Eurographics Symposium on Rendering - Experimental Ideas & Implementations (2017)
P. Sander and M. Zwicker (Editors)

# *k*-d Tree Construction Designed for Motion Blur

X. Yang[1], Q. Liu[2], B.C. Yin[1], Q. Zhang[1], D.S. Zhou[3], X.P. Wei[1]

[1] Dalian University of Technology, [2] Shanghai Jiao Tong University, [3] Dalian University

**Abstract**

*We present a k-d tree construction algorithm designed to accelerate rendering of scenes with motion blur, in application scenarios where a k-d tree is either required or desired. Our associated data structure focuses on capturing incoherent motion within the nodes of a k-d tree and improves both data structure quality and efficiency over previous methods. At build-time stage, we tracks primitives with motion that is significantly distinct from other primitives within the node, guarantee valid node references and the correctness of the data structure via primitive duplication heuristic and propagation rules. Our experiments with this hierarchy show artifact-free motion-blur rendering using a k-d tree, and demonstrate improvements against a traditional BVH with interpolation and a MSBVH structure designed to handle moving primitives, particularly in render time.*

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing.

## 1. Introduction

Motion blur [ETH*09] effects occur naturally in photography due to camera or object motion with non-zero shutter times. However, rendering accurate motion blur effects significantly increases complexity due to either extra geometry, extra sampling, or both. This additional complexity makes motion-blur computation difficult in real-time applications, so realistic motion-blur effects has traditionally been limited to offline renderers such as ray tracers.

Most ray tracers use acceleration structures that dramatically decrease ray-object intersection computations when compared to a non-hierarchical geometric representation. In general a *k*-d tree [PGSS06, CKL*10] provides better report times for viewing and shadow rays intersection tests than other common acceleration structures, such as BVH, meanwhile supports efficient packet tracing and frustum traversal [WMG*09]. Besides, *k*-d tree is very promising in a variety of graphics applications, including nearest photon queries in photon mapping, and nearest neighbor search in point cloud modeling and particle-based fluid simulation.

While we expect nearby primitives in animated objects to behave coherently and have similar motion vectors, unfortunately, this is often not the case. Primitives can travel in different directions and move across the split planes, causing a need to frequently rebuild the tree hierarchy. As an object moves, the primitives may move outside the node or across the split plane between two child nodes, both of which will invalidate the references in the node. Traversing a conventionally-constructed *k*-d tree in the presence of motion blur may result in either artifacts or significantly more traversals and intersections, substantially reducing the efficiency of the *k*-d tree.

In this paper, we propose a new *k*-d tree construction algorithm, the "Motion Blur *k*-d" tree (MBKD). The MBKD is based on the *k*-d tree and effectively targets general motion blur. We introduce one new kind of virtual node, which we call a *BP_node*, that collects all the primitives whose incoherent motion would cause inefficiencies when generating the tree; then, we design a recursive propagation algorithm that propagates duplicate primitives in the *BP_nodes* down the tree. Our experiments with this hierarchy show artifact-free motion-blur rendering using a *k*-d tree, and demonstrate improvements against a traditional BVH with interpolation and a MSBVH structure designed to handle moving primitives, particularly in render time.

## 2. Related Work

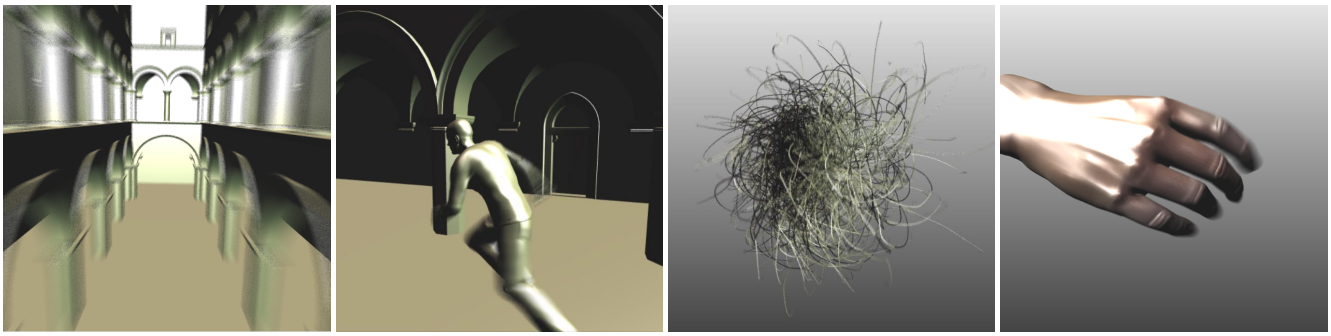
### 2.1. Acceleration Structures

High-performance ray tracing requires acceleration structures to reduce the number of unnecessary ray intersections. Havran [Hav07] and more recently Karras [Kar12], in the context of parallel data structures, provide excellent summaries of the strengths and weaknesses of various acceleration structures.

A Bounding Volume Hierarchy (BVH) is an object hierarchy where each tree node stores a bounding volume for its subtree's geometry and leaves reference the primitives. Because the primitives in BVHs can usually only be stored in one leaf, some scenes may



DOI: 10.2312/sre.20171200



(a) Sponza (b) Ben (c) Hairball (d) Hand

**Figure 1:** *Rendering motion-blur involves tracking primitives as they move during a camera's shutter interval at sample* = 16 *with 512*512 resolution using Intel Core i7 4700MQ processor. Conventional acceleration structures like a k-d tree struggle to handle such primitives, resulting in artifacts or loss of performance. We introduce MBKD, an acceleration structure which addresses these issues by modifying a conventional k-d tree.*

have overlapping bounding boxes. Overlaps increase the number of traversals and ray-primitive operations, affecting rendering performance. While recent research addresses this problem [WK06, DK08, SFD09], very little work actually addresses the complication of motion blur.

Compared with other acceleration structures, *k*-d trees generally obtain faster ray traversal calculation and adapt better to highly varying geometric densities by introducing spatial splits, but take substantially longer time to construct. Objects in a *k*-d tree are grouped into nodes by a decision procedure such as the surface area heuristic (SAH) [CKL*10], which enables optimization of computation costs between tree traversal steps and triangle intersections. The extensive literature on fast computation of *k*-d trees that are optimized for different applications mostly targets optimized construction algorithms, traversal algorithms, and intersection algorithms on both the CPU [CKL*10, SSK07] and GPU [WZL11, ZHWG08]. However, techniques that efficiently handle moving primitives for a *k*-d tree organization are less well established.

### 2.2. Motion Blur

Fast rendering of motion blur has been an active area of research in computer graphics. Most mature rendering systems approximate motion and defocus blur with image-space post-processing. Recent research utilizes rasterization-based methods [MCH*11, VTS*12], or a newer and programmable motion effect [HHRZ12, SSBG10], but we focus here on the intersection of ray tracing and motion blur.

Stochastic ray tracing [CPC84] is an important method for photorealistic motion-blur-effect simulation. The traditional method for motion blur extends the bounding box of the enclosed primitive in the time dimension. The actual bounding boxes and geometry used during ray traversal are then determined by linear interpolation [CFLB06, ZHR*09]. Glassner [Gla88] developed hybrid trees, which use both concepts of spatial and object hierarchies. Olsson [Ols00] proposed 4D *k*-d trees for time-continuous animations. McGuire et al. [MESL10] presented a hybrid algorithm on the GPU for rendering approximate motion and defocus blur with precise stochastic visibility evaluation. Gribel et al.'s visibility algorithm [GBAM11] allows rendering motion blur with per-pixel anti-aliasing.

Hou et al. [HQL*10] presented a BVH based on 4D OBB hypertrapezoids that project into 3D object bounding boxes in spatial dimensions to get tighter bounding boxes. Grünschloß et al. [GSNK11] proposed the MSBVH data structure, which is based on the SBVH [SFD09] to support efficient ray tracing of motion blur by interpolating node bounding boxes, while at the same time reducing node overlap using spatial splits as introduced with the SBVH. They used a clipping method to generate the leaf clipping box, propagated the bounds up the hierarchy, then interpolated the node bounding box as in traditional BVH interpolation.

Interpolation methods work well on some acceleration data structures, but cannot obtain correct results on *k*-d trees. Even Bkd-Tree [PAAV03] is the hybrids of *k*-d trees and BVH to interpolate for motion blur, its essential structure is based on BVH.

## 3. Motion-Blur *k*-d Tree

### 3.1. MBKD Construction

This work focuses specifically on a *k*-d tree construction algorithm designed for efficient handling of motion blur, including the difficult case of incoherent motion. We define an incoherent primitive as a primitive that, due to its motion, move across the split plane to intersect another node and thus invalids the tree hierarchy. Our method is based on the idea that we should relocate primitives, when appropriate with a minimized number of redundant primitive references, to enable fast and correct rendering. Our data structure preserves the advantages of the *k*-d tree while adding the benefit of efficient traversal for moving primitives.

#### 3.1.1. Duplication

Nodes in MBKD can be viewed as a traditional *k*-d tree node, say *Normal_node* with a virtual *BP_node* attached, except the root node, which is only considered as a *Normal_node*. Primitives that do

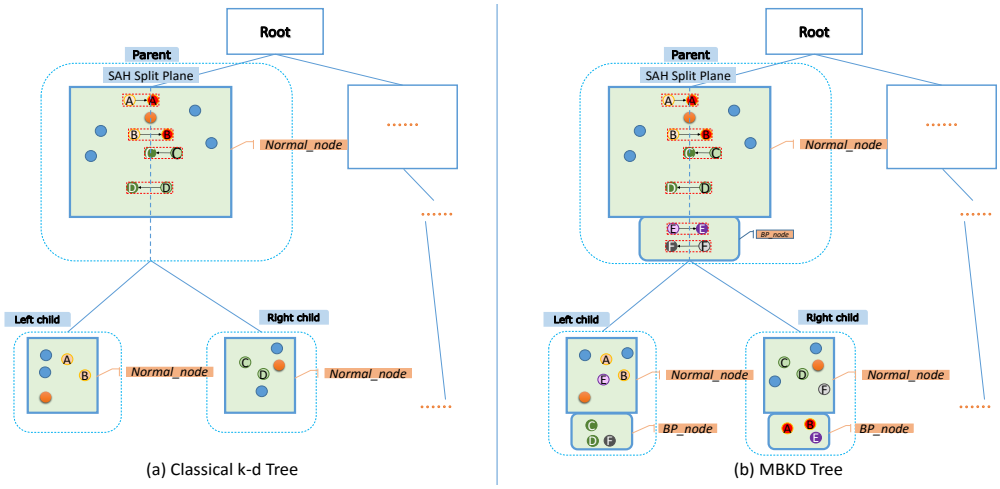



**Figure 2:** *Comparison of the structures of the classical k-d tree and MBKD tree. In a classical k-d tree, moving primitives (A, B, C and D in figure (a)) would be assigned to only one child and do not contribute to motion blur. Instead, our MBKD tree uses a BP_node to store these primitives, as shown in figure (b). For primitives A and B, they are duplicated into the right child's BP_node, and C and D are duplicated into the left child's BP_node. The E and F in the BP_node of the parent will continue to be propagated into the next child nodes. The primitive E is placed in the left child Normal_node, and stored in the right child BP_node for the duplication based on its initial position and moving route. The primitive F is processed in a similar way. The static primitives of the parent are placed in the Normal_nodes into two child nodes, as the traditional k-d tree construction.*

not move across the split plane are only stored in a *Normal_node* and not duplicated. The *BP_node* only stores duplicated primitives crossing the split plane; it is not a spatial node like the *Normal_node* but instead more of a virtual node.

The first step is to detect all the primitives with incoherent motion within the *Normal_nodes* of a *k*-d tree, that move across a split plane intersect another node and worsen the tree hierarchy. Thus, we duplicate these destructive primitives across the split plane during the time shutter into appropriate *BP_nodes*, to maintain a valid tree hierarchy, and build a *BP_node* for storing them. If no primitives in a *Normal_node* cross the split plane, we generate no *BP_node* for that *Normal_node*.

Specifically, we compute the split plane for a parent node using SAH at $t = 0.5$. If a triangle in a *Normal_node* with its time-extended bounding box crosses this split plane, we know this triangle must be duplicated in some child *BP_node*. Figure 3 shows the 4 moving-primitive cases we consider that require special handling to properly compute motion blur. Primitives in motion in a *Normal_node* may cross the split plane at different times, some primitives may be only contained in one child at $t = 0.5$, assuming which is left child, and it may move into right child at some time point during the time shutter. We first assign that primitive into a child *Normal_node* like a traditional *k*-d tree node construction. Then, we duplicate it into new built *BP_node* attached of the other child *Normal_node*, where the affiliated *Normal_node* of the *BP_node* initially does not contain this duplicated primitive. We collect all of these primitives featured with the description above into the the corresponding *BP_nodes* during this movement, and the other rest of the primitives of the parent node are assigned using the SAH into two child *Normal_nodes*. If a primitive is keeping staying across the split plane during the whole time shutter, it will only be assigned into both child *Normal_nodes* just as in the classical *k*-d tree construction. There are also another 4 moving-primitive cases, which are analogous but move begin with the right child.

It should be pointed that, when we continue to select the split plane for subdivision, *BP_node* does not participate in the split plane selection, SAH cost is computed among the primitives from the *Normal_node*, not including the primitives in the *BP_node*, in order to decrease the sorting time and prevent difficult primitives from confusing the split plane selection. While detecting the primitives that cross the split plane, some extended bounding boxes of the primitives also may cross the boundary of the current node. In order to make each ray travel down to the leaf node through MBKD, and intersect correctly with the primitives probably crossing the boundary of the node, we construct an extended bounding box for the root node, which contains all the possible positions for the primitives during the time shutter. Then, when further splitting the node, we can only consider the primitives related to the selected split plane, instead do not consider other primitives around the boundary, and make sure the primitives always can be contained in a node during the time shutter.

Selecting the split plane has a significant effect on the quality of the resulting tree. We build the hierarchy from top to bottom, exploit temporal coherence in motion, first consider the scene pose at $t = 0.5$ during the initial construction. Wald et al. [Wal07] mentioned that the topology of this pose can work reasonably well for temporally-close scene poses. Although the case to which Wald et al. refers is not aimed at incoherent motion, it still works well for most tree nodes. While it makes intuitive sense to construct the initial *k*-d tree at the center of the time interval rather than

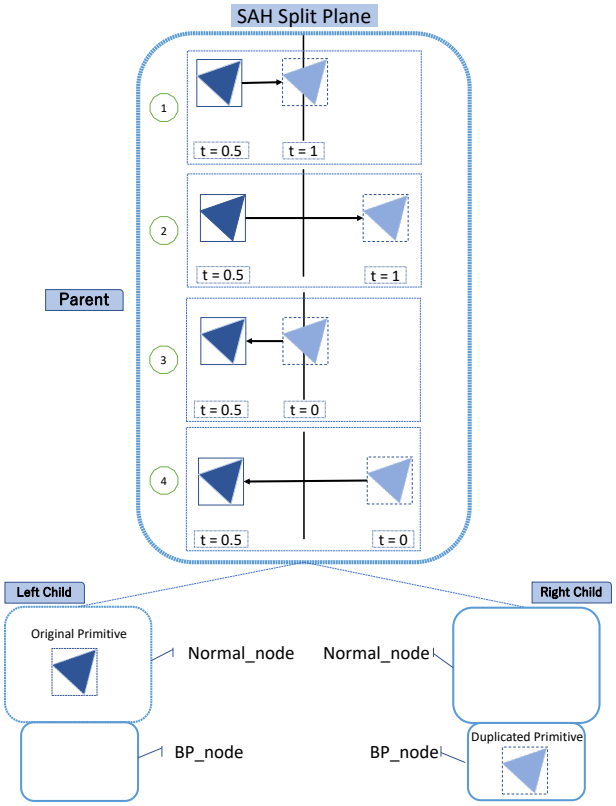



**Figure 3:** *Duplication Heuristics. Here, we focus on the primitives in the Normal_node of a parent, and identify 8 cases contributing to motion blur. In cases 1, 2, 3, and 4, the primitive is in the left child's Normal_node at $t = 0.5$. It can either move across the split plane(cases 1 and 3) or pass completely into the right child area(cases 2 and 4). These primitives are initially assigned to left child's Normal_node in traditional k-d tree. Here, they are also duplicated into the right child's BP_node.*

at either end, we also quantified our intuition: we compared the performance of the MBKD when the initial $k$-d tree was built at times $= (0, 0.5, 1.0)$ by counting the number of traversals and intersections. Implementation also shows that using $t = 0.5$ is always better than either endpoint, with an average of 12% fewer intersections and 10% fewer traversals when compared against $t = 0$ and $t = 1$.

We also modify the traditional method of counting primitives to better account for motion. In the conventional SAH model, if a primitive crosses the split plane, the number of primitives of both the left and right child nodes increases by 1. However, that primitive may be only contained in one child node during a particular time period. Thus, time attributes can be added to the primitives in our method. If a primitive crosses the split plane between $t = 0$ and $t = 1$, we can compute the split time at the crossover point (two different entry and exit instances, $t = t_{c_1}$ and $t = t_{c_2}$). The left primitive has range $t = 0$ to $t = t_{c_2}$, and the right primitive has range $t = t_{c_1}$ to $t = 1$. Then, rather than counting that primitive as 1 in both the left and right nodes, we instead use $(t_{c_2} - 0)$ and $(1 - t_{c_1})$:

$$C_P = K_T + \frac{K_I}{SA(N)} \times [L + R]$$

$$L = \sum_{N_l} (t_{c_2} - t_0) \times SA(N_l)$$

$$R = \sum_{N_r} (t_1 - t_{c_1}) \times SA(N_r)$$

Here $\sum_{N_l} (t_{c_2} - t_0) \times SA(N_l)$ and $\sum_{N_r} (t_1 - t_{c_1}) \times SA(N_r)$ are respectively the intersection cost of the left and right nodes, $SA(N_l)$ and $SA(N_r)$ are respectively the surface area of the AABB in the left and right nodes, $SA(N)$ is the surface area of the AABB in the parent node, and $K_T$ and $K_I$ are respectively the cost constants for a traversal and an intersection operation. This change makes a better estimate of the primitives on each side of the split plane across the entire time interval, and obtains better split planes, better than the conventional SAH model with an average of 7% fewer duplication primitives in our experiment.

### 3.1.2. Propagation

Duplicating the crossing primitives in *Normal_nodes* guarantees that any ray cannot miss the correct intersection primitives in the leaf nodes along the tree hierarchy during the time shutter. The duplicated primitives in a *BP_node* are only responsible to its affiliated *Normal_node* of parent node, and they should be propagated into their two child nodes to ensure their contribution to motion blur. Here, we use a propagation heuristic for all the primitives in *BP_nodes* to assign them into the following child *Normal_nodes* or *BP_nodes* along the tree hierarchy until they reach the appropriated leaf nodes, as shown in Figure **??**. In this example, we show primitives moving from left to right. Since there are only 3 position relationships between a primitive and its nodes—fully in the left node, fully in the right node, and on the splitting plane—we only consider six propagation cases based on their ownership relationships during the entire interval.

Specifically, We use the same split plane in the parent node as judgment criterion to propagate the primitives in the *BP_node* of the parent, and the primitives in a *BP_node* can also be propagated into its child *Normal_node* or *BP_node*. If a primitive is only assigned into one or two child *Normal_nodes*, that means it will not happen to cross the current split plane during the time shutter. An efficient propagation strategy improves duplication primitive organization in the tree structure, and avoid the generation of large *BP_nodes*. While our duplication and propagation strategies eliminate invalid node references, they may cause more primitives to be contained in child nodes. Although we do not consider these new duplication primitives for SAH sorting, the propagation process from the *BP_nodes* may still bring some extra assignment cost. These costs are unavoidable but acceptable given the performance improvements in rendering that we discuss later.

## 3.2. The Traversal Phase

Our MBKD ray traversal algorithm follows a depth-first traversal order, and applies the traditional $k$-d tree traversal method for all *Normal_nodes* along the tree hierarchy. Rays determine traversal

routes down along the tree by comparing with split planes of the *Normal_node*s. The *BP_nodes* do not participate in the traversal process. Thus, they do not need to be maintained after the initial construction finish. When rays intersect with primitives in leaf nodes, because all the primitives that pass the node area are recorded, we can make intersection tests directly with each primitive. In addition, we also record the time intervals, in which contained in a node for these primitives, thus if the time interval of a primitive does not belong to the scope of current assigned ray time, it will abandon intersection computations.

## 4. Results

We implemented our MBKD using C++ on an Intel Core i7 4700MQ Processor. Our main comparison is against three popular choices for scenes with motion blur: a regular BVH method, the MSBVH [GSNK11]. Our test scenes consisted of the 4 models shown in our teaser image (Figure 1): Sponza, Ben, Hairball and Hand at sample = 16 with 512×512 resolution.

The test scenes demonstrate the need for motion blur effects and a wide spectrum of varied and difficult motion, to fully validate the efficiency of our method. For the Sponza scene, we rotate the camera at a fixed position for more irregular motion to render camera motion blur. The Ben scene shows a person in the Sponza geometry running in one direction with incoherent motion in Ben's arms and legs. This implies a hierarchy where the overall motion is fixed (the ground does not move), but the person has locally bad motion (object motion). In the Hairball scene, we tested the same geometry as used in MSBVH, and used the Bullet physics engine [Bul11] for some rather crude animation, every strand of hair moves in a different direction, this is an ideal example of incoherent motion.

For these scenes, we present two main results: building time and render time. The measurements were performed using a custom rendering system, based on the PBRT implementation [PH10]. Table 1 shows the results of our MBKD for motion blur with their building and render times, and compares the relevant and absolute figures of merit against an interpolated BVH baseline ("BVH_i") and MSBVH method. As shown in the results, the construction of MBKD and MSBVH usually takes a longer time than BVH_i, but requires less render time, which is similar to the comparison between a conventional *k*-d tree, BVH and SBVH. It means these corresponding motion blur structures—BVH_i, MSBVH, and MBKD–keep their original hierarchy merits (BVH, SBVH, and *k*-d tree) features and benefits. More importantly, compared to a MSBVH, our MBKD obtains faster traversal performance in all the test scenes, with similar or slightly-increased construction times in most of the test scenes, though the duplication process also brings some extra *BP_node*s construction cost. Figures 4 and 5 show a heat map visualization of the traversals and intersections per pixel, respectively. Our implementation of the MBKD achieves faster render time than previous methods, despite the increased cost of construction.

We also analyzed the quantity of duplicate primitives, additional duplicate cost, and propagation cost of the MBKD. Table 2 compares the relative number of incoherent primitives contained in *BP_node*s for the test scenes, and shows the comparison of the relative time respectively taken to finish the duplicate and propagation

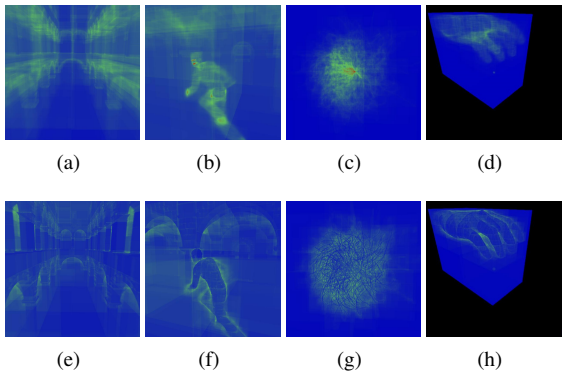



**Figure 4:** *The number of ray traversals with the bounding box per pixel visualized as a heat map for the test scenes Sponza, Ben, Hairball, and Hand.* $(a)-(d)$ *is for the interpolated BVH, and* $(e)-(h)$ *is for the MBKD.*

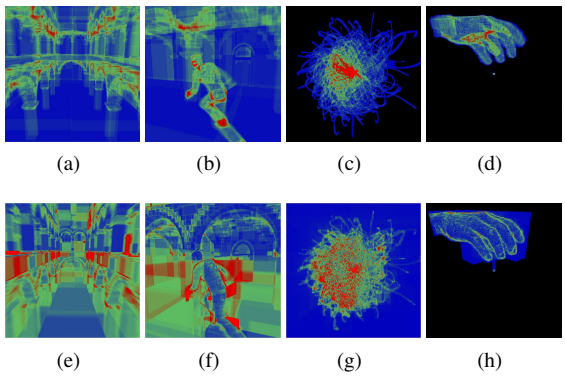



**Figure 5:** *The number of ray intersections with the bounding box per pixel visualized as a heat map for the test scenes Sponza, Ben, Hairball, and Hand.* $(a)-(d)$ *is for the interpolated BVH, and* $(e)-(h)$ *is for the MBKD.*

process during construction. The cost of the additional propagation work caused by the MBKD is modest: the quantity of duplicate primitives, the time cost brought by the additional primitives duplicate in *BP_node*s, and the time cost for the duplicate primitives' propagation together account for a small proportion of the corresponding overall cost. The results in Tables 1 verify that the propagation cost does not cause a degradation of rendering performance; instead, our MBKD has faster traversal compared with the other approaches, because of its more efficient spatial hierarchy structure.

An important difference of MBKD from a traditional *k*-d tree is the memory usage, though primitives duplicated in *BP_node*s definitely cause extra memory consumption. Table 3 compares the memory footprint. The results here show a small increase in memory consumption of up to 15% for the Hairball scene. This extra memory cost is acceptable given the performance improvement in rendering from method.

| *Scene* | *Acceleration Structure* | *Building Time (%)* | *Render Time (%)* | *Building Time (s)* | *Render Time (s)* |
|---|---|---|---|---|---|
| Sponza | BVH_i | 100 | 100 | 0.69 | 23.87 |
| | MSBVH | 152 | 73 | 1.05 | 17.39 |
| | **MBKD** | **158** | **62** | **1.09** | **14.90** |
| Ben | BVH_i | 100 | 100 | 1.99 | 24.44 |
| | MSBVH | 120 | 75 | 2.39 | 18.30 |
| | **MBKD** | **128** | **65** | **2.55** | **15.95** |
| Hairball | BVH_i | 100 | 100 | 5.66 | 17.59 |
| | MSBVH | 119 | 86 | 6.74 | 15.13 |
| | **MBKD** | **124** | **65** | **7.04** | **11.39** |
| Hand | BVH_i | 100 | 100 | 0.14 | 5.33 |
| | MSBVH | 150 | 74 | 0.21 | 3.96 |
| | **MBKD** | **164** | **55** | **0.23** | **2.91** |

**Table 1:** *This table shows a comparison of build-time and render-time between MBKD, MSBVH, and BVH_i to handle moving primitives. Building Time (%) and Render Time (%) show the same numbers normalized to a BVH_i.*

| | Duplication Quantity(%) | Duplication Time (%) | Propagation Time (%) |
|---|---|---|---|
| Sponza | 5.3% | 21.7% | 1.5% |
| Ben | 5.9% | 19.9% | 2.9% |
| Hairball | 10.5% | 17.9% | 5.9% |
| Hand | 13.9% | 23.4% | 15.5% |

**Table 2:** *Comparison of relative occurrence of duplicate-primitive quantity, duplication time, and propagation time in MBKD-trees.*

| *Structure* | Sponza | Ben | Hairball | Flake | Hand |
|---|---|---|---|---|---|
| BVH_i | 93MB | 170MB | 470MB | 65MB | 42MB |
| MSBVH | 101MB | 179MB | 503MB | 72MB | 47MB |
| **MBKD** | **122MB** | **219MB** | **578MB** | **87MB** | **55MB** |

**Table 3:** *Comparison of memory footprint of BVH_i, MSBVH and MBKD representation of the same scenes.*

## 5. Conclusions

Our MBKD is based on a *k*-d tree hierarchy and allows fast motion blur computation. In this work, we propose a method to process primitives moving across split planes, which otherwise causes node references to change and makes a classic *k*-d tree unsuitable for motion blur. The core idea is "duplication when moving across, redistribute when propagating". Our algorithm focuses specifically on handling incoherent motion, which can degrade the acceleration structure and cause a dramatic deterioration in rendering performance with incorrect rendering results. Beyond straightforward performance and parallelization enhancements to this work, we hope to explore what we believe is the most significant contribution of this work: our algorithmic approach of balancing and optimizing computation through duplication, propagation. We expect to extend this idea to other acceleration structures and use these lessons to design more appropriate data structures for dynamic rendering.

## Acknowledgement

The authors wish to acknowledge the support of NSFC grant 61632006, 61300084, 61370141, National High-tech R&D Program of China (Grant No. 2015AA7046207), Open Project Program of the State Key Lab of Structural Analysis for Industrial Equipment (Grant No. GZ15107), Program for Changjiang Scholars and Innovative Research Team in University (No.IRT_15R07), and the Fundamental Research Funds for the Central Universities (Grant No.DUT2017TB04). And thanks Yunfei Wang for video editing help.

## References

[Bul11] BULLET PHYSICS LIBRARY: Bullet 3D game multiphysics library. http://code.google.com/p/bullet/, 2011. 5

[CFLB06] CHRISTENSEN P. H., FONG J., LAUR D. M., BATALI D.: Ray tracing for the movie Cars. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing* (2006), IEEE, pp. 1–6. 2

[CKL*10] CHOI B., KOMURAVELLI R., LU V., SUNG H., BOCCHINO R. L., ADVE S. V., HART J. C.: Parallel SAH k-d tree construction. In *High Performance Graphics* (2010), pp. 77–86. 1, 2

[CPC84] COOK R. L., PORTER T., CARPENTER L.: Distributed ray tracing. In *Computer Graphics (Proceedings of SIGGRAPH 84)* (July 1984), pp. 137–145. 2

[DK08] DAMMERTZ H., KELLER A.: The edge volume heuristic - robust triangle subdivision for improved BVH performance. In *IEEE Symposium on Interactive Ray Tracing* (2008), pp. 155–158. 2

[ETH*09] EGAN K., TSENG Y.-T., HOLZSCHUCH N., DURAND F., RAMAMOORTHI R.: Frequency analysis and sheared reconstruction for rendering motion blur. *ACM Transactions on Graphics 28*, 3 (July 2009), 93:1–93:13. 1

[GBAM11] GRIBEL C. J., BARRINGER R., AKENINE-MÖLLER T.: High-quality spatio-temporal rendering using semi-analytical visibility. *ACM Trans. Graph. 30*, 4 (July 2011), 54:1–54:12. 2

[Gla88] GLASSNER A. S.: Spacetime ray tracing for animation. *IEEE Computer Graphics & Applications 8*, 2 (Mar. 1988), 60–70. 2

[GSNK11] GRÜNSCHLOSS L., STICH M., NAWAZ S., KELLER A.: MSBVH: An efficient acceleration data structure for ray traced motion blur. In *High Performance Graphics* (Aug. 2011), pp. 65–70. 2, 5

[Hav07] HAVRAN V.: *Heuristic Ray Shooting Algorithms*. Dissertation thesis, Lund University, Feb. 2007. 1

[HHRZ12] HUANG X., HOU Q., REN Z., ZHOU K.: Scalable programmable motion effects on GPUs. *Computer Graphics Forum 31*, 7pt2 (Sept. 2012), 2259–2266. 2

[HQL*10] HOU Q., QIN H., LI W., GUO B., ZHOU K.: Micropolygon ray tracing with defocus and motion blur. *ACM Transactions on Graphics 29*, 4 (July 2010), 64:1–64:10. 2

[Kar12] KARRAS T.: Maximizing parallelism in the construction of BVHs, octrees, and *k*-d trees. In *High Performance Graphics* (2012), pp. 33–37. 1

[MCH*11] MUNKBERG J., CLARBERG P., HASSELGREN J., TOTH R., SUGIHARA M., AKENINE-MÖLLER T.: Hierarchical stochastic motion blur rasterization. In *High Performance Graphics* (2011), pp. 107–118. 2

[MESL10] MCGUIRE M., ENDERTON E., SHIRLEY P., LUEBKE D.: Real-time stochastic rasterization on conventional GPU architectures. In *High Performance Graphics* (2010), pp. 173–182. 2

[Ols00] OLSSON J.: *Ray-Tracing Time-Continuous Animations using 4D KD-Trees*. Dissertation thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Nov. 2000. 2

[PAAV03] PROCOPIUC O., AGARWAL P. K., ARGE L., VITTER J. S.: *Bkd-Tree: A Dynamic Scalable kd-Tree*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 46–65. 2

[PGSS06] POPOV S., GÜNTHER J., SEIDEL H.-P., SLUSALLEK P.: Experiences with streaming construction of SAH KD-trees. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing* (Sept. 2006), pp. 89–94. 1

[PH10] PHARR M., HUMPHREYS G.: *Physically Based Rendering, Second Edition: From Theory To Implementation*, 2nd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010. 5

[SFD09] STICH M., FRIEDRICH H., DIETRICH A.: Spatial splits in bounding volume hierarchies. In *High Performance Graphics* (2009), pp. 7–13. 2

[SSBG10] SCHMID J., SUMNER R. W., BOWLES H., GROSS M.: Programmable motion effects. *ACM Transactions on Graphics 29*, 4 (July 2010), 57:1–57:9. 2

[SSK07] SHEVTSOV M., SOUPIKOV A., KAPUSTIN A.: Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes. *Computer Graphics Forum 26*, 3 (2007), 395–404. 2

[VTS*12] VAIDYANATHAN K., TOTH R., SALVI M., BOULOS S., LEFOHN A.: Adaptive image space shading for motion and defocus blur. In *High Performance Graphics* (2012), pp. 13–21. 2

[Wal07] WALD I.: On fast construction of SAH-based bounding volume hierarchies. In *Proceedings of the 2007 IEEE/Eurographics Symposium on Interactive Ray Tracing* (Sept. 2007), pp. 33–40. 3

[WK06] WÄCHTER C., KELLER A.: Instant ray tracing: The bounding interval hierarchy. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques* (Aire-la-Ville, Switzerland, Switzerland, 2006), EGSR '06, Eurographics Association, pp. 139–149. 2

[WMG*09] WALD I., MARK W. R., GÜNTHER J., BOULOS S., IZE T., HUNT W., PARKER S. G., SHIRLEY P.: State of the art in ray tracing animated scenes. *Computer Graphics Forum 28*, 6 (2009), 1691–1722. 1

[WZL11] WU Z., ZHAO F., LIU X.: SAH KD-tree construction on GPU. In *High Performance Graphics* (2011), pp. 71–78. 2

[ZHR*09] ZHOU K., HOU Q., REN Z., GONG M., SUN X., GUO B.: RenderAnts: Interactive Reyes rendering on GPUs. *ACM Transactions on Graphics 28*, 5 (Dec. 2009), 155:1–155:11. 2

[ZHWG08] ZHOU K., HOU Q., WANG R., GUO B.: Real-time KD-tree construction on graphics hardware. *ACM Transactions on Graphics 27*, 5 (Dec. 2008), 126:1–126:11. 2