Spatial-Blocked Bloom Filters: Cache-Efficient Spatial Membership for Raster and Voxel Maps

Moritz Laass
Technical University of Munich
School of Engineering and Design
Munich, Bavaria, Germany
moritz.laass@tum.de

Paul Walther
Technical University of Munich
School of Engineering and Design
Munich, Bavaria, Germany
paul.walther@tum.de

Martin Werner
Technical University of Munich
School of Engineering and Design
Munich, Bavaria, Germany
martin.werner@tum.de

Abstract

Efficient membership queries for sparse spatial data, 2D raster cells and 3D voxels, are central to geospatial analytics, remote sensing, real-time rendering, and collision avoidance for Unmanned Aerial Vehicles (UAVs) that encode their environment as voxel occupancy maps. Traditional Bloom filters minimize collisions by destroying data locality via hashing. However, spatial data is highly correlated: neighboring cells are often queried together, and real maps are connected. We propose the Spatial-Blocked Bloom Filter (SBBF), which uses Space-Filling Curves (SFCs) to map multi-dimensional coordinates onto a blocked Bloom filter structure. Bit-splitting derives the block index from low-order SFC bits and the intra-block mask from high-order bits, so SBBF gains cache locality for sequential spatial traversals and benefits from hardware prefetching during raster scans and volume rendering. Our evaluation shows approximately 5–7× fewer instructions and 4–5× lower false positive rates than cache-optimized baselines, with sequential neighborhood queries seeing the largest cache-locality benefit.

CCS Concepts

• **Information systems** → **Data structures**; **Spatial-temporal systems**; • **Computing methodologies** → *Spatial and physical reasoning*.

Keywords

Bloom filter, space-filling curves, raster data, voxel data, voxel occupancy maps, point clouds, cache optimization, spatial indexing

ACM Reference Format:
Moritz Laass, Paul Walther, and Martin Werner. 2026. Spatial-Blocked Bloom Filters: Cache-Efficient Spatial Membership for Raster and Voxel Maps. In *The 34th ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL '26), November 03–06, 2026, Riverside, CA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3841645.3844201

1 Introduction

Spatial data structures underpin applications from geospatial analytics and remote sensing to computer graphics and robotics. In 2D, raster representations encode satellite imagery, land cover classifications, and digital elevation models. In 3D, voxel-based representations have seen a resurgence driven by neural radiance fields (NeRFs), sparse voxel octrees (SVOs), and large-scale point cloud processing. In robotics, a UAV that encodes its 3D environment as a voxel occupancy map needs an onboard structure with a fixed memory budget: dense grids do not fit at fine resolution, while a Bloom-filter encoding does. Across all of these domains the fundamental operation is the membership query: *Does a spatial element exist at a coordinate* $\mathbf{p} \in \mathbb{R}^n$*?*

Bloom filters have been successfully applied to compress raster and voxel datasets, including global-scale spatial data [10, 11] to answer such queries. Their advantage lies in a $O(1)$ query time and a compact memory usage, but they suffer from poor cache performance due to random access patterns. Previously proposed Blocked Bloom filters (BBFs) [8] improve cache performance by confining bits to a single cache line.

In this paper, we argue that preserving spatial locality by aligning the memory layout of the filter with the spatial structure of the data using Space-Filling Curves (SFCs) in the Spatial-Blocked Bloom Filter (SBBF) further improves cache performance. Thereby, SFCs linearize multi-dimensional data while preserving proximity [7], and work identically for 2D and 3D coordinates. We use Morton (Z-order) and Hilbert curves as SFCs and show that this yields faster indexing (bit-interleaving instead of hashing), lower false positive rates, and faster neighborhood queries. Therefore, we also contribute fast 2D/3D Morton and Hilbert encoders[1]. Thus, we make the following contributions:

- SBBF, the first Bloom filter variant that replaces hash-based block indexing with space-filling curves, preserving spatial locality for cache-efficient sequential traversals.
- We analyze SFC+seed combinations as hash function replacements, with block uniformity comparable to common hash functions like XXH3 while preserving spatial locality.
- We show lower FPR and less instructions than cache-optimized baselines, on real and synthetic data.

2 Related Work

Blocked Bloom filters (BBFs) address cache inefficiency of standard Bloom filters, which require up to k random memory accesses per query. BBFs [8] store all k bits of an element into a single block with the size of a cache line, selected by a hash function, thereby reducing cache misses from k to 1 with a small FPR penalty. Precomputed bit patterns further reduce per-query work and enable SIMD implementations. Lang et al. [3] further optimized this with register-blocked and cache-sectorized variants that test all k bits with a single comparison. Both works treat keys as uniformly

[1] Source code available at https://github.com/mlaass/sbbf

SIGSPATIAL '26, Riverside, CA, USA

ACM ISBN 979-8-4007-2950-8/2026/11
https://doi.org/10.1145/3841645.3844201

random: they optimize *intra-query* cache behavior but ignore *inter-query* locality, so spatially adjacent queries still touch unrelated blocks.

Space-filling curves (SFCs) such as the Morton curve [7] and the Hilbert curve [2] map multi-dimensional coordinates to one dimension while preserving proximity, and are widely used for spatial indexing and cache-friendly array layouts [9]. For range filtering, succinct structures such as SuRF [12] and Rosetta [6] support prefix queries but pay tree-traversal latency per lookup. Roaring bitmaps [5] provide exact membership over SFC-encoded keys, but their memory footprint grows with data entropy, whereas a Bloom filter guarantees a fixed budget. To our knowledge, SBBF is the first filter to derive both the block index and the intra-block bit pattern from a single SFC evaluation, replacing hash-based block selection with a locality-preserving mapping.

3 Methodology

We introduce the Spatial-Blocked Bloom Filter (SBBF), shown in Figure 1. A spatial coordinate $\mathbf{p}$ (generally in $\mathbb{R}^n$, but practically either (x, y) for 2D raster data or (x, y, z) for 3D voxels) is first mapped to a 1D SFC value s. The low-order bits of s determine the block index, ensuring spatially adjacent cells map to adjacent memory blocks. The high-order bits serve as a "regional signature" to derive the intra-block bit mask.

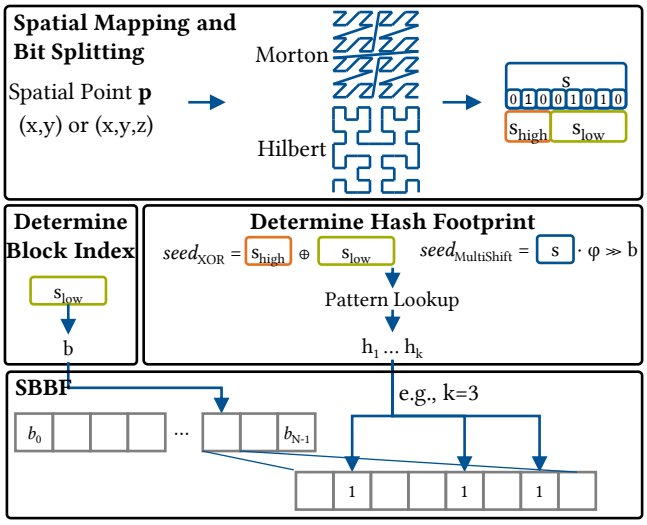

Figure 1: SBBF Architecture. Low SFC bits (s_{low}) select block b_i (enabling sequential access). The seed for intra-block hashing combines high and low bits (e.g., $s_{\text{high}} \oplus s_{\text{low}}$ or multiply-shift) to derive k bit positions via linear combinations as in double-hashing or with a pattern lookup.

3.1 Spatial Mapping and Bit-Splitting

The core of SBBF is the mapping function that converts a spatial coordinate $\mathbf{p}$ into a block index and a bit mask seed. Given an SFC value $s = \text{SFC}(\mathbf{p})$, we partition s into two components:

$$s = s_{\text{high}} \cdot 2^p + s_{\text{low}} \tag{1}$$

where $p = \log_2 N$ and N is the number of blocks, and each block contains B bits. The total filter size is $m = N \cdot B$ bits.

3.2 Block Index

To determine the block index b for a location $\mathbf{p}$, we use:

$$\text{b} = s_{\text{low}} = \text{SFC}(\mathbf{p}) \bmod 2^p = \text{SFC}(\mathbf{p}) \bmod N \tag{2}$$

Why low bits for block index? This serves two purposes:

(1) *Sequential traversal efficiency.* The low-order bits of an SFC change with every unit step in the coordinate space. Spatial neighbors map to blocks $b_i, b_{i+1}, \ldots$, enabling the CPU to leverage L1/L2 prefetching during neighborhood queries and raster/volume traversals; in our measurements this yields 40–160× lower L1-dcache miss rates than hash-based filters.

(2) *Uniform block occupancy.* Consider the alternative: using high bits for block selection. Since high bits identify coarse spatial regions, clustered data would concentrate in a few blocks, creating "hotspots" with high occupancy variance. By Jensen's inequality, this variance increases aggregate FPR. Using low bits instead distributes spatially coherent data across blocks in a round-robin fashion, ensuring near-uniform occupancy even for highly clustered inputs.

For arbitrary spatial neighbors, the SFC's locality property ensures they map to nearby (though not necessarily adjacent) blocks. Hilbert curves provide tighter locality than Morton, but both achieve substantially better cache reuse for neighborhood queries than random hashing.

3.3 Intra-Block Hashing

Once block b is selected, we must determine which k bits to set within that block. Standard Bloom filters would invoke k independent hash functions, but this is computationally expensive. Instead we use a seed-based hash replacement, where the seed is obtained either through an XOR operation or a multiply-shift:

$$seed_{\text{XOR}}(\mathbf{p}) = s_{\text{high}} \oplus s_{\text{low}} \tag{3}$$

$$seed_{\text{MultiShift}}(\mathbf{p}) = s \cdot \varphi_{64} \gg r \tag{4}$$

Why combine high and low bits? Using only the high bits (s_{high}) as the seed causes false positives to appear as topologically coherent partial "copies" of the original data, since coarse spatial regions share identical seeds. We evaluate two combination strategies: XOR ($s_H \oplus s_L$) and multiply-shift ($s \cdot \varphi_{64} \gg r$), where φ_{64} is the 64-bit Fibonacci hash constant (0x9E3779B97F4A7C15) and the shift by r selects the high-order bits. XOR provides simpler implementation and lower FPR on uniformly distributed data, while multiply-shift's stronger avalanche properties produce more spatially incoherent false positives.

Pattern Lookup. Following Putze et al. [8], we precompute a table of Ω random k-bit patterns (typically $\Omega = 2^{16}$), indexed by seed mod Ω, yielding the intra-block bit mask in a single memory access while enabling SIMD implementations. This trades 4 KB of table space for a reduced instruction count (Table 1).

Collision behavior. Unlike standard Bloom filters, where collisions are uniformly distributed, SBBF collisions exhibit spatial structure. Two points $\mathbf{p}$ and $\mathbf{q}$ collide (map to identical block and mask) when sharing the same low bits (same block) and when their

XOR seeds $s_H \oplus s_L$ are identical. Since the seed combines both coarse (high bits) and fine (low bits) spatial information, collisions occur between points that share structural similarity at multiple scales.

4 Experiments

In the following, we compare SBBF's indexing components against standard hash functions, evaluate scalability, neighborhood query performance, false positive rates, and end-to-end operation cost.

4.1 Experimental Setup

All benchmarks were conducted on a Linux workstation with a 12th Gen Intel Core i7-12700KF (12 cores, 20 threads, 3.6 GHz base, 5.0 GHz turbo) and 64 GB DDR4 RAM. Code was compiled with GCC 13.3 using `-Ofast -march=native`. We use the nanobench library [1] with Linux `perf_event` for cycle-accurate measurements. While newer CPUs (Intel 13th–14th gen, AMD Zen 4) feature larger L2/L3 caches and improved branch predictors, the fundamental latency hierarchy remains: L1 access (~4 cycles) versus LLC (~40 cycles) versus DRAM (~200 cycles). SBBF's advantage stems from converting random LLC/DRAM accesses into sequential L1 hits via SFC-ordered traversal, a benefit that persists across architectures.

Datasets. We evaluate on two dataset categories:

Real-world geographic data. The GDELT Global Event Database [4] provides 1.93M georeferenced news events. For 2D experiments, coordinates are quantized to 16-bit precision. For 3D experiments, we include an event category dimension (QuadClass: verbal/material cooperation/conflict).

Synthetic data. For controlled experiments we generate: (1) uniform random distributions with 100K elements across the 16-bit coordinate space, and (2) clustered distributions with 50 Gaussian clusters ($\sigma = 500$).

Baselines. We compare SBBF against two cache-optimized Bloom filter variants:

- **BlockedBF** [8]: 512-bit cache-line-aligned blocks
- **RegisterBF** [3]: 64-bit register-blocked design

All filters use similar memory budgets (1.05 MB) for fair comparison.

4.2 Indexing Component Performance

Table 1 compares the building blocks of spatial Bloom filter indexing using system-independent metrics: retired instructions, CPU cycles, and Instructions Per Cycle (IPC).

Morton encoding uses far fewer instructions than the hash-function baselines (Table 1). The Hilbert encoding uses more instructions than Morton, trading computation for improved spatial locality. For intra-block bit selection, `pattern_lookup` derives the full k-bit mask in a single table access, requiring only 11 instructions per operation. This is consistent to measured insert and query latencies in Figure 2.

4.3 Neighborhood Query Performance

Table 2 compares neighborhood query performance for 3D data (6 neighbors per point). SBBF exploits SFC locality: spatially adjacent cells map to nearby memory addresses, enabling cache-line reuse across neighbor queries.

Table 1: Indexing Component Performance

Method	Instr.	Cycles	IPC	M ops/s
Hash Functions (16 bytes)				
MurmurHash3	129	22	5.9	168
XXH3	93	14	6.6	255
SFC Encoding				
Morton 2D (PDEP)	15	8	1.9	443
Morton 3D (PDEP)	20	8	2.5	450
Hilbert 2D (LUT)	72	15	4.8	238
Hilbert 3D (LUT)	173	51	3.4	70
Intra-block Hashing (k=4)				
pattern_lookup	11	8	1.4	468

Table 2: 3D Neighborhood Query Performance (6 neighbors)

Filter	Instr.	Cyc.	IPC	×BL
SBBF-Morton	**56**	**9**	6.2	0.21×
SBBF-Hilbert	104	23	4.5	0.53×
BlockedBF	258	43	6.0	1.00×
RegisterBF	254	39	**6.5**	0.91×

SBBF-Morton achieves substantially fewer cycles than the baselines for neighbor queries (Table 2). The high IPC indicates excellent instruction-level parallelism from cache hits. Morton outperforms Hilbert here because its simpler encoding allows for faster computation of neighbor coordinates. The performance is also shown in latency measurements (Figure 2).

4.4 False Positive Rate Analysis

Table 3 compares observed FPR across synthetic and real distributions. All filters use $k = 8$ hash functions, 64-bit blocks, and identical memory budgets (1.05 MB, 100K elements) for fair comparison.

SBBF achieves 4–5× lower FPR than BlockedBF, with the largest gap on 3D clustered data. RegisterBF achieves FPR comparable to SBBF but at a higher instruction count (Table 4).

4.5 SBBF End-to-End Performance

Table 4 compares end-to-end operation performance using the same system-independent metrics as Table 1. SBBF-Morton uses far fewer instructions than BlockedBF in both 2D and 3D (Table 4). SBBF-Hilbert also uses fewer instructions than BlockedBF in 2D while providing better spatial locality. The lower IPC for SBBF reflects reduced instruction-level parallelism but is offset by the lower instruction count. All SBBF results use 64-bit blocks.

5 Conclusion

We presented SBBF, a spatial-blocked Bloom filter that aligns memory layout with spatial structure by deriving the block index and the intra-block bit pattern from a single space-filling curve evaluation. SBBF needs 5–7× fewer instructions and yields 4–5× lower false positive rates than cache-optimized baselines, with the largest gains on sequential neighborhood queries. Our 2D/3D Morton encoders

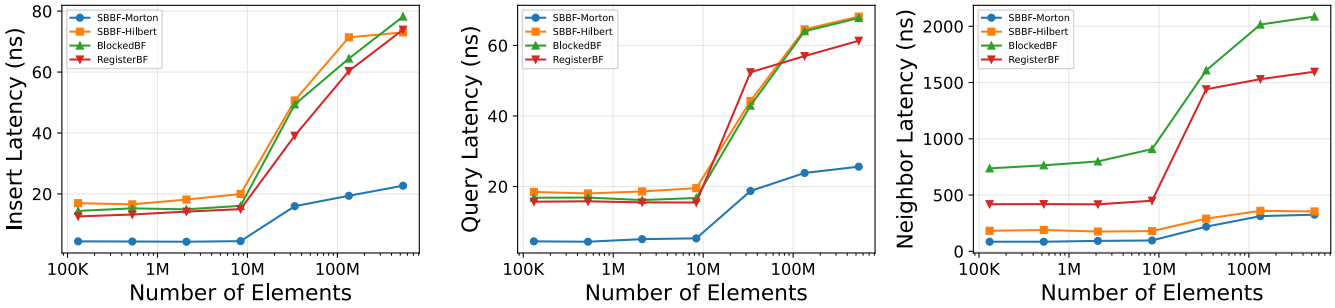

Figure 2: 3D scalability benchmarks (128K–2B elements): insert, query, and neighbor latency with the pattern-lookup strategy.

Table 3: False Positive Rate Comparison (pattern-lookup intra-block strategy). ×BL is the FPR relative to the BlockedBF baseline (lower is better).

		2D		3D	
Dist.	Filter	FPR	×BL	FPR	×BL
Synthetic Uniform (100K elements), FPR in 10^{-4}					
	SBBF-Morton	**7.6**	0.24×	7.2	0.25×
	SBBF-Hilbert	8.8	0.28×	7.5	0.26×
	BlockedBF	31	1.00×	29	1.00×
	RegisterBF	8.4	0.27×	**6.8**	0.24×
Synthetic Clustered (100K elements, 100 clusters), FPR in 10^{-4}					
	SBBF-Morton	**7.6**	0.26×	**5.3**	0.18×
	SBBF-Hilbert	8.3	0.29×	6.7	0.23×
	BlockedBF	29	1.00×	29	1.00×
	RegisterBF	8.5	0.30×	6.3	0.22×
GDELT Global Events [4] (1.9M events), FPR in 10^{-5}					
	SBBF-Morton	**2.6**	0.27×	**7.5**	0.23×
	SBBF-Hilbert	2.9	0.30×	**7.5**	0.23×
	BlockedBF	9.6	1.00×	33	1.00×
	RegisterBF	2.8	0.30×	8.8	0.27×

Table 4: End-to-End Operation Performance (pattern-lookup intra-block strategy)

		Insert		Query	
Dim	Filter	Instr.	Cyc.	Instr.	Cyc.
2D					
	SBBF-Morton	**51**	**16**	**54**	**16**
	SBBF-Hilbert	106	30	109	29
	BlockedBF	241	48	255	50
	RegisterBF	236	45	239	45
3D					
	SBBF-Morton	**40**	**16**	**54**	**17**
	SBBF-Hilbert	195	64	209	71
	BlockedBF	261	56	273	54
	RegisterBF	250	51	253	49

are faster than state-of-the-art hash functions. In future work we will study multi-threaded traversals and topological post-processing of query results on connected maps. Our implementation is publicly available at https://github.com/mlaass/sbbf.

Acknowledgments

Use of AI Tools. In accordance with ACM policy on authorship, generative AI tools (Claude, Gemini) were utilized to assist with code development, debugging, and manuscript editing. All AI-generated content was reviewed, verified, and revised by the authors, who take full responsibility for the final work.
Funding. This work is partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 507196470.

References

[1] Martin Ankerl. 2019. nanobench: Simple, Fast, Accurate Single-Header Microbenchmarking Functionality for C++11/14/17/20. https://github.com/martinus/nanobench
[2] Wanli Chen, Xinge Zhu, Guojin Chen, and Bei Yu. 2022. Efficient Point Cloud Analysis Using Hilbert Curve. In *European Conference on Computer Vision (ECCV)*.
[3] Harald Lang, Thomas Neumann, Alfons Kemper, and Peter Boncz. 2019. Performance-Optimal Filtering: Bloom Overtakes Cuckoo at High Throughput. *Proceedings of the VLDB Endowment* 12, 5 (2019), 502–515. doi:10.14778/3303753.3303757
[4] Kalev Leetaru and Philip A. Schrodt. 2013. GDELT: Global Data on Events, Location, and Tone, 1979–2012. In *International Studies Association Annual Convention*. San Francisco, CA. Available at https://www.gdeltproject.org/.
[5] Daniel Lemire, Gregory Ssi-Yan-Kai, and Owen Kaser. 2016. Consistently faster and smaller compressed bitmaps with Roaring. *Software: Practice and Experience* 46, 11 (2016), 1547–1569. doi:10.1002/spe.2402
[6] Siqiang Lu, Conglong Jin, Lee Breslau, Vladislav Shkapenyuk, and Torsten Suel. 2020. Rosetta: A Robust Space-Time Optimized Range Filter for Key-Value Stores. In *Proceedings of the 2020 International Conference on Management of Data (SIGMOD)*. ACM, 2071–2086.
[7] Guy M. Morton. 1966. *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing*. Technical Report. IBM Ltd., Ottawa, Canada.
[8] Felix Putze, Peter Sanders, and Johannes Singler. 2009. Cache-, Hash-, and Space-Efficient Bloom Filters. *ACM Journal of Experimental Algorithmics* 14 (2009), 4.4:4.1–4.4:4.18. doi:10.1145/1498698.1594230
[9] Uznir Ujang, Francois Anton, Suhaibah Azri, Alias Abdul Rahman, and Darka Mioc. 2014. 3D Hilbert Space Filling Curves in 3D City Modeling for Faster Spatial Queries. *International Journal of 3-D Information Modeling* 3, 2 (2014).
[10] Martin Werner. 2019. GloBiMaps – A Probabilistic Data Structure for In-Memory Processing of Global Raster Datasets. In *27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '19)*.
[11] Martin Werner. 2021. GloBiMapsAI – An AI-Enhanced Probabilistic Data Structure for Global Raster Datasets. *ACM Transactions on Spatial Algorithms and Systems* (2021). doi:10.1145/3453184
[12] Huanchen Zhang, Hyeontaek Lim, Viktor Leis, David G. Andersen, Michael Kaminsky, Kimberly Keeton, and Andrew Pavlo. 2018. SuRF: Practical Range Query Filtering with Fast Succinct Tries. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD)*. ACM, 323–336.