Fully Parallel Kd-Tree Construction for Real-Time Ray Tracing

Zonghui Li, Tong Wang, and Yangdong Deng
Institute of Microelectronics, Tsinghua University
zonghui.lee@gmail.com sagacamus@163.com dengyd@mail.tsinghua.edu.cn

Abstract

This work proposes a fully parallel kd-tree construction algorithm, which depends on the Morton code to identify all candidate split planes and derive their exact positions in parallel. Our techniques drastically shorten construction process. Experimental results on a set of frequently used scenes prove that the proposed kd-tree construction algorithm outperforms a state-of-the-art algorithm kd-tree construction algorithm by over one order of magnitude.

Keywords: real-time ray tracing, kd-tree, kd-tree construction, Morton code, GPU

1. Introduction

Kd-tree is one of the most popular spatial acceleration structures for ray tracing. It recursively partitions the space to adapt to the spatial distribution of objects in a top-down manner. Typically, the kd-tree construction starts from the root node and proceeds in a breadth-first fashion. As a result, the limited amount of available parallelism at the first a few levels leads to under-utilization of the computing power of modern GPUs. Moreover, the obtainable parallelism at a given level can only be known after the previous level is constructed. Such a fact suggests frequent launching of kernels under the GPU programming model and also hinders a higher level of performance.

In this paper, we present a highly efficient kd-tree construction algorithm, which can be effectively implemented on GPUs. We borrow the idea of Morton code based ordering of primitives. Instead of resorting to SAH, we utilize the inherent characteristics to identify split planes. A path compression procedure then build the while tree hierarchy. Instead of building a kd-tree layer-by-layer from the top, the whole processing flow can be performed in fully parallel fashion by taking advantage of a conceptual complete binary tree. The proposed approach enables a higher level of parallelism.

2. Fully Parallel Kd-tree Construction

The algorithmic flow of our kd-tree construction algorithm is organized as follows. First, the primitives are ordered by the Morton curve. A balanced and complete tree corresponding to a regular distribution of primitives is also conceptually created for the underlying 3-D grid. We only use the concept of the above balanced tree but never really build it. Figure 1 shows such a tree. The primitives in a finest sub-space can be naturally grouped into a trunk according to their Morton codes. The trunks serve as the leaf nodes and are attached to the target kd-tree. Since every unique Morton code has a corresponding node in the balanced tree, many nodes are actually redundant when primitives in a scene are

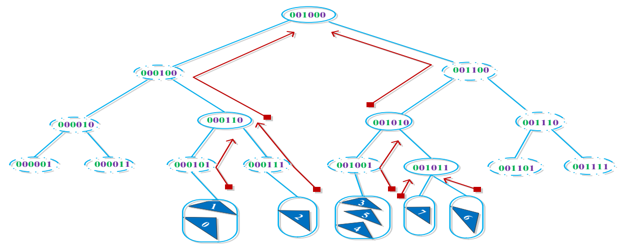

Figure 1: *A fully balanced and complete tree corresponding to the Z-order curve. Nodes with a solid circle are essential, while the rest nodes redundant. Note that the tree is conceptual and does not be really constructed.*

irregularly distributed. For instance, nodes "000001" and "000011" as well as their ancestor nodes in Figure 1 do not correspond to any primitives. In fact, only those nodes experiencing a transition of the significant bit in the Morton codes when moving up to their parent nodes are essential. In Figure 1, essential nodes are marked with a solid circle, while redundant nodes are represented with a dotted circle. We proposed a path compression procedure to remove all redundant nodes from the conceptual complete binary tree and convert it into a kd-tree. Because the redundant nodes are deleted along paths connecting essential nodes, the resultant tree is called a path compressed kd-tree. The red lines in Figure 1 indicate the compressed path among essential nodes. The parent and child relation among nodes are determined by the underlying Morton codes. The method concurrently processes essential nodes at all leaves and enables maximized parallelism.

3. Results and Analysis

We tested our kd-tree construction algorithm on a NVIDIA GTX 580 GPU (Fermi architecture) with the code compiled with CUDA 5.0 Toolkit. The construction results were compared with a recent work by Wu et al. [2011]. Since the results of Wu et al. [2011] was reported on a GTX 280 GPU (also based on Fermi architecture) We use a performance divisor of 2.0 to translate the computing times measured on GTX280 and then compare them with the results on GTX580. The performance divisor is selected by measuring the performance of workloads with similar computing patterns on the two GPUs. All scenes were tested under a resolution of 1024×1024 with only primary rays. The comparison of construction time is reported in Table I. The results prove that our construction speed outperforms the previous work by over one order of magnitude.

Table I. *Kd-tree construction time compared with [Wu et al. 2011]*

Scene	Bunny	Fairy	Conference	Sibenik	Dragon	Buddha
Wu et al. (ms)	26	-	-	-	232	303
Our work (ms)	4.3	5.6	5.9	4.7	8.7	9.3

References

WU, Z., ZHAO, F., LIU, X. 2011. SAH KD-Tree Construction on GPU. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG '11, ACM, pp. 71–78.