Fast Closest Point Queries

Max Slater (Advisors: Rohan Sawhney, Keenan Crane)

THE GEOMETRY COLLECTIVE AT CARNEGIE MELLON UNIVERSITY

Abstract

A closest point query (CPQ) asks a simple question: relative to an input location, where is the closest point on some surface? Finding an answer is similar to intersecting a ray with the surface, a problem that prior work has greatly optimized both algorithmically and practically. However, finding a closest point exhibits different runtime behavior than finding a ray intersection, hence requires choosing a different set of tradeoffs. Through developing optimized thread/SIMD-parallel CPU and GPU implementations, we explore the best choices for closest points across different traversal strategies, tree widths, levels of input coherency, and primitive bounding volumes.

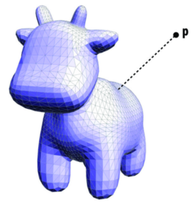


Figure 1. CPQ on a cow

Background

Prior research on fast ray intersection queries has focused on optimizing the construction and query behavior of bounding volume hierarchies (BVHs), which we build upon in this project. A BVH is a tree structure representing a collection of primitive shapes, where each sub-tree holds both a portion of the primitives and their enclosing bounding volume. For any given application, one may choose from a variety of hyper-parameters: node splitting heuristic, type of bounding volume, branching factor, leaf size, and traversal algorithm (stack-based, stack-less, wide, or threaded).

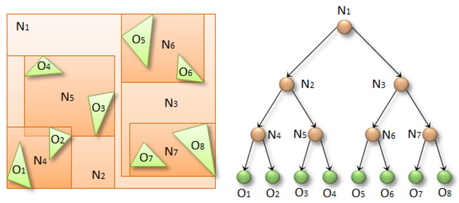


Figure 2. Binary BVH using axis-aligned bounding boxes

Applications

Signed Distance Functions: Closest points act as a bridge between the world of meshes and SDFs, which are themselves useful for sphere tracing and logical shape operations.

Monte Carlo Geometry Processing: the walk on spheres algorithm is used to estimate partial differential equations in geometric domains - each step computes a closest point in order to find the largest empty sphere centered at the query.

Collisions: closest points can be used for collision detection between primitives and meshes in physics simulations.

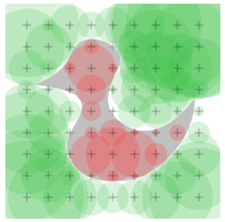
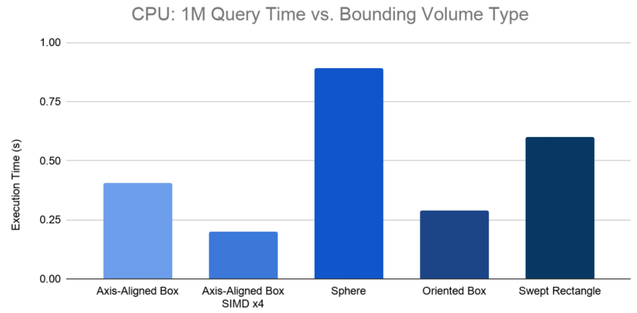


Figure 3. SDF Spheres

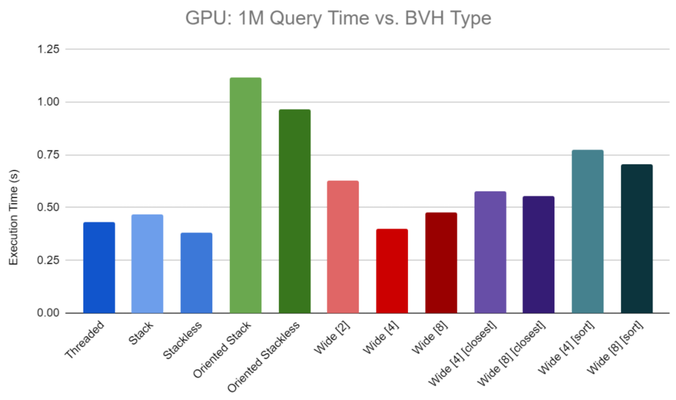
Results - CPU

Our CPU implementation computes closest point queries 2-8x faster than Embree, Intel's open-source BVH library. Our findings agreed with previous work on rays in some respects: optimizing for single point queries by building vectorized trees was more efficient than vectorizing queries themselves since traversals tend to diverge. Unlike for rays, we found that using more complex bounding volumes, such as oriented bounding boxes and sphere swept rectangles, gave performance benefits due to better primitive culling. Scaling via thread-based parallelism was linear as expected. Bench-marks were run on a 32-thread Ryzen 5950X @ 4.45GHz.



Results - GPU

On the GPU, vectorizing queries was inevitable, as were issues with divergence. We compared many different BVH traversal techniques, but concluded that all are bottle-necked by computing closest points on individual primitives, since even coherent queries may require checking large but only partially overlapping sets of primitives. We also found that while wide trees provide some benefit by reducing dependent memory accesses, using them to prioritize closer nodes was not worthwhile, as this decreases coherence. Rigid stack-less approaches give better results, since they enforce a traversal order while decreasing register pressure. Finally, except in cases of degenerate geometry, use of oriented bounding boxes hurt performance by increasing computation per node but not increasing coherence. Bench-marks were run on an RTX 3090.



Discussion

Current systems for ray queries make use of BVHs constructed from axis aligned bounding boxes, spatial splits, and SIMD vectorization for both internal nodes and primitives. GPU systems, on the other hand, focus on query vectorization and memory-efficient BVHs with high branching factors. These systems suffer from similar issues with divergence, but tend to exhibit very efficient culling during BVH traversal.

Our results across both the CPU and GPU show a contrasting theme for closest point queries: poor culling behavior. Because each branch of our BVH must be evaluated if its bounding volume is in range of our query, pruning becomes severely limited. This fact means many optimizations useful for rays are less successful for CPQs. Particularly, traversing much of the tree increases the cost of divergence for vectorized queries - and even nearby CPQs are not always coherent.

However, some options previously disregarded for ray intersections, such as more complex primitive bounding volumes, turn out to be useful for CPQs. On the CPU, we saw significant gains when using oriented boxes due to improved culling, and extra gains from vectorized trees when compared to rays (likely due to wasting fewer node queries). On the GPU, we found that while stack-less traversals and wide trees can improve performance, all approaches are similarly bottle-necked by divergent per-primitive closest point computations.

Future Work

Further work on this project involves transitioning our GPU implementation into a broadly useful library for both standalone closest point computations and integrations into other applications. Additionally, we plan on exploring the potential uses of hardware accelerated ray intersections for closest point queries. Other work has successfully applied hardware RT to problems such as tet-mesh point location and k-nearest-neighbor queries, so we believe further performance gains could be realized for closest points as well.

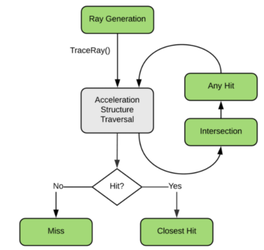


Figure 4. Hardware ray pipeline

References

[1] Manfred Ernst and Gunther Greiner.
Multi bounding volume hierarchies.
In *2008 IEEE Symposium on Interactive Ray Tracing*, pages 35–40, 2008.

[2] Michal Hapala, Tomáš Davidovič, Ingo Wald, Vlastimil Havran, and Philipp Slusallek.
Efficient stack-less bvh traversal for ray tracing.
In *Proceedings of the 27th Spring Conference on Computer Graphics*, SCCG '11, page 7–12, New York, NY, USA, 2011. Association for Computing Machinery.

[3] Evan Shellshear and Robin Ytterlid.
Fast distance queries for triangles, lines, and points using sse instructions.
Journal of Computer Graphics Techniques (JCGT), 3(4):86–110, December 2014.

[4] Henri Ylitie, Tero Karras, and Samuli Laine.
Efficient Incoherent Ray Traversal on GPUs Through Compressed Wide BVHs.
In Vlastimil Havran and Karthik Vaiyanathan, editors, *Eurographics/ High Performance Graphics*. ACM, 2017.

[5] Robin Ytterlid and Evan Shellshear.
Bvh split strategies for fast distance queries.
Journal of Computer Graphics Techniques (JCGT), 4(1):1–25, January 2015.