Ordered Depth-first Layouts for Ray Tracing

Jae-Ho Nah[1]*, Jeong-Soo Park[1], Jin-Woo Kim[1], Chanmin Park[2], Tack-Don Han[1]
Yonsei University, Korea[1]
Samsung Electronics, Korea[2]

Abstract

We present an ordered depth-first tree layout for ray tracing. Among two child nodes, a child node which has larger surface area is stored next to its parent node. Hence, the probabilities that a ray visits same cache line increase. Our approach can be easily and widely used to various ray tracing systems with very small overhead because it is based on existing depth-first layouts.

CR Categories: Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Ray tracing, Computer Graphics [I.3.3]: Computational Geometry and Object Modeling—Hierarchy and Geometric Transformations

Keywords: cache, tree layout, ray tracing

1 Introduction

Binary space partitioning (BSP) trees are widely used in computer graphics. Especially, axis-aligned BSP trees (a.k.a. kd-trees) are a representative acceleration structure in ray tracing [Havran 2000]. Because ray tracing with tree structures needs many visits to nodes, it is important to design cache-efficient layouts for high-performance ray tracing.

In this paper, we present a novel tree layout for efficient ray tracing. It is an improvement of depth-first layout to utilize parent-child localities. In this layout, we determine the order of the left and right child nodes according to their surface area, so the probability that a ray visits same cache line increase. In our benchmarks, our ordered depth-first layouts show lower cache miss ratio than original 8-byte depth-first layouts and 12-byte ordinary subtree layouts.

Our ordered depth-first layouts have certain advantages. First, these layouts reduce cache misses during ray traversal, so improve ray tracing performance. Second, our approach is cache-oblivious and platform-independent, so it can be applied widely. Third, our approach requires only simple modifications of depth-first layouts, so it has negligible overhead and can be easily integrated to existing ray tracing systems.

2 Related Work

In kd-trees, a compact 8-byte depth-first layout [Pharr and Humphreys 2010] is widely used for ray tracing because of its small memory footprint and depth-first search manner of ray traversal. In this layout, a parent node and left-child node is adjacent, so they have parent-child localities. Therefore, this layout has only a single pointer for right child node. An 8-byte node is made up two values. First 32-bit value is split value for split coordinate in case of the inner node or to the number of primitives in case of leaves. Also, a two-bit flag to differentiate between inner nodes with three split axes and leaf nodes is squeezed into this first value. Second 32-bit value is a pointer to right child in case of the inner node or the primitive lists in case of leaves.

Subtree layouts more actively utilize parent-child localities than depth-first layouts. These approaches decompose an entire tree into clusters. [Havran 1999] presented two cache-aware subtree representations: an ordinary subtree and a compact subtree. Each subtree fits to a cache line in these layouts, so parent-child localities in a subtree exist on both the left and right child nodes. The ordinary subtree layout requires both two child node pointers because there are no continuity between subtrees. Thus, the ordinary subtree layout needs at least 12 bytes per node. The compact subtree layout removes pointers among the nodes inside the subtree, but it complicates ray traversal for node addressing.

[Yoon and Manocha 2006] presented cache-oblivious subtree layouts. This is designed to work well for different memory hierarchies including the L1/L2 caches, main memory, and disk. This algorithm requires preprocessing for subtree clustering to predict the runtime access patterns. This preprocessing can be a burden on ray tracing of dynamic scenes.

3 Our Approach

We improve 8-byte depth first layouts. In original depth-first layout [Pharr and Humphreys 2010], the arragment criterion of child nodes is geometric position. In contrast, ordered depth-first layouts use surface area to arrange child nodes instead of geometric position. That is, the child node which has larger surface area is stored next to its parent node. Because the probability of a ray intersecting a node is proportional to its surface area, the probabilities that a ray visits same cache line also increase.

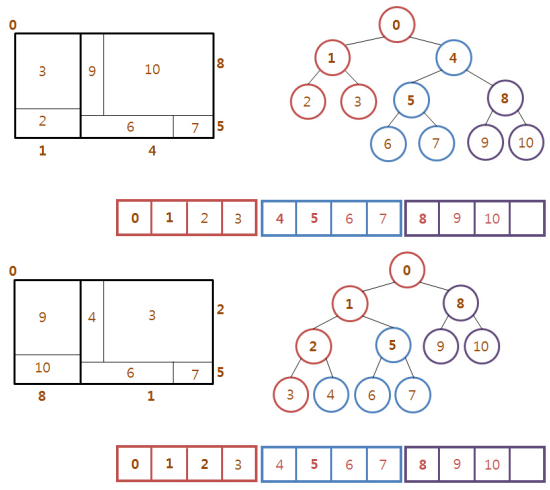

Figure 1: *(up) original depth-first layout, (down) ordered depth-first layout*

*e-mail: jhnah@msl.yonsei.ac.kr

Figure 1 shows how our approach differs from original depth-first layouts. Figure 1-(up) and 1-(down) represents an original depth-first layout and our ordered depth-first layout, respectively. For each case, leaf node 10 and 3 have the largest surface area among entire leaf nodes. If a ray visits leaf node 10 in figure 1-(up), the ray fetches to three cache lines. In contrast, if a ray fetches leaf node 3 in figure 1-(down), the ray visits to only one cache line because node 3 is stored in the same cache line to node 0 (root).

Because most of tree construction algorithms are based on surface area heuristic (SAH) [MacDonald and Booth 1990], we reuse this surface area value without any overhead when we determine order of child nodes. Only we add 1-bit flag to represents whether node is ordered in reverse to original depth-first layout. Because this 1-bit reorder flag can be embeded into 8-byte node representation, our layout doesn't need additional memory space.

Tree traversal using ordered depth-first layout is almost same except for the use of the 1-bit reorder flag. For front-to-back tree traversal, we should reference the reorder flag. If the flag is true, we consider two child nodes are reversed.

4 Experimental Results

We implemented a Whitted ray tracer[Whitted 1980] based on kd-trees to feasible proposed layouts. The ray tracer makes a statistical data of memory accesses which used for input of Dinero IV cache simulator [Edler and Hill 1998]. We chose three benchmark scenes: BART kitchen with 110K triangles, Fairy with 174K triangles, and Sponza with 80K triangles (Figure 2). For tree construction, we used a SAH with maximum tree depth 32 and leaf size 1. For ray traversal, we used single-ray recursive tracing. All images were rendered with 512x512 resolution and ray recursion depth was set to 4.

Figure 2: *Benchmark scenes - BART kitchen, Fairy, and Sponza*

We compared three different layouts: 8-byte depth-first layouts, 12-byte ordinary subtree layouts, and proposed 8-byte ordered depth-first layouts. We set the 4-way 8 KB set associative cache with 64 byte block size for performance evaluation. This configuration is similar to modern many-core processors.

Table 1 and 2 shows benchmark results. Our ordered depth-first layouts outperform other approaches in all used scenes. Especially, our ordered depth-first layouts reduce the required memory bandwidth by 16 - 30 percent compared to original depth-first layouts even with same memory footprint. In addition, our ordered depth-first layouts have same memory footprints to original depth-first layouts, so ours have around 40 percent less memory footprints than subtree layouts. Our approach stores 8 nodes in a 64 byte cache line, but ordinary subtree layout can store up to 5 nodes in a 64 byte cache line due to 12-byte node data and additional pads for cache alignment.

Table 1: *Benchmark results 1 - cache miss ratio (percent)*

Scene	Depth-first Layout	Ordinary Subtree	Ordered Depth-first Layout
BART kitchen	4.62	4.48	3.50
Fairy	11.29	9.25	7.84
Sponza	1.89	1.76	1.57

Table 2: *Benchmark results 2 - memory footprint*

Scene	The number of nodes	Memory usage	
		Ordinary Subtree	Depth-first Layout
BART kitchen	463481	5.8MB	3.7MB
Fairy	1817027	22.7MB	14.5MB
Sponza	918383	11.5MB	7.1MB

5 Conclusions

We proposed an ordered depth-first layout which maximizes parent-child locality using simple node ordering. Due to simplicity and cache-oblivious manner of our method, it can be widely appliable to ray tracers based on CPUs, GPUs, and dedicated hardware. We expect that our method can be applied to not only kd-trees but also other tree hierarchies such as general BSP trees and bounding volume hierarchies. In addition, we think our method can be useful for other applications which use depth-first search such as collision detection, photon mapping, and so on.

Acknowledgements

This work was supported by Samsung Electronics Co., Ltd. The Kitchen scene is the courtesy of A Benchmark for Animated Ray Tracing (BART). The Fairy scene is the courtesy of Utah 3D Animation Repository. The Sponza scene is the courtesy of Marko Dabrovic. We would like to thank the folks at ompf.org for their valuable discussions.

References

EDLER, J., AND HILL, M., 1998. Dinero IV trace-driven uniprocessor cache simulator. `http://www.cs.wisc.edu/~markhill/DineroIV/`.

HAVRAN, V. 1999. Analysis of cache sensitive representation for binary space partitioning trees. *Informatica 23*, 3 (May), 203–210.

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

MACDONALD, D. J., AND BOOTH, K. S. 1990. Heuristics for ray tracing using space subdivision. *The Visual Computer 6*, 3, 153–166.

PHARR, M., AND HUMPHREYS, G. 2010. *Physically Based Rendering*, 2 ed. Elsevier.

WHITTED, T. 1980. An improved illumination model for shaded display. *Communications of the ACM 23*, 6 (June), 343–349.

YOON, S.-E., AND MANOCHA, D. 2006. Cache-efficient layouts of bounding volume hierarchies. *Computer Graphics Forum*, 507–516.