5, 6.

[MB22] MEISTER, DANIEL and BITTNER, JIŘÍ. "Performance Comparison of Bounding Volume Hierarchies for GPU Ray Tracing". *Journal of Computer Graphics Techniques (JCGT)* 11.4 (Oct. 2022), 1–19 2.

[McG17] MCGUIRE, MORGAN. *Computer Graphics Archive*. July 2017 9.

[MOB*21] MEISTER, DANIEL, OGAKI, SHINJI, BENTHIN, CARSTEN, et al. "A survey on bounding volume hierarchies for ray tracing". *Computer Graphics Forum*. Vol. 40. 2. Wiley Online Library. 2021, 683–712 2.

[ORo85] O'ROURKE, JOSEPH. "Finding minimal enclosing boxes". *International journal of computer & information sciences* 14 (1985), 183–199 2.

[PK87] PULLEYBLANK, RON and KAPENGA, JOHN. "The Feasibility of a VLSI Chip for Ray Tracing Bicubic Patches". *IEEE Computer Graphics and Applications* 7.3 (1987), 33–44 2.

[SE10] SEGOVIA, BENJAMIN and ERNST, MANFRED. "Memory efficient ray tracing with hierarchical mesh quantization". *Proceedings of Graphics Interface 2010*. GI '10. Ottawa, Ontario, Canada: Canadian Information Processing Society, 2010, 153–160 2.

[SVCM23] SABINO, RODOLFO, VIDAL, CRETO AUGUSTO, CAVALCANTE-NETO, JOAQUIM BENTO, and MAIA, JOSÉ GILVAN RODRIGUES. "Building Oriented Bounding Boxes by the intermediate use of ODOPs". *Computers & Graphics* 116 (2023), 251–261 2.

[VEPG23] VITSAS, NICK, EVANGELOU, IORDANIS, PAPAIOANNOU, GEORGIOS, and GKARAVELIS, ANASTASIOS. "Parallel Transformation of Bounding Volume Hierarchies into Oriented Bounding Box Trees". *Computer Graphics Forum* 42.2 (2023). Ed. by MYSZKOWSKI Karol-Niessner, MATTHIAS, 245–25410 pages 2, 3, 6, 8.

[VWB19] VAIDYANATHAN, K., WOOP, S., and BENTHIN, C. "Wide BVH traversal with a short stack". *Proceedings of the Conference on High-Performance Graphics*. HPG '19. Strasbourg, France: Eurographics Association, 2019, 15–19 2.

[WBB08] WALD, INGO, BENTHIN, CARSTEN, and BOULOS, SOLOMON. "Getting rid of packets - Efficient SIMD single-ray traversal using multi-branching BVHs -". *2008 IEEE Symposium on Interactive Ray Tracing*. 2008, 49–57 2.

[WBW13] WOOP, SVEN, BENTHIN, CARSTEN, and WALD, INGO. "Watertight ray/triangle intersection". *Journal of Computer Graphics Techniques (JCGT)* 2.1 (2013), 65–82 9.

[WMZ*20] WALD, INGO, MORRICAL, NATE, ZELLMANN, STEFAN, et al. "Using Hardware Ray Transforms to Accelerate Ray/Primitive Intersections for Long, Thin Primitive Types". *Proc. ACM Comput. Graph. Interact. Tech.* 3.2 (Aug. 2020) 8.

[Woo04] WOOP, SVEN. "A Programmable Hardware Architecture for Real-time Ray Tracing of Coherent Dynamic Scenes". (Apr. 2004) 6.

[WWB*14] WALD, INGO, WOOP, SVEN, BENTHIN, CARSTEN, et al. "Embree: a kernel framework for efficient CPU ray tracing". *ACM Trans. Graph.* 33.4 (July 2014) 3.

[YKL17] YLITIE, HENRI, KARRAS, TERO, and LAINE, SAMULI. "Efficient incoherent ray traversal on GPUs through compressed wide BVHs". *Proceedings of High Performance Graphics*. HPG '17. Los Angeles, California: Association for Computing Machinery, 2017 2, 6, 9.

© 2025 The Author(s).