RasterDB: Implementing an Efficient Index Update Pipeline

Akash Maji, Database Systems Lab
SR No: 24212
Department of Computer Science and Automation
Indian Institute of Science, Bangalore - 560012

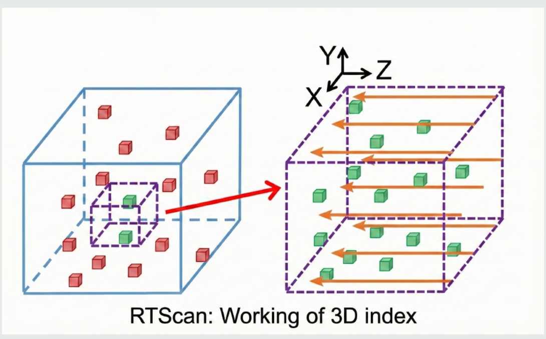
Overview

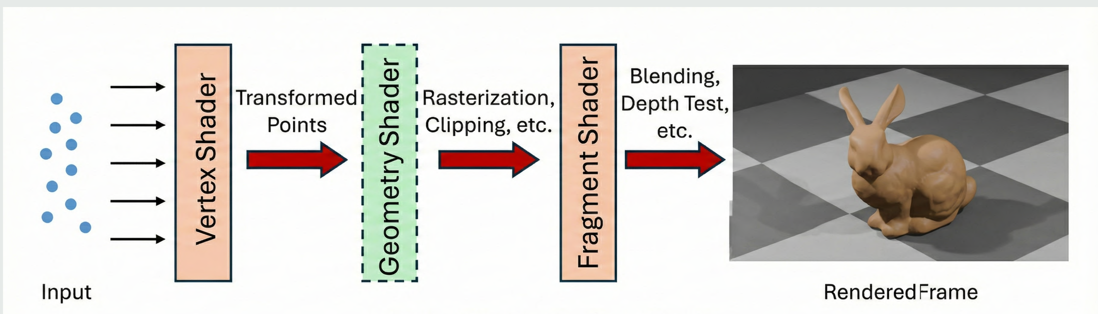
- GPU rasterization pipelines enable efficient database column indexing using simpler arithmetic comparisons than ray tracing–based geometric-intersection methods.
- We introduce **CompactScanIndex**, a GPU-accelerated index that supports dynamic point insertions and deletions on top of static index **RasterScanIndex**.

Introduction

- Recent GPU architectural advances enable massive parallelism and efficient memory access, making GPUs well-suited for data-intensive database workloads.
- Prior GPU-based indexing techniques [1], [2] focus on index construction and query execution, but require full rebuilds to handle data updates, which are expensive.
- We demonstrate that GPUs can efficiently support dynamic index updates using a uniform bin-based layout with atomic operations.

Background





Typical Rasterization Pipeline

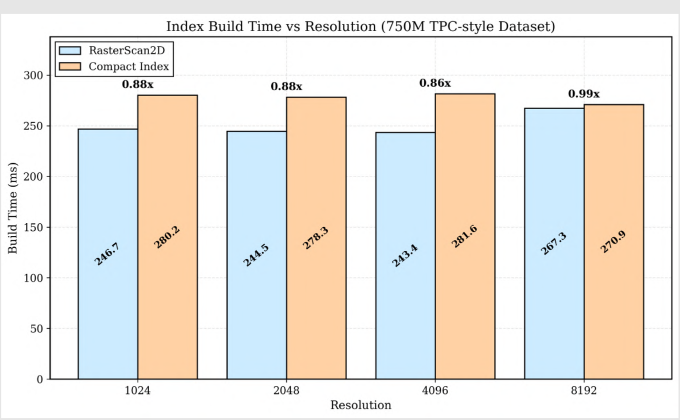
RTScan [2]. 3D Points modelled as *cubes* in 3D data space. Shoot parallel *rays* into query region (ray-tracing). Need costly BVH traversal. Inefficient for database indexing.
RasterScan [1]. 3D Points in 2D grid modelled as image. Replace *rays* with *lines* drawn through rasterization. Efficient due to well-behaved data organization.

Experimental Testbed

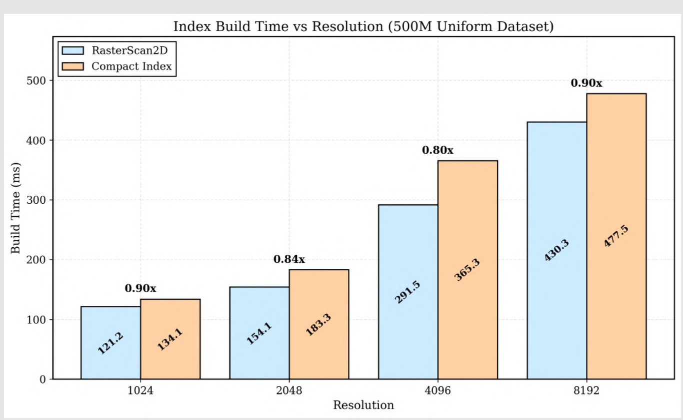
- **Evaluation Scope:** Comparison of CompactScan [3] against RasterScan [4]
  - *Three Integer Columns:* Datasets with 3 integer columns. $D = \{(x_i, y_i, z_i) \mid i = 1, \ldots, N\}$
  - *Batch Operations:* Updates in batches. Single-point updates are supported but costlier.
  - *Synthetic Data:* Uniform, Normal, Zipf, and TPC-C–style datasets at multiple sizes.
  - *Queries:* Support 3D range queries of form $0 <= x <= 10, 10 <= y <= 20, 20 <= z <= 30$. Ten queries $Q1, Q2, \ldots Q10$ at selectivity $10\%, 20\%, \ldots 100\%$ respectively tested. (Selectivity means percentage of total rows filtered. $Q5$ returns N/2 rows, $Q10$ returns all N rows)

Static Index Build Times and Query Times

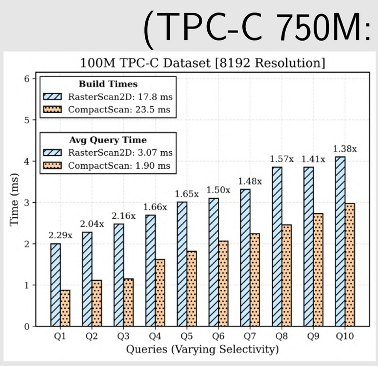
- We compare index build times (median of 11 runs) at multiple index resolutions R.
- Build times are slightly *higher* due to additional buffers and intermediate steps.
- **Extent-based pruning** *skips* empty bins, **reducing** atomic operations.
- *CompactScan* is **faster** than *RasterScan* on range queries at varying query selectivities.
- *CompactScan* is **suitable** for static scenarios with slight index build overhead.



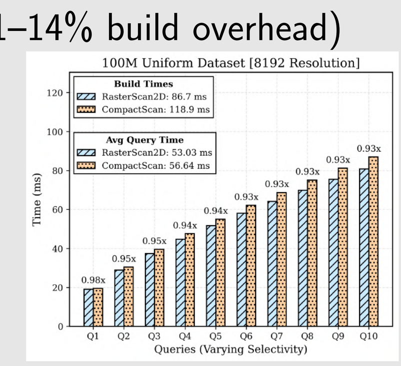
(TPC-C 750M: 1–14% build overhead)



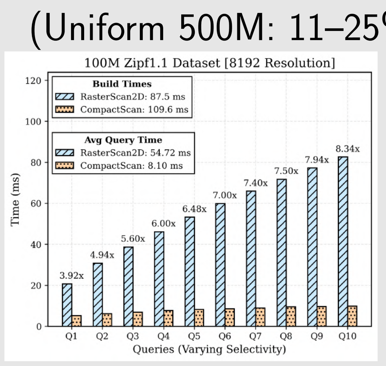
(Uniform 500M: 11–25% build overhead)



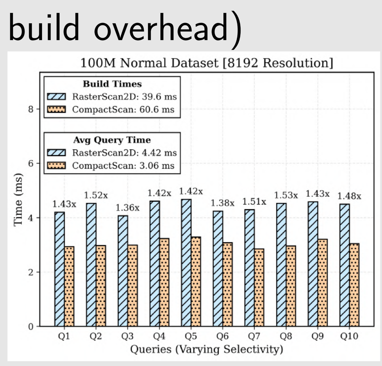
(TPC-C 100M)
1.60x Avg Speedup



(Uniform 100M)
0.95x Avg Speedup

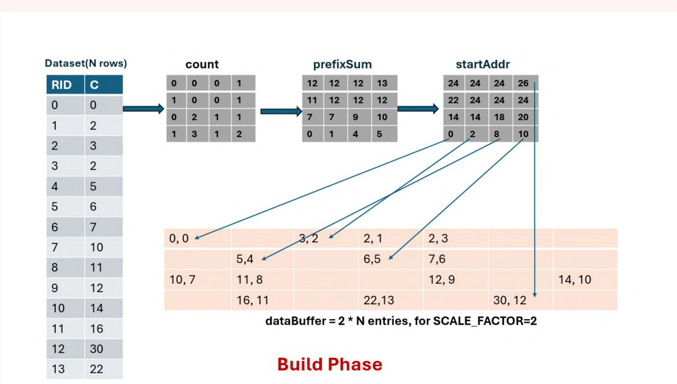


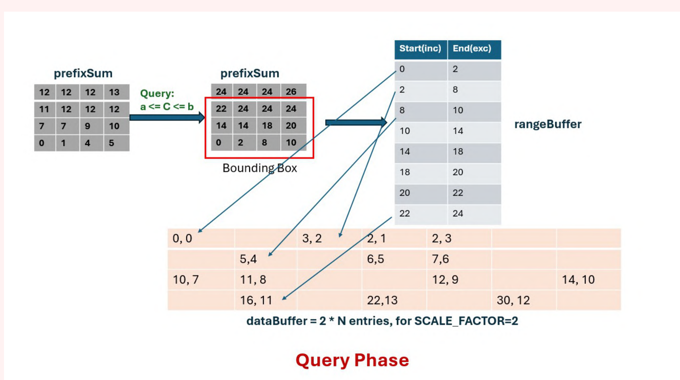
(Zipf 100M)
6.75x Avg Speedup



(Normal 100M)
1.45x Avg Speedup

Visual Working





- SCALE_FACTOR is a hyperparameter for extra space per bin to support updates.
- INDEX_RESOLUTION is a hyperparameter to control bin occupancy.

Implementation Details

- **Index Structure:** Partition 3D data into $R \times R$ bins. R is INDEX_RESOLUTION. Each point (x, y, z) maps to bin (b_x, b_y) where:

$$b_x = \left\lfloor \frac{(x - x_{min}) \cdot R}{x_{max} - x_{min} + 1} \right\rfloor, \quad b_y = \left\lfloor \frac{(y - y_{min}) \cdot R}{y_{max} - y_{min} + 1} \right\rfloor \tag{1}$$

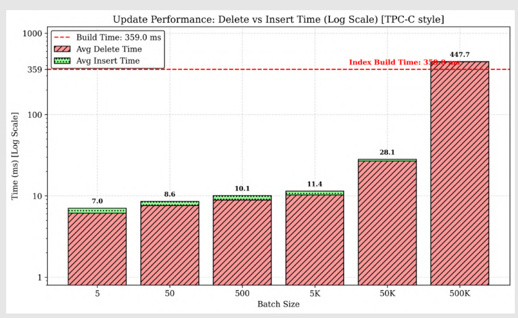
  The index maintains the following GPU buffers:
  startAddrBuffer: Stores starting offset of each bin in the data buffer.
  capacityBuffer : Stores maximum capacity per bin. Used for *overflow* detection.
  extentBuffer : Stores current number of used entries per bin.
  dataBuffer : Stores all indexed points (x, y, z, rowID) as entries.
- **Build Pipeline:** Build the index structure on GPU with input data.
  Pass 1: Histogram (Count Pass): Compute #points mapped per bin.
  Pass 2: Prefix Sum: Compute prefix sum of the counts.
  Pass 3: Buffer Setup: Copy offsets for bin data space.
  Pass 4: Data Insertion (Build Pass): Insert points into bins.
- **Query Pipeline:** Run range queries.
  Pass 1: Box detection: Obtain a 2D rectangle containing all bins.
  Pass 2: Result collection: Process non-empty bins. Collect resulting points.
- **Update Pipeline:** Supports in-place updates. Separate delete and insert operations.
  Delete Operation: Logical invalidation. Supports future compaction. Scans bin space.
  Insert Operation: Uses the same insert pass as the Build Pipeline.

Dynamic Index Update Performance

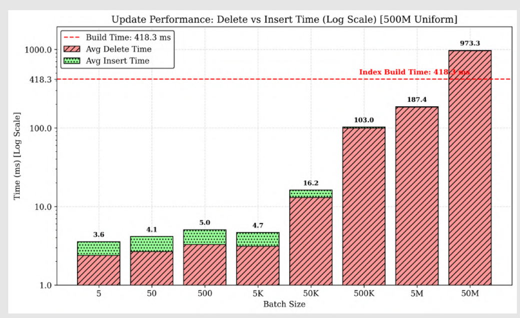
Table 3: Index Update Speedup Against Index Build

Batch Size	TPC-C	Normal	Uniform
5	51×	187×	118×
50	42×	154×	101×
500	36×	42×	83×
5,000	31×	16×	90×
50,000	13×	14×	26×

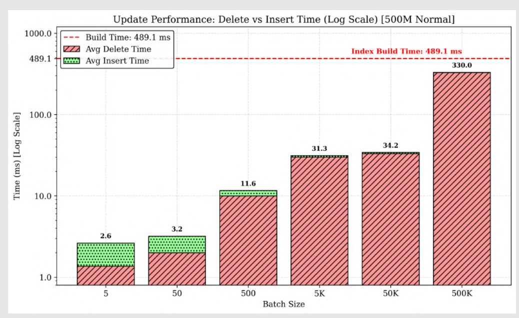
- Random batches of different sizes.
- Update is modelled as delete followed by insert.
- **CompactScan** supports deletions via linear-time $O(bin\_size)$ bin scans and constant-time $O(1)$ insertions.
- Optional compaction to reclaim space.



A.(TPC-C 500M)



B.(Uniform 500M)

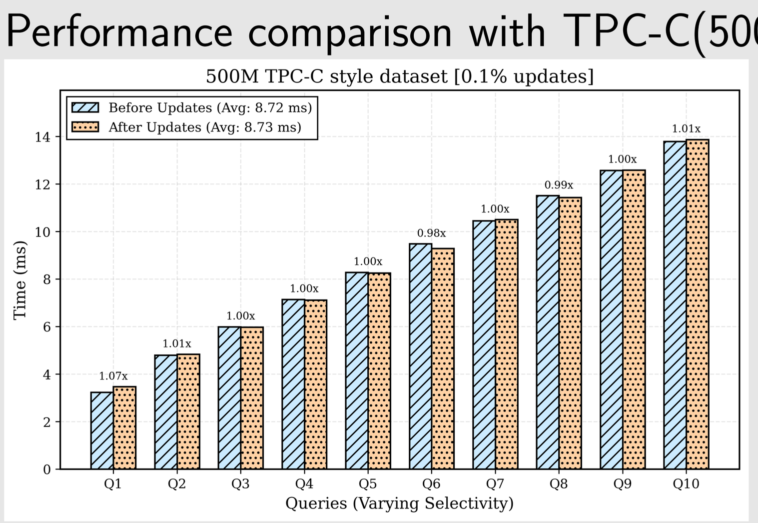


C.(Normal 500M)

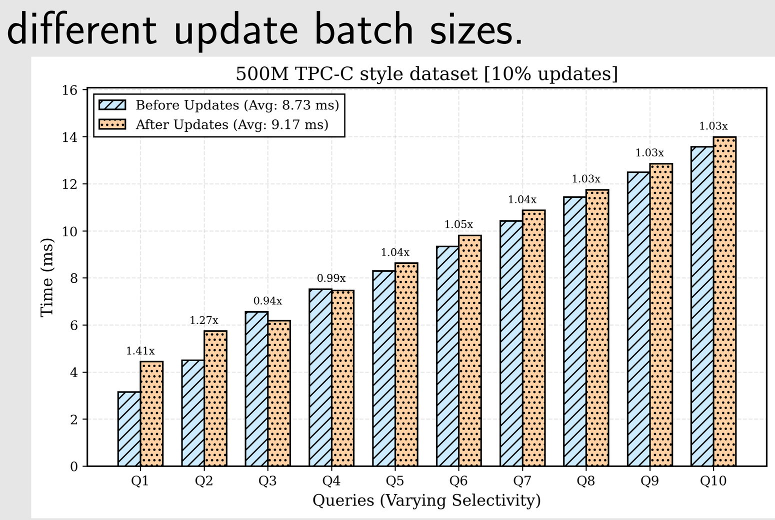
Observation. CompactScan achieves substantial speedups over full index rebuilds, especially for small and medium batch sizes. For bigger delete batches, not suitable.

Query After Updates

Performance comparison with TPC-C(500M) at different update batch sizes.



(500K updates)



(50M deletes)

Analysis.

- Updates cause a modest 3 to 5% query time increase for most queries.
- CompactScan maintains stable query performance even after 10% updates.
- The observed variations ($\pm 5\%$) in average query times seem acceptable.

Future Work

- *Adaptive Binning:*
  - **Equi-Depth Histograms:** Bins with approximately equal numbers of elements.
  - **Multi-Resolution Binning:** Hierarchical bins with finer granularity in dense regions.
- *Extended Types:* Support for 64-bit integers, floating-point values, strings etc.

References and Links

Harish Doraiswamy and Jayant R. Haritsa. "Raster Is Faster: Rethinking Ray Tracing in Database Indexing". In: *Conference on Innovative Data Systems Research (CIDR)*. 2026. URL: https://vldb.org/cidrdb/papers/2026/p18-doraiswamy.pdf.

Yangming Lv et al. "RTScan: Efficient Scan with RT Cores". In: *Proceedings of the VLDB Endowment (PVLDB)* (2024). URL: https://dl.acm.org/doi/10.14778/3648160.3648183.

Akash Maji. *CompactScan Source Code*. GitHub repository. 2026. URL: https://github.com/akashmaji946/raster-scan/tree/mid-term.

Microsoft Research. *RasterScan: GPU Rasterization-Based Indexing*. https://github.com/Microsoft/raster-scan. Source code repository. 2025.