Fast Closest Point Queries

For Monte Carlo Geometry Processing

Max Slater
https://thenumbat.github.io/15400-s21/

1 Overview

I will be working with Professor Keenan Crane and his PhD student Rohan Sawhney, both of the Geometry Collective at CMU. My project is to explore algorithmic and practical optimizations for performing closest point queries, particularly in the context of Rohan & Keenan's most recent paper, *Monte Carlo Geometry Processing* (MCGP) [5].

Closest point queries, or CPQs, solve a straightforward problem: relative to an input location, where is the closest point on some surface? When working with, for example, triangulated surfaces, this is easy to compute by simply taking the minimum distance to each triangle. However, the naive process is dramatically slower than necessary, as we may compute closest points far faster than linearly in the input size. As it turns out, there are many disparate ways to optimize CPQ algorithms, both algorithmically and in practical implementation.

CPQs are a foundational tool in geometric computing, as many relevant algorithms depend on them, including but not limited to: collision detection and simulation, simulating distance-dependent forces, computing signed distance fields, sphere tracing implicit surfaces, and MCGP. All of these algorithms depend on CPQs as a basic primitive, so their performance depends heavily on the efficiency of computing millions or billions of CPQs. If we can produce a faster CPQ primitive implementation, the performance of many algorithms stand to benefit. However, despite their usefulness, fast CPQs are less studied or highly optimized than queries like ray-mesh intersections, which are now implemented in GPU hardware [4].

In the context of MCGP, the performance gains are particularly important, as CPQs are the basic primitive of the core walk-on-spheres algorithm, which iteratively jumps to a random location on the largest bounding sphere centered at each sample point. The radius of the bounding sphere may be determined by computing a CPQ on the boundary of the domain, the result of which is necessarily on the sphere.

Of course, as CPQs are not a new domain, there are previous fast methods for computing them. Most previous methods rely on constructing a bounding volume hierarchy (BVH) on the domain, which attempts to partition the primitive shapes into a tree structure that may be efficiently queried in logarithmic time. However, these methods are typically optimized for finding exact closest points, or harder queries like finding the closest pair of points on two surfaces, which is useful for collision detection. In MCGP, we do not require such a specific result, as there are several looser constraints. First, we do not require a closest point, but only a closest distance: a bounding sphere is defined by radius only. Second, we do not even require an exact closest distance, but rather a lower bound: a non-maximal bounding sphere will still produce correct results, if sacrificing some efficiency. Finally, the walk-on-sphere process iteratively approaches the boundary of the domain, so we many be able

to take advantage of caching effects due to computing closest distances from a convergent sequence of points.

The focus of this project will hence be investigating how we might be able to take advantage of the looser constraints of MCGP in order to improve the efficiency of CPQs. If successful, we will be able to improve the performance of MCGP, broadening its applicability and usefulness in geometric computing. Our first main point of inquiry will be in creating an optimized implementation of state-of-the-art BVH construction and query strategies on modern GPU hardware—particularly, we hope to produce a fast and useful BVH library for both CPQs and ray intersections, as no good option currently exists for CPQs. In particular, we will attempt to apply hardware ray-intersection cores, as well as multi-core and SIMD parallelism to accelerate CPQs on BVHs. This may be challenging, as GPU hardware, even RT cores, are not a priori specialized for CPQs, and are almost always limited by memory latency. Second, we will explore algorithmic improvements in constructing and querying BVH structures for the looser constraints of MCGP, for example computing conservative closest distances. Similarly, we may explore the use of various other data structures. Challenges in this stage may include finding genuinely useful BVH optimizations or other representations, as this topic is already reasonably well studied.

2 Goals

2.1 75% Goal

1. Implement and benchmark various state of the art BVH construction and query strategies.
2. Improve parallel scaling of current BVH implementations, matching state of the art implementations in ray intersection and exceeding in CPQs.
3. Porting BVH strategies to the GPU with similar scalability. Implementing relevant optimizations like stackless traversal, memory compression, etc.
4. Exploring the use of hardware RT cores to accelerate GPU BVH CPQs.

2.2 100% Goal

1. All 75% goals.
2. Using hardware RT cores to obtain significant speedup for CPQs.
3. Producing production quality BVH library running on the CPU and GPU. Library intends to be a standard tool for fast CPQs.

2.3 125% Goal

1. All 100% goals.
2. Building on BVH library to explore algorithmic optimizations for MCGP covered earlier.
3. Algorithmic systems optimizations for coherent queries, CPQ workload imbalance, caching, and estimates.
4. (Maybe) working on new types of queries for MCGP, such as off-center bounding spheres.

3 Milestones

3.1 15-300: First Technical Milestone

1. Complete background reading.
2. Become comfortable with current codebase/implementations.
3. Begin benchmarking current CPU implementations.

3.2 15-400: Feb 15

1. Complete CPU benchmarking.
2. Preliminary (or final) results regarding improving SIMD scaling.

3.3 15-400: Mar 1

1. Complete scaling work, obtain faster CPQs than current commercial implementation (Embree).
2. Begin work on GPU implementations and benchmarking.

3.4 15-400: Mar 15

1. Work on GPU implementations.
2. Begin exploring use of hardware RT. Hope to obtain publishable results in this area, as well as publishable software artifacts.

3.5 15-400: Mar 29

1. Complete GPU implementations, again match ray intersection and exceed CPQ state of the art performance.
2. Work on making implementations usable and robust.
3. Work on hardware RT implementation.

3.6 15-400: Apr 12

1. Complete hardware RT work.
2. Continue polishing library.
3. Start exploring new optimization strategies.

3.7 15-400: Apr 26

1. Work on optimizations for conservative distance bounds.
2. Work on optimizations for workload scheduling and cache optimization.
3. Hope to also obtain publishable results in these areas.
4. Continue polishing library.

3.8 15-400: May 10

1. Complete useful and robust library including all implementations.
2. Implement previously mentioned optimizations, exceeding state of the art.

3.9 Potential Summer

1. 125% Goals.
2. Continue polishing CPQ library.

4 Literature

I have been reading the papers given in the references section. They give an overview of current work in optimizing BVHs, as well as using hardware ray tracing to compute queries other than ray-mesh intersections. I am not currently lacking background in any particular area, but I will continue finding/reading literature on these subjects.

5 Resources

Relevant software is readily available (C++, OptiX/Embree/CUDA, Vulkan/D3D/GLSL, etc.). In order to implement and test a GPU implementation with hardware ray tracing, I will need a recent GPU with hardware RT cores, e.g. an NVIDIA RTX 3080 or 3070 (∼$750/$550). It is likely possible to access one remotely, but would be highly inconvenient for visualization/debugging and benchmarking in a controlled environment. Hence, hopefully I can be provided one directly. (If not, I'll get one for myself anyway...)

References

[1] Keenan Crane, Kayvon Fatahalian, Stelian Coros, Michael Choquette, Se-Joon Chung, Sky Gao, Qiuyi Jia, Zach Shearer, Bryce Summers, Nick Sharp, and Maxwell Slater. 15-462 course material, August 2020.

[2] Eric Galin, Eric Guérin, Axel Paris, and Adrien Peytavie. Segment tracing using local lipschitz bounds. *Computer Graphics Forum*, 39(2):545–554, 2020.

[3] 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.

[4] NVIDIA. Nvidia turing gpu architecture, 2018. https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf.

[5] Rohan Sawhney and Keenan Crane. Monte carlo geometry processing: A grid-free approach to pde-based methods on volumetric domains. *ACM Trans. Graph.*, 39(4), 2020.

[6] 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.

[7] Martin Stich, Heiko Friedrich, and Andreas Dietrich. Spatial splits in bounding volume hierarchies. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, page 7–13, New York, NY, USA, 2009. Association for Computing Machinery.

[8] Ingo Wald, Will Usher, Nate Morrical, Laura Lediaev, and Valerio Pascucci. RTX Beyond Ray Tracing: Exploring the Use of Hardware Ray Tracing Cores for Tet-Mesh Point Location. In *High-Performance Graphics - Short Papers*, 2019.

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