1 Introduction

Bounding volume hierarchies (BVHs) are widely used to accelerate the performance of various geometric and graphics applications. These applications include ray tracing, collision detection, visibility queries, dynamic simulation, and motion planning. These applications typically precompute BVHs of input models and traverse the BVHs at runtime in order to perform intersection or culling tests.

A major problem with using BVHs is that BVHs require large amounts of the memory space. Therefore, BVHs of large models consisting of hundreds of millions of triangles can take tens of gigabytes of space. Moreover, the typical data access pattern on BVHs cannot be determined at the preprocessing time and is random at runtime. Therefore, accessing BVHs at runtime can have low I/O efficiency and cache utilization.

An aggravating trend is that the growth rate of the data access speed is significantly slower than that of the processing speed. Therefore, the problem of high storage requirements and low I/O efficiency/cache utilization of BVHs will become more pronounced in the future.

Several approaches have been developed to address this problem. One class of methods uses compact in-core BV representations by using quantized BV information or by exploiting the connectivity information of an original mesh and coupling the mesh and the BVH. Another class of methods stores BV nodes in a cache-coherent manner to improve cache utilization and, thus, improve the performance of traversing BVHs. However, due to the widening gap between data access speeds and processing speeds, prior work may not provide enough reduction in storage requirements nor achieve high I/O efficiency during the BVH traversal.

2 Our Approach

In order to efficiently access BVHs and improve the performance of various applications using BVHs, we propose a novel BVH compression and decompression method supporting random access. We compress BVs of a BVH by sequentially reading BVs in the BV layout of the BVH. We choose our compression method to preserve the original layout of the BVH in order to achieve the high cache utilization which the original layouts may maintain. We decompress the original layout of the BVH into a set of clusters. We assign consecutive BVs in the BV layout to each cluster and set each cluster to have the same number of BVs to quickly identify a cluster containing a BV node requested by an application at runtime. We compress each cluster separately from other clusters so that the clusters can be decompressed in any order. We define an atomic BVH access API supporting transparent and random access on the compressed BVHs. The runtime framework fetches and decompresses the cluster into an in-core representation. Based on our in-core representation, we can very efficiently support random access to applications. Our runtime BVH access framework is guaranteed to return the correct BV information of the requested data when applications access the compressed data via our BVH access API.

We employ the RACM representation [Yoon and Lindstrom 2007] to further reduce the storage requirement of meshes, which are used together with BVHs for various applications.

We achieve up to a 12 : 1 compression ratio in our benchmark models. We implement two different applications, ray tracing and collision detection, to verify the benefits of our proposed method. In these applications, we improve the performance by up to a factor of four over using uncompressed data.

3 Advantages of Our Approach

1. **Wide applicability**: The provided BVH access API allows various applications to transparently access the compressed BVHs. Moreover, our BVH access API supports random access and does not restrict the access pattern of BVH-based applications.

2. **Low storage requirement**: Our RACBVH representation has up to a 12:1 compression ratio compared to an uncompressed BV representation.

3. **Improved performance**: We can achieve up to a 4:1 performance improvement on our tested applications over using uncompressed data.

References