Fragmentation-Aware BVH Contraction

Shinji Ogaki

July 30, 2016

Abstract

N-ary BVH is developed to exploit SIMD instructions and has become one of the most popular data structures in CPU ray tracing. The state-of-the-art n-ary BVH construction algorithms create partially filled nodes, which leads to poor memory utilization. This becomes more noticeable for wider n-ary BVHs. We address this issue by merging multiple n-ary nodes that have empty space. The proposed method reduces the number of n-ary nodes by about 40% without compromising rendering performance.

1 Introduction

Today the n-ary bounding volume hierarchy (BVH) is widely used to achieve high performance on modern CPUs. In order to efficiently perform ray traversal, BVH should be stored in a SIMD-friendly layout. The common data structure for the node of n-ary BVHs is a SOA (structure of arrays), and the whole tree is normally presented as an AOSOA (array of structure of arrays).

There are mainly two problems when utilizing increasingly wider SIMD units. One is that ray tracing performance does not scale linearly with the SIMD width because of the cost of sorting intersected nodes. Another often overlooked problem is that partially filled nodes are created during construction. For example, OBVH (8-ary BVH) has empty slots up to 6 and thus 75% of the space for node references is wasted in the worst case.

In this article we focus on the later memory fragmentation problem and introduce a simple algorithm to remove empty space by merging multiple n-ary nodes that have empty space. Our data structure uses a DAG (directed acyclic graph) representation similar to the one used in sparse voxel octrees [8].

Our method runs fast and uses a small amount of working memory because n-ary nodes are merged during contraction. With our algorithm the number of n-ary nodes is reduced by about 40% compared to the state-of-the-art methods in the case of OBVH. Furthermore, the introduced overhead in ray traversal is almost negligible.

2 Related Work

2.1 Collapsing BVH

An n-ary BVH can be built by collapsing a binary BVH [3, 1].

Dammertz et al. [3] build a 2^k-ary BVH ($k = 2, 3, \cdots$) by keeping each k-th level of a binary BVH and discarding the rest. This approach creates partially filled nodes in the middle of the tree.

The other common approach is to use contraction techniques. Surface area contraction [13] collapses binary nodes with a larger surface area first to reduce the SAH cost of the resulting n-ary BVH. Ray specialized contraction [7] aims to achieve better performance by collapsing frequently visited nodes first. Before collapsing a binary BVH, this technique requires to cast a small number of representative rays to roughly estimate how often each node is traversed. Both methods provide high rendering performance, however, create nodes with empty space at leaf nodes.

Pinto [11] introduced an adaptive collapsing technique where each node can have variable number of child nodes. In the paper two different algorithms were proposed: (1) greedy top down method and (2) optimal collapse-node method. The former is equivalent to surface area contraction and the later achieves better performance by using dynamic programming. It is shown that both methods run faster than QBVH (4-ary BVH), however, the performance comparison is done using all-hit test and the improvement for closest-hit test is not reported.

2.2 Directed Acyclic Graph Representation

Sparse voxel representations such as SVO (sparse voxel octree) are a standard way of encoding volume data. A DAG based representation allows to reduce the number of nodes by one to three orders of magnitude [8]. There are a few variants that exploit a similarity transform [12] or embed arbitrary data such as colors [2] to achieve further efficiency. However, these approaches cannot be directly applied to n-ary BVHs used for ray tracing because geometric primitives in each leaf are normally different.

3 Background on Contraction

As background, we first review the the contraction algorithm to create a non-fixed-witdth n-ary BVH from a binary BVH. As defined by Gu et al. [7], *contraction* for an interior BVH node is an operation to hoist all its children to its parent, and remove this node from the tree.

The expected cost of ray tracing on the given n-ary node A is defined as

$$C_A = \sum_{i=1}^{N_A} C_b + P_A(i)C_i, \tag{1}$$

where C_i is the cost of traversing the i-th child node of A, $P_A(i)$ the probability of a ray hitting the i-th child node knowing it hits A, N_A the number of child nodes of the node A, and C_b the cost of ray-box intersection test. A contraction operation is beneficial if the following condition is met:

$$C_b + C_A > \sum_{i=1}^{N_A} C_b + P_{A_{parent}}(i) C_i, \tag{2}$$

where A_{parent} is the parent node of the node A.

Extending the cost model for fixed-width n-ary BVHs (N_A is not greater than the SIMD width N), we have

$$C_A = C_{mb} + \sum_{i=1}^{N_A} P_A(i) C_i, \tag{3}$$

where C_{mb} the cost of multi-box intersection test performed by SIMD units. The state-of-the-art top-down contraction techniques [7] collapse nodes until node elements are filled, or there is nothing to collapse.

4 Fragmentation-Aware Contraction

Figure 1: Splitting one of primitive lists of a leaf to fill empty space.

Although we can easily fill empty space by adjusting the number of primitives per leaf (Figure 1), this is neither always possible nor necessarily improves the quality of tree. Therefore, there is a need for a better method. For the sake of simplicity, we assume that each leaf has only one geometric primitive. This assumption is actually meaningful when using instanced objects, i.e. there exists a leaf contains a pointer to another BVH.

Our algorithm first builds a binary BVH and then collapses it using a top-down contraction algorithm. A binary BVH can be built by using any of existing methods. (Note that, in practice, an n-ary BVH should be built directly not to waste memory. It is not necessary to explicitly build a binary BVH and then collapse it into an n-ary BVH. However, the mechanism behind the direct construction uses the same idea of contraction algorithms.) The difference of our contraction algorithm from the state-of-the-art techniques is that multiple subtrees containing a small number of primitives are contracted into a single n-ary node. As a result, n-ary nodes with empty space are merged and the memory fragmentation problem is solved. In the following subsections, we elaborate the details of our algorithm.

4.1 Moving Empty Space to Leaf Nodes

Advanced BVH builders such as Bonsai [6] cause partially filled nodes in the middle of the tree, and Ganestam et al. [5] do not propagate empty space to leaf nodes because it does not improve performance. However, we want to ensure that empty slots are only created at leaf nodes to make our algorithm simple. Therefore, we use a top-down contraction algorithm.

4.2 Merging Nodes with Empty Slots

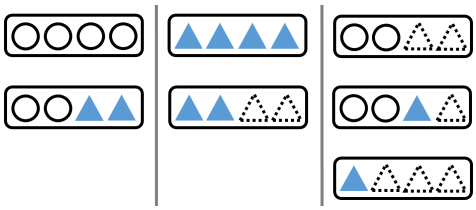


Figure 2: Various node types. Circles represent pointers to child nodes, blue triangles are pointers to primitives, and dashed triangles are empty slots. The nodes in the left and middle columns exist but the nodes in the right column do not exist.

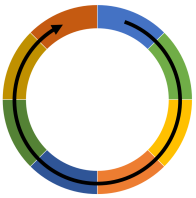


Figure 3: We use a temporal ring buffer with the size of 8. This buffer stores the indices of previously generated n-ary nodes with more than one empty slots.

N-ary nodes with empty space are merged not after but during the collapsing phase to reduce the amount of working memory. In order to efficiently perform this merging operation, it is necessary to quickly find where such nodes are located. We tested a variety of methods and found the following simple algorithm works quite well.

The proposed algorithm uses a small ring buffer to store indices of n-ary nodes that have empty slots. When a subtree containing $L(< N-1)$ geometric primitives is collapsed, we would like to find a node with $M(= N-L)$ empty slots. However, this is not always possible, hence we try to find a node that minimizes $|(L+M)-N|$ such that $L+M \leq N$ by scanning all entries of the buffer (Figure3). If such a node is found, the subtree is merged into the node

Listing 1: N-ary BVH node structure.

```
#include<immintrin.h>
static const int N = 8;
// N Floats
typedef __m256 FloatN;
// N Bounding Boxes
struct Bounds3dN
{
        // For Min and Max
        FloatN BoundsX[ 2 ];
        FloatN BoundsY[ 2 ];
        FloatN BoundsZ[ 2 ];
};
// N-ary BVH Node
struct NodeN // 256 Bytes
{
        Bounds3dN Boxes;             // 192
        int       Indices      [N]; // 32
        short     NumPrimitives[N]; // 16
        short     BitMasks     [N]; // 16
};
```

to fill its empty space. If not, a new n-ary node is created. The index of the newly created node is written to the buffer in a round robin fashion if it has more than one empty slots. (This is because if there is only one empty slots, this node cannot be shared as explained in Figure 2.) We found that setting the size of the ring buffer as 8 works well.

4.3 Node Data Structure

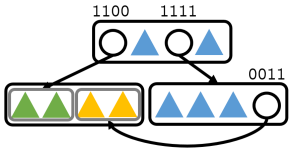
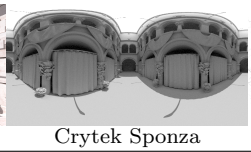
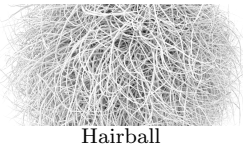


Figure 4: Bit masks.

Since there are nodes referred by multiple nodes, we need to be able to identify which node elements belong to which node. This is crucial to avoid redundant intersection tests, and also useful when combining our algorithm with shadow ray traversal order techniques [9, 10]. Therefore, we add bit masks to the n-ary BVH node data structure (Figure 4 and Listing 1). The introduced overhead in the ray-traversal code is almost negligible because it is only a single bit mask operation per node visit.

5 Results

	Octane sample	Crytek Sponza	Hairball
Fill rate	99.8% (54.7%)	99.6% (56.0%)	98.5% (54.5%)
Contraction time	14ms (13ms)	62ms (52ms)	703ms (699ms)
# nodes	11296 (20619)	42391 (75423)	469124 (848193)

Table 1: Fill rate, contraction time, and the number of nodes with and without our algorithm.

We implemented the proposed technique in our rendering system. We run all tests on a PC with Intel ®Core ™i7-6700T with Turbo Boost disabled. In our tests BVH is built with object splitting and the number of primitives per leaf is set as 1. We define the fill rate R as:

$$R = (1 - \frac{\sum_{i=1}^{T} V(i)}{T \times N}) \times 100, \tag{4}$$

where T is the total number of n-ary nodes, N the width of SIMD, and $V(i)$ gives the number of the empty slots of the i-th node. We achieved an over 98% fill rate using a ring buffer of size 8 for all scenes, and the number of nodes are reduced by about 40%. There is a subtle increase in the contraction time.

6 Conclusion and Future Work

In this article we presented a very simple algorithm using a DAG representation to efficiently eliminate empty space created in n-ary BVHs. The number of nodes is reduced by about 40% compared to the state-of-the-art techniques. We also showed that the overheads introduced both in the contraction and ray traversal parts are very small.

We would like to investigate if it is possible to find a combination of subtrees to improve rendering performance. Extending this approach to efficiently implement treelet restructuring [4] for N-ary BVHs would be another interesting research avenue.

Acknowledgement

The Octane sample model is included in Octane renderer's test scenes (`https://home.otoy.com/render/octane-render/demo/`). The Crytek Sponza and hairball models are available at `http://graphics.cs.williams.edu/data/meshes.xml`.

References

[1] Attila T. Áfra. Faster Incoherent Ray Traversal Using 8-Wide AVX Instructions. Technical report, Babeş-Bolyai University, Cluj-Napoca, Romania, August 2013.

[2] Bas Dado, Timothy R. Kol, Pablo Bauszat, Jean-Marc Thiery, and Elmar Eisemann. Geometry and Attribute Compression for Voxel Scenes. *Computer Graphics Forum (Proc. Eurographics)*, 35(2), may 2016.

[3] Holger Dammertz, Johannes Hanika, and Alexander Keller. Shallow Bounding Volume Hierarchies for Fast SIMD Ray Tracing of Incoherent Rays. In *Proceedings of the Nineteenth Eurographics Conference on Rendering*, EGSR '08, pages 1225–1233, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[4] Leonardo R. Domingues and Helio Pedrini. Bounding Volume Hierarchy Optimization Through Agglomerative Treelet Restructuring. In *Proceedings of the 7th Conference on High-Performance Graphics*, HPG '15, pages 13–20, New York, NY, USA, 2015. ACM.

[5] P. Ganestam, R. Barringer, and M. Doggett. SAH guided spatial split partitioning for fast BVH construction. *Journal of Computer Graphics Techniques (JCGT)*, 35(2), 2016.

[6] P. Ganestam, R. Barringer, M. Doggett, and T. Akenine-Möller. Bonsai: Rapid Bounding Volume Hierarchy Generation using Mini Trees. *Journal of Computer Graphics Techniques (JCGT)*, 4(3):23–42, September 2015.

[7] Yan Gu, Yong He, and Guy E. Blelloch. Ray Specialized Contraction on Bounding Volume Hierarchies. *Computer Graphics Forum*, 2015.

[8] Viktor Kämpe, Erik Sintorn, and Ulf Assarsson. High Resolution Sparse Voxel DAGs. *ACM Trans. Graph.*, 32(4):101:1–101:13, July 2013.

[9] Jae-Ho Nah and Dinesh Manocha. SATO: Surface Area Traversal Order for Shadow Ray Tracing. *Computer Graphics Forum*, 33(6):167–177, 2014.

[10] S. Ogaki and A. Derouet-Jourdan. N-ary BVH Child Node Sorting Technique for Occlusion Test. *Journal of Computer Graphics Techniques (JCGT)*, 5(2):23–42, April 2016.

[11] Andre Susano Pinto. Adaptive Collapsing on Bounding Volume Hierarchies for Ray-Tracing. In H. P. A. Lensch and S. Seipel, editors, *Eurographics 2010 - Short Papers*. The Eurographics Association, 2010.

[12] Alberto Jaspe Villanueva, Fabio Marton, and Enrico Gobbetti. SSVDAGs: Symmetry-aware Sparse Voxel DAGs. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '16, pages 7–14, New York, NY, USA, 2016. ACM.

[13] Ingo Wald, Carsten Benthin, and Solomon Boulos. Getting rid of packets - efficient simd single-ray traversal using multi-branching bvhs . In *IEEE Symposium on Interactive Ray Tracing, 2008. RT 2008*, pages 49–57. IEEE, 2008.