Accelerating Adaptive Distance Fields Construction on GPUs via Memory Layout Optimization and Traversal Algorithm Enhancement

2230410 Daiki Taniguchi

This research aims to accelerate the construction of Adaptively Sampled Distance Fields (ADFs) on modern GPUs. Distance Fields are scalar fields fundamental to computer graphics and physical simulation, storing the shortest distance from any 3D point to an object's surface. While memory-efficient ADFs, which use hierarchical structures like octrees, are an important alternative to memory-intensive uniform grids, their non-uniform, irregular data structure presents a significant performance challenge for GPU parallel processing. GPUs are optimized for regular, coalesced memory access, making the tree-traversal operations required for ADFs a bottleneck. This study analyzes state-of-the-art GPU-based ADF construction methods, such as the Bounding Volume Hierarchy (BVH) based approach (BADF), and identifies that the distance query stage remains the primary performance bottleneck. This stage, which traverses a BVH to find the precise nearest distance for each ADF sample point, consumes over 68% of the total computation time, and for complex models like Dragon and Buddha, this figure exceeds 85%.

This thesis identifies two specific causes for this inefficiency. First, the inefficient memory layout of the baseline implementation, which uses a standard Array of Structures (AoS) layout for BVH nodes. In this layout, disparate data, such as the 24-byte AABB and, for example, 8-bytes of child indices, are stored together in a single 32-byte struct. During traversal, GPU threads often only need to read the AABB for distance checks. The AoS layout forces the GPU to load the entire 32-byte struct, wasting memory bandwidth on the unused child index data. This data arrangement also impedes memory coalescing. When a warp of 32 threads attempts to access the AABBs of 32 different nodes, their target data is scattered in memory (e.g., at 32-byte strides), preventing the GPU from merging these into a single, efficient transaction. Second, the inefficient traversal algorithm used in the baseline is a standard stack-based, depth-first search with a fixed child traversal order. This uniform approach (e.g., always pushing the left child, then the right) is suboptimal for distance queries. It can result in exploring geometrically distant nodes first, which delays finding a tight minimum distance bound. The efficiency of BVH traversal hinges on finding this tight bound as early as possible to prune large, irrelevant sections of the tree. The fixed-order search performs this pruning inefficiently, leading to a number of unnecessary node visits and distance calculations.

To resolve these bottlenecks, this thesis proposes and evaluates two complementary optimization techniques. The first technique addresses the mem-

ory access inefficiency by transitioning the BVH data from an AoS to a Structure of Arrays (SoA) layout. This approach physically separates the different data members into distinct, contiguous arrays: one array containing all AABBs, and another containing all child index pairs. This SoA layout (implemented as a hybrid SoA) ensures that when a warp of threads accesses AABBs, these accesses target contiguous memory locations. This is designed to restore memory coalescing, maximize memory bandwidth utilization, and improve cache efficiency, as cache lines are filled only with data of the type being accessed. A theoretical model developed in this work (Eqs. 3.1 & 3.2) supports this, showing that SoA transaction counts depend only on the accessed data size (S_{data}) rather than the full struct size (S_{struct}). For AABB-only access (24 bytes), this change is theoretically predicted to reduce memory transactions by 25% (from 8 to 6 transactions per coalesced access) compared to accessing the full 32-byte struct.

The second technique is an algorithmic enhancement to the traversal itself, named Closer Child Traversal (CCT). This optimization replaces the fixed-order search with a dynamic, distance-based search order, applying a Best-First Search principle. When at an internal node, the CCT algorithm computes the distance from the query point to the AABBs of both the left and right child nodes. It then compares these two distances and strategically pushes the farther child onto the stack first, followed by the closer child. Because the stack is a Last-In, First-Out (LIFO) data structure, the node that is processed in the next iteration of the loop is guaranteed to be the one closer to the query point. This dynamic ordering finds the true nearest polygon much earlier in the search. This updates the minimum distance bound ('minDistSq') sooner, which in turn allows the pruning check to be more effective, culling a larger number of unnecessary branches and reducing the total computational and memory access load.

These proposed methods were implemented and evaluated on an NVIDIA GeForce RTX 3080 GPU using four standard 3D models (Bunny, Armadillo, Dragon, and Buddha). The evaluation compared three versions: the 'Baseline' (AoS, fixed order), an 'SoA'-only version, and the final 'SoA + CCT' implementation. The results demonstrated the complementary effects of each optimization. Applying only the SoA layout provided a consistent performance improvement, making the kernel 1.37x to 1.47x faster than the baseline. This was attributed to superior memory efficiency; for the Dragon model, SoA reduced L1 Load Requests by 21 % (from 240.0M to 190.2M), which closely matches the 25% theoretical prediction. The addition of the CCT algorithm provided a further speedup. The final 'SoA + CCT' implementation was 4.46x to 6.08x faster than the original baseline in the distance query kernel. This major speedup was driven by a reduction in the quantity of memory accesses. For the Dragon model, the combined approach reduced the number of L1 Load Requests by 93% (from 240.0M down to 16.8M) and DRAM Read Sectors by 76% (from 83.0M to 19.7M).

By resolving the primary bottleneck, the entire ADF construction process was accelerated by up to 3.92x. The analysis also revealed an important trade-off: for some models (Bunny and Armadillo), the CCT's dynamic access pattern slightly increased DRAM reads compared to the SoA-only version, suggesting a minor reduction in cache locality. However, the overall execution time was still reduced, demonstrating that the benefit of algorithmically reducing the total number of work items (L1 requests) outweighed the minor negative impact on cache efficiency. This research confirms that a two-part strategy—optimizing the low-level data layout for the GPU architecture (SoA) and optimizing the high-level algorithm to reduce total work (CCT)—is effective and complementary for accelerating complex tree-traversal problems on GPUs.