# Hardware Acceleration for Spatial Databases: Building Next-Generation Query Engines Using Ray Tracing Cores

Rubao Lee
The Ohio State University
lee.11875@osu.edu

Liang Geng
The Ohio State University
geng.161@buckeyemail.osu.edu

Xiaodong Zhang
The Ohio State University
zhang@cse.ohio-state.edu

## ABSTRACT

Modern database systems are increasingly shaped by domain-specific hardware, yet using such hardware effectively still requires new system abstractions, programming methods, and algorithmic reformulations. This tutorial presents a case study of hardware-accelerated spatial database systems. We explain how Ray Tracing (RT) cores in commodity GPUs, originally designed for graphics rendering, can be repurposed to accelerate key spatial data operations by treating spatial search and refinement tasks as ray–geometry intersection problems. The tutorial is centered on four recent projects from our group: RT-based spatial join, RT-based spatial indexing, RT-based Hausdorff distance computation, and the integration of RT acceleration into the industrial spatial database Apache SedonaDB. Together, these works provide a coherent path from hardware principles and RT programming, to query operators and index design, to real system deployment. The tutorial targets both researchers and practitioners who are interested in building next-generation query engines for spatial analytics on emerging hardware.

**PVLDB Reference Format:**
Rubao Lee, Liang Geng, and Xiaodong Zhang. Hardware Acceleration for Spatial Databases: Building Next-Generation Query Engines Using Ray Tracing Cores. PVLDB, 19(12): 4862 - 4866, 2026.
doi:10.14778/3827998.3828056

**PVLDB Artifact Availability:**
The source code, data, and/or other artifacts have been made available at https://rubaolee.github.io/vldb26-tutorial/index.html.

## 1 INTRODUCTION

Ray tracing hardware (e.g., RT cores in NVIDIA GPUs) has evolved rapidly from a graphics-oriented accelerator into a promising multi-purpose engine for data-intensive computing. RT cores accelerate two operations that are notoriously expensive in software: bounding volume hierarchy traversal and ray–primitive intersection testing [2]. These operations are not limited to rendering. They also resemble many geometric search and filtering tasks in spatial data processing [4, 27, 28, 32]. This observation opens a new opportunity for database research: instead of treating RT hardware as an unrelated graphics feature, we can reinterpret it as a specialized search engine for spatial query execution [4, 5, 13, 15, 32].


Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828056

**Table 1: The evolution of Ray Tracing (RT) cores across recent NVIDIA microarchitectures, comparing the flagship consumer and professional/workstation graphics cards for each generation along with their active RT core counts.**

| Architecture | Year | Consumer GPU | RT | Workstation GPU | RT |
|---|---|---|---|---|---|
| Turing [17] | 2018 | RTX 2080 Ti | 68 | Quadro RTX 8000 | 72 |
| Ampere [18] | 2020 | RTX 3090 | 82 | RTX A6000 | 84 |
| Ada Lovelace [19] | 2022 | RTX 4090 | 128 | RTX 6000 Ada | 142 |
| Blackwell [20] | 2025 | RTX 5090 | 170 | RTX PRO 6000 | 188 |

Spatial database systems are a compelling domain for this direction. Their workloads are dominated by geometric predicates, spatial indexing, spatial joins, and distance computations. These operators are computationally intensive, irregular, and increasingly important in applications such as GIS analytics, digital twins, urban computing, location intelligence, and embodied AI [7, 8, 23, 25, 26, 30, 31]. At the same time, commodity GPUs already ship with increasingly powerful ray tracing cores, as shown in Table 1. This makes RT-based acceleration both technically attractive and practically relevant.

**An Overview of the Tutorial.** The tutorial can be structured as either a 1.5-hour or a 3-hour session, depending on the level of detail included in the presentation. We begin by introducing the fundamentals of ray tracing, including its underlying concepts, hardware accelerators, and programming models, using NVIDIA OptiX [21] as a concrete example. The core of the tutorial consists of four connected case studies: ray tracing for spatial joins, ray tracing for spatial indexing, ray tracing for spatial functions (using Hausdorff distance as an example), and the integration of ray tracing into Apache SedonaDB [1]. Finally, we discuss broader research opportunities beyond spatial data processing.

**Target Audience.** We target a broad audience of database researchers, systems researchers, GPU programmers, and practitioners interested in building next-generation query engines on modern hardware. Students at both undergraduate and graduate levels, as well as junior colleagues in academia and industry, are welcome. The tutorial is also relevant to closely related communities such as data mining, knowledge discovery, computer graphics systems, and GIS. We assume only a basic background in database systems or GPU programming, and no prior experience with graphics applications is required. To make RT-based database acceleration accessible to a general VLDB audience, we use intuitive geometric examples and simplified code fragments throughout the tutorial.

## 2 RELATED MATERIALS

**Related Tutorials.** To the best of our knowledge, this is the first tutorial at a premier database venue that systematically introduces

how to build spatial database engines using ray tracing hardware. While recent database tutorials have discussed hardware-conscious query processing, heterogeneous systems, and GPU acceleration in general [11, 16, 22], they have not covered the RT programming model, RT-specific algorithm reformulation, or the end-to-end path from prototype research to database system integration. The unique value of this tutorial lies in its comprehensive coverage of hardware principles, programming methods, query operators, indexing, advanced functions, and industrial adoption in one coherent story.

**Related Work from Authors.** The authors have published multiple RT-based systems papers in recent years, covering RT-based acceleration for relational databases [12, 24], graph applications [28], and spatial databases, including spatial joins [4], a general RT-based spatial indexing library [5], and Hausdorff distance computation [6]. The authors have also integrated these research results into SedonaDB as an industrial RT engine [3]. Collectively, these works span algorithm design, GPU systems engineering, software abstraction, and system deployment. This tutorial builds on these efforts to present a broader methodology for transforming RT hardware into a practical accelerator for spatial database systems.

# 3 TUTORIAL OUTLINE

## 3.1 RT Programming

We begin with the RT hardware execution model and the NVIDIA software stack, including BVH construction, ray generation, hit processing, and programmable callbacks. The goal is not to teach graphics rendering, but to show database researchers how to express spatial search problems in RT terms. We will explain the difference between using RT cores as a low-level primitive and using them as a database building block. Particular attention will be given to data layout, precision issues, callback overhead, and the interplay between RT cores and conventional GPU cores.

## 3.2 RT for Spatial Join

The first case study is the spatial join. We explain how Point-In-Polygon (PIP), Line-Segment Intersection (LSI), and polygon overlay can be reformulated as ray tracing problems. The tutorial will use our RayJoin [4] work to show how LSI and PIP can be mapped onto ray casting and intersection search, and how high-precision requirements in spatial databases differ from graphics rendering.

*3.2.1 RT-based LSI.* RayJoin maps the LSI problem directly into the native hardware abstractions of GPU ray tracing:

**Building the BVH.** Line segments on one side (base map) are wrapped in Axis-Aligned Bounding Boxes (AABBs) to build a hardware-accelerated Bounding Volume Hierarchy (BVH). The segments in another set (query map) are treated as rays.

**Exploiting the AnyHit Shader.** A single query segment is mapped to a ray, which might intersect multiple segments in the base map. To capture all of them, RayJoin exploits the `AnyHit` shader. Instead of stopping the ray at the first collision, the `AnyHit` shader allows the hardware to report a hit and keep going. For every AABB collision detected, it triggers a high-precision mathematical check to verify if the actual segments intersect, finding all overlapping pairs in a single pass.

*3.2.2 RT-based PIP.* For PIP queries, RayJoin uses an efficient adaptation of ray tracing to determine if a point is bounded by a polygon:

**Building the BVH.** As before, each line segment is mapped to an AABB, and a BVH is built from the AABBs' bounding line segments of the base map.

**Exploiting the Closest-Hit Shader.** A ray is cast from the query point outward in a single direction. Instead of checking every single boundary crossing (like the traditional even-odd rule), RayJoin exploits the `ClosestHit` shader. By identifying the very first (closest) boundary edge the ray hits, the algorithm can check the facing direction of that specific edge. This allows the GPU to resolve the query by stopping at the first valid collision.

*3.2.3 Overcoming Precision Limitation.* Because RT cores are designed for rendering graphics, their hardware intersection units natively operate using single-precision floating-point (FP32) arithmetic. For spatial databases, this introduces rounding errors that can cause the GPU to miss boundary collisions (false negatives). RayJoin solves this using a hardware-software hybrid approach:

**AABB Padding.** To guarantee the RT hardware never misses a true intersection due to FP32 rounding errors, RayJoin slightly inflates the bounding boxes (AABBs) by a tiny epsilon value before building the BVH. This forces the hardware to be overly cautious, ensuring all actual overlapping segments are caught.

**Robust Predicates (Software Refine).** The RT hardware is only used to find potential overlaps. Once a collision is triggered, the custom shaders abandon the hardware's FP32 math entirely. Instead, they execute precise geometric predicates to confirm the intersection, ensuring the final spatial query is accurate.

## 3.3 RT for Spatial Indexing

The second case study focuses on indexing. RT hardware naturally accelerates BVH traversal, but using it effectively for database workloads requires a suitable abstraction. We present the design ideas behind a general-purpose RT spatial indexing library, LibRTS [5], which supports point and range queries and mutable datasets.

*3.3.1 Mapping Spatial Queries to Ray Tracing Paradigms.* To execute spatial lookups on RT cores, spatial queries must be reformulated into the native language of the RT pipeline: ray and geometry. Geometries are wrapped in AABBs to construct a BVH. Subsequently, queries are converted into different forms of rays.

**Point Queries.** A point query translates into a single short ray that traverses the BVH to identify overlapping primitives.

**Range Queries.** Because rays lack spatial extent, LibRTS resolves bounding box queries using a two-pass diagonal intersection method. It first casts the query box's diagonal as a ray against indexed boxes, then casts rays derived from the indexed boxes' anti-diagonals against the query boxes, effectively tricking RT cores into multidimensional range searches.

*3.3.2 Addressing Load-balancing Challenge.* When repurposing RT cores for spatial queries, highly skewed geometric distributions cause a severe load-balancing challenge. Because RT cores rely on a single-ray programming model, all intersections for a ray are processed by one thread. This causes some threads to process many intersections while others handle very few. Standard GPU workload

rebalancing techniques cannot resolve this because the framework restricts thread block synchronization.

To resolve this, LibRTS introduces a static load-balancing method called "Ray Multicast". This technique evenly splits $N$ primitives into $k$ sets and distributes them into $k$ non-overlapping sub-spaces. A query ray is then duplicated into $k$ rays, with each ray responsible for finding intersections in just one sub-space. By having $k$ threads cast these $k$ rays, the maximum number of intersections any single thread must process is reduced to $\frac{|N|}{k}$.

*3.3.3 Handling Dynamic Data.* To support mutability without costly full rebuilds, LibRTS uses a two-level "Instancing" structure. Newly inserted geometries are built into smaller bottom-level Geometry Acceleration Structures (GASs), which are subsequently linked by a top-level Instance Acceleration Structure (IAS). For updates, LibRTS leverages OptiX's built-in capability to modify primitive coordinates within the existing structure. Deletions are efficiently handled by converting removed geometries into "degenerate cases"—setting their minimum and maximum coordinates to identical values. This shrinks their bounding boxes to zero, rendering them invisible to ray-casting queries without requiring complex structural removal from the tree.

## 3.4 RT for Spatial Functions (Case Study: Hausdorff Distance)

The third case study demonstrates how RT can be used to accelerate spatial functions that fundamentally rely on nearest neighbor search. As a representative example, we consider the standard PostGIS function $ST\_Hausdorff Distance$, which computes the Hausdorff distance (HD) between two geometries.

*3.4.1 Algorithm Overview and Spatial Partitioning.* HD measures the maximum discrepancy between two point sets, a process that is highly computationally intensive for large spatial datasets. The computation of HD is essentially a process of finding the nearest neighbors of points. While RT cores have been used to accelerate the underlying nearest-neighbor search [13, 15, 32], naively assigning each data point to its own AABB creates an excessively large BVH tree, leading to massive traversal overhead. To solve this, our solution X-HD implements a uniform grid to group spatially proximal points into shared cells. By expanding the Minimum Bounding Rectangle (MBR) of each non-empty cell by a search radius to define the AABBs, X-HD drastically reduces the BVH size, resulting in faster tree construction and fewer intersections.

*3.4.2 Pruning Non-Contributing Points.* X-HD optimizes performance by pruning redundant computations. It leverages HD estimators to calculate the upper and lower bounds of the distance between a given query point and a cell's MBR. If the upper bound of this distance is less than or equal to the current global maximum distance, the query point cannot increase the final HD value and is entirely discarded. Conversely, if the lower bound exceeds the query point's currently discovered minimum distance, the algorithm safely skips all points within that MBR. Furthermore, X-HD incorporates an Early Break (EB) strategy during cell traversal. If a calculated point-to-point distance is less than or equal to the global maximum distance, the inner search loop terminates immediately.

*3.4.3 Addressing Load-balancing issue within Shaders.* While the uniform grid reduces BVH complexity, skewed real-world data distributions create severe load imbalances, resulting in "heavy" cells containing tens of thousands of points. Because RT shaders utilize a single-ray programming model lacking synchronization primitives, processing these dense cells serially leads to poor hardware utilization. X-HD mitigates this by selectively offloading the distance computations of heavy cells from the RT shaders to a dedicated CUDA kernel. In this dedicated kernel, thread blocks cooperatively process the heavy cells to restore load balance.

## 3.5 RT in Action: Integrating RT into SedonaDB

The fourth case study discusses system integration in SedonaDB. This section focuses on engineering questions that matter to the VLDB community: how to connect RT acceleration with SQL execution, how to handle geometry layout on GPUs, how to avoid excessive BVH construction overhead, and how to support a wide range of spatial predicates. In this part, we explain RayBooster [3], including the design of a monolithic BVH, the use of RT cores inside a unified refinement engine, and the lessons from integrating RT acceleration into a production-oriented spatial database stack.

*3.5.1 Mapping Database Queries to Ray Tracing Problems.* To effectively harness RT cores, RayBooster translates standard geospatial operations into native ray-tracing tasks. During the filtering phase, spatial queries are modeled as rays cast against a hardware-accelerated BVH to detect overlapping bounding boxes. For the refinement phase, RayBooster introduces a unified RelateEngine that casts rays against individual line segments to compute the Dimensionally Extended 9-Intersection Model (DE-9IM), serving as a universal topological descriptor. Additionally, because RT frameworks are inherently built for 3D space, RayBooster employs a novel "Z-Stacking" technique. It places 2D geometries on individual X-Y planes and encodes their unique feature IDs into the unused Z-axis, allowing the system to build a single, monolithic BVH without overwhelming the GPU.

*3.5.2 Performance and Cost Analysis.* Offloading these massive geometric calculations to RT cores fundamentally transforms the engine's throughput. With this integration, SedonaDB not only runs faster but also reduces operational costs. Benchmark evaluations show that RayBooster's RT-driven architecture achieves up to a 5.93× performance speedup over the standard CPU baseline, which directly translates to a 59.02% reduction in operational cloud computing costs. Ultimately, RayBooster demonstrates that repurposing RT units is a highly efficient and cost-effective strategy for scaling industrial spatial analytics.

# 4 BEYOND SPATIAL

Although this tutorial focuses on spatial databases, the presented ideas have implications beyond one application domain. Recent work has shown that RT hardware can also accelerate relational database indexing [9, 10], SQL query engines [12, 24], nearest-neighbor search [13, 15, 32], and other irregular traversal workloads [14, 15, 29]. We briefly discuss these developments to help the audience understand the broader significance of RT hardware in data

systems. At the same time, we will be explicit about current limitations, including the difficulty of programming RT frameworks, hardware-specific constraints, precision management, and the challenge of integrating RT-based operators into full database systems. These limitations motivate future work on abstractions, compiler support, and cross-domain system design.

## 5 SCHEDULE AND PRESENTERS

The schedule for the 1.5-hour session is organized as follows:

- **Background and RT Programming (15 minutes):** Introduction to ray tracing concepts and programming models.
- **Spatial Join Acceleration (15 minutes):** Techniques for leveraging RT cores to accelerate spatial joins.
- **Spatial Indexing (15 minutes):** Designing and implementing RT-friendly spatial indexes.
- **Hausdorff Distance Computation (15 minutes):** A case study on accelerating complex spatial functions.
- **SedonaDB Integration (15 minutes):** Practical solutions from integrating RT into an industrial spatial database.
- **Discussion and Q&A (15 minutes):** Exploring broader opportunities and addressing audience questions.

**Rubao Lee** is a Senior Research Scientist in the Department of Computer Science and Engineering at The Ohio State University. His research focuses on database system optimization and parallel algorithm engineering. He has led multiple R&D efforts in building high-performance, GPU-accelerated database systems.

**Liang Geng** is a Ph.D. candidate in the Department of Computer Science and Engineering at The Ohio State University. His recent research bridges the gap between ray tracing hardware and spatial database workloads. This work has led to key publications on RT-based spatial joins, spatial indexing, advanced spatial functions, and their practical integration into Apache SedonaDB.

**Xiaodong Zhang** is a Professor in the Department of Computer Science and Engineering at The Ohio State University. His research spans data management, scalable systems, and computer architecture support for data-intensive computing. Numerous innovations from his research group have been successfully adopted in commercial processors, operating systems, and enterprise data systems.

## ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their constructive comments and suggestions. The work is supported in part by the U.S. National Science Foundation under grants CCF2005884, CCF-2210753, CCF2312507, and OAC-2310510.

## REFERENCES

[1] Apache Software Foundation. 2025. *Apache Sedona-db Documentation.* https://sedona.apache.org/sedonadb/latest/ Accessed: 2026-01-12.
[2] John Burgess. 2020. Rtx on—the nvidia turing gpu. *IEEE Micro* 40, 2 (2020), 36–44.
[3] Liang Geng, Rubao Lee, Dewey Dunnington, Feng Zhang, Jia Yu, and Xiaodong Zhang. 2026. RayBooster: A Ray Tracing Engine to Accelerate SedonaDB. *Proceedings of the VLDB Endowment* 19, 12 (2026).
[4] Liang Geng, Rubao Lee, and Xiaodong Zhang. 2024. RayJoin: Fast and Precise Spatial Join. In *Proceedings of the 38th ACM International Conference on Supercomputing*. 124–136. https://doi.org/10.1145/3650200.3656610
[5] Liang Geng, Rubao Lee, and Xiaodong Zhang. 2025. LibRTS: A Spatial Indexing Library by Ray Tracing. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. 396–411. https://doi.org/10.1145/3710848.3710850
[6] Liang Geng, Zhehu Yuan, Rubao Lee, Fusheng Wang, and Xiaodong Zhang. 2026. X-HD: Fast Hausdorff Distance Computation with Ray Tracing. In *Proceedings of the 40th ACM International Conference on Supercomputing (ICS '26)*. Association for Computing Machinery, New York, NY, USA, 945–959. https://doi.org/10.1145/3797905.3800509
[7] Michael Grieves et al. 2014. Digital twin: manufacturing excellence through virtual factory replication. *White paper* 1, 2014 (2014), 1–7.
[8] Agrim Gupta, Silvio Savarese, Surya Ganguli, and Li Fei-Fei. 2021. Embodied intelligence via learning and evolution. *Nature communications* 12, 1 (2021), 5721.
[9] Justus Henneberg and Felix Schuhknecht. 2023. RTIndeX: Exploiting Hardware-Accelerated GPU Raytracing for Database Indexing. *Proc. VLDB Endow.* 16, 13 (Sept. 2023), 4268–4281. https://doi.org/10.14778/3625054.3625063
[10] Justus Henneberg, Felix Schuhknecht, Rosina Kharal, and Trevor Brown. 2025. More Bang For Your Buck (et): Fast and Space-efficient Hardware-accelerated Coarse-granular Indexing on GPUs. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE, 1320–1333.
[11] Kaisong Huang, Yuliang He, and Tianzheng Wang. 2022. The past, present and future of indexing on persistent memory. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3774–3777.
[12] Yangming Lv, Kai Zhang, Ziming Wang, Xiaodong Zhang, Rubao Lee, Zhenying He, Yinan Jing, and X. Sean Wang. 2024. RTScan: Efficient Scan with Ray Tracing Cores. *Proc. VLDB Endow.* 17, 6 (Feb. 2024), 1460–1472. https://doi.org/10.14778/3648160.3648183
[13] Durga Keerthi Mandarapu, Vani Nagarajan, Artem Pelenitsyn, and Milind Kulkarni. 2024. Arkade: k-Nearest Neighbor Search With Non-Euclidean Distances using GPU Ray Tracing. In *Proceedings of the 38th ACM International Conference on Supercomputing*. 14–25. https://doi.org/10.1145/3650200.3656601
[14] Vani Nagarajan, Rohan Gangaraju, Kirshanthan Sundararajah, Artem Pelenitsyn, and Milind Kulkarni. 2025. RT-BarnesHut: Accelerating Barnes-Hut Using Ray-Tracing Hardware. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. 43–56.
[15] Vani Nagarajan, Durga Mandarapu, and Milind Kulkarni. 2023. RT-kNNS Unbound: Using RT Cores to Accelerate Unrestricted Neighbor Search. In *Proceedings of the 37th International Conference on Supercomputing*. 289–300. https://doi.org/10.48550/arXiv.2305.18356
[16] Marcos Nogueira Lobo de Carvalho, Alkis Simitsis, Anna Queralt Calafat, and Óscar Romero Moral. 2024. Workload placement on heterogeneous CPU-GPU systems. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4241–4244.
[17] NVIDIA Corporation. 2018. *NVIDIA Turing GPU Architecture.* Whitepaper. NVIDIA Corporation. https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf Accessed: 2026-01-12.
[18] NVIDIA Corporation. 2020. *NVIDIA Ampere GA102 GPU Architecture.* Whitepaper. NVIDIA Corporation. https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf Accessed: 2026-01-12.
[19] NVIDIA Corporation. 2022. *NVIDIA Ada GPU Architecture.* Whitepaper. NVIDIA Corporation. https://images.nvidia.com/aem-dam/Solutions/geforce/ada/nvidia-ada-gpu-architecture.pdf Accessed: 2026-01-12.
[20] NVIDIA Corporation. 2025. *NVIDIA RTX Blackwell GPU Architecture.* Whitepaper. NVIDIA Corporation. https://images.nvidia.com/aem-dam/Solutions/geforce/blackwell/nvidia-rtx-blackwell-gpu-architecture.pdf Accessed: 2026-01-12.
[21] Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, et al. 2010. Optix: a general purpose ray tracing engine. *Acm transactions on graphics (tog)* 29, 4 (2010), 1–13.
[22] Pedro Pedreira, Deepak Majeti, and Orri Erling. 2024. Composable Data Management: An Execution Overview. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4249–4252.
[23] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. 2019. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF international conference on computer vision*. 9339–9347.
[24] Xuri Shi, Kai Zhang, X Sean Wang, Xiaodong Zhang, and Rubao Lee. 2025. RayDB: Building Databases with Ray Tracing Cores. *Proceedings of the VLDB Endowment* 19, 1 (2025), 43–55.
[25] Sanjay Somanath, Vasilis Naserentin, Orfeas Eleftheriou, Daniel Sjölie, Beata Stahre Wästberg, and Anders Logg. 2024. Towards urban digital twins: A workflow for procedural visualization using geospatial data. *Remote Sensing* 16, 11 (2024), 1939.
[26] Fei Tao, He Zhang, Ang Liu, and Andrew YC Nee. 2018. Digital twin in industry: State-of-the-art. *IEEE Transactions on industrial informatics* 15, 4 (2018), 2405–2415.

[27] Ingo Wald, Will Usher, Nathan Morrical, Laura Lediaev, and Valerio Pascucci. 2019. RTX beyond ray tracing: Exploring the use of hardware ray tracing cores for tet-mesh point location. *High Performance Graphics (Short Papers)* 7 (2019), 13.

[28] Zhixiong Xiao, Mengbai Xiao, Yuan Yuan, Dongxiao Yu, Rubao Lee, and Xiaodong Zhang. 2025. A case study for ray tracing cores: Performance insights with breadth-first search and triangle counting in graphs. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 9, 2 (2025), 1–25.

[29] Hongrui Zhang, Yunan Zhang, and Hung-Wei Tseng. 2025. RTSpMSpM: Harnessing Ray Tracing for Efficient Sparse Matrix Computations. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*. 359–373.

[30] Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. 2014. Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 5, 3 (2014), 1–55.

[31] Yu Zheng, Xing Xie, Wei-Ying Ma, et al. 2010. GeoLife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.* 33, 2 (2010), 32–39.

[32] Yuhao Zhu. 2022. RTNN: accelerating neighbor search using hardware ray tracing. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 76–89. https://doi.org/10.1145/3503221.3508409