# Enhancing Time and Space Efficiency of Kd-tree for Ray-tracing Static Scenes

Byeongjun Choi, Woong Seo, Insung Ihm
Department of Computer Science and Engineering
Sogang University, Seoul, Korea

## Abstract

In the ray-tracing community, the surface-area heuristic (SAH) has been employed as a de facto standard strategy for building a high-quality kd-tree. Aiming to improve both time and space efficiency of the conventional SAH-based kd-tree in ray tracing, we propose to use an extended kd-tree representation for which an effective tree-construction algorithm is provided. Our experiments with several test scenes revealed that the presented kd-tree scheme significantly reduced the memory requirement for representing the tree structure, while also increasing the overall frame rate for rendering.

**Keywords:** Ray tracing, kd-tree construction, surface area heuristic, voxel visibility, space reduction.

## 1 Backgrounds

The kd-tree, routinely used for efficient ray tracing, has a problem that a large number of triangles intersecting with splitting planes are repeatedly duplicated during the recursive space-subdivision process, which often leads to inefficient, large and tall binary trees with high triangle redundancy. It was reported in [Choi et al. 2013] that for the well-known Conference scene, the total number of triangles in the leaf nodes increased more than 14 times that of the original triangles. To ease the problem, they proposed a new space-efficient kd-tree representation where an inner node is permitted to optionally store a triangle that would otherwise be duplicated in an excessive number of the leaf nodes in the standard representation. Combined with the new heuristic measures for deciding when and how to select triangles for inner nodes, their kd-tree scheme markedly reduced the memory demands for storing the tree structure, while avoiding any serious degradation of the timing performance.

On the other hands, the SAH, which is used as a de facto standard strategy for building a good kd-tree, is based on rather simple assumptions that may not always hold for complicated ray-tracing situations, thus often leading to inaccurate cost metrics. In another work, Choi et al. presented the concept of "voxel-visibility heuristic" for defining improved cost metrics, allowing more sophisticated estimation of the chance of a voxel being hit by rays [Choi et al. 2012]. With these advanced cost metrics, they were able to build kd-trees that reduced markedly the cost of ray-object intersection computation, increasing significantly the frame rate for ray tracing.

## 2 Our approach and results

Interestingly enough, the aforementioned two previous works, one for alleviating the memory space requirement and the other for speeding up the ray tracing computation, can effectively be combined to enhance both time and space efficiency of the kd-tree in ray-tracing various scenes with nontrivial complexities. Figure 1 illustrates the proposed kd-tree construction algorithm that exploits the advantages of both methods. In the pre-processing stage, an initial kd-tree is constructed using the cost metric based on the voxel visibility as described in [Choi et al. 2012]. For this, we discretize an input scene domain using a rectangular grid of resolution $256 \times 256 \times 256$, and sample 4,096 directions uniformly around each cell to evaluate the voxel visibility, i.e., to estimate the incident ray density for each of the six directions (Generate a visibility dataset). Then, the grid dataset is exploited to build a kd-tree using the cost metric defined in terms of the estimated voxel visibility values (Build an initial kd-tree using the visibility-based cost metric).

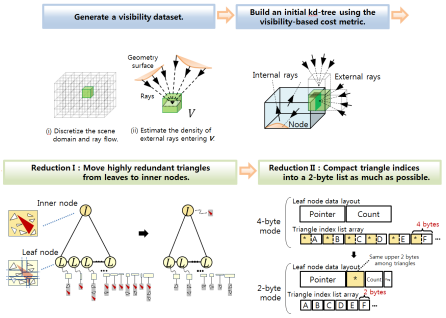


**Figure 1:** *Hybrid kd-tree construction algorithm.*

Once a time efficient kd-tree is obtained, it is converted into space more efficient one as described in [Choi et al. 2013]. First, a simple recursive process is carried out using the occupancy and frequency measures for removing excessively duplicate triangles from the leaf nodes of the kd-tree (Reduction I). After the culled triangles are stored in proper inner nodes, the triangle index list associated with the leaf nodes of the resulting tree is compressed further using a compact data layout (Reduction II).

For several scenes with about 100K to 13M triangles, we have tested the new kd-trees and evaluated both memory space requirements and frame rates. When images were rendered at $1024 \times 1024$ pixels on a PC with dual 3.1 GHz Intel Xeon 8-Core CPUs, the new kd-tree scheme was able to achieve not only 33.36% of kd-tree-size reduction on average but also up to 7.8% of frame-rate increase, compared to the renderings with the kd-trees built by the routinely used construction technique [Wald and Havran 2006]. In particular, when the hybrid construction algorithm was applied to large scenes like San Miguel (10,500,551 triangles) and Power Plant (12,748,510 triangles), we were able to achieve 33.4% and 45.1% of memory reduction, respectively, whereas the frame rate generally increased up to 7.8%.

## References

CHOI, B., CHANG, B., AND IHM, I. 2012. Construction of efficient kd-trees for static scenes using voxel-visibility heuristic. *Computers & Graphics 36*, 1, 38–48.

CHOI, B., CHANG, B., AND IHM, I. 2013. Improving memory space efficiency of kd-tree for real-time ray tracing. *Computer Graphics Forum 32*, 7, 335–344.

WALD, I., AND HAVRAN, V. 2006. On building fast kd-trees for ray tracing, and on doing that in O(Nlog N). In *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, 61–69.