Method for Memory-Bounded Construction of a Globally Complete, High-Zoom-Level Cloud-Optimized GeoTIFF from Tile-Access Logs

Sascha Brawer · sascha@brawer.ch · Bern, Switzerland

Published on Technical Disclosure Commons (https://www.tdcommons.org/) · September 2, 2026

Abstract. A memory-bounded method is disclosed for constructing a globally complete, high-zoom-level Cloud-Optimized GeoTIFF — a planet-wide raster at web-map zoom level 18, on the order of 10^{11} cells — from a rolling window of tile-access logs, using working memory that is essentially constant in the number of cells. The output is a standards-compliant COG with a full reduced-resolution overview pyramid, derived by robustly aggregating the per-tile time series; it is several hundred times smaller than a naïve dense encoding. The method combines a level-embedding space-filling tile key, a two-stage disk-backed ordering (per-period external sort followed by a cross-period streaming merge), a bounded root-to-leaf raster working set that follows from consuming the ordered stream in tree pre-order, single-pass inline overview generation, per-cell robust aggregation computed without materializing the time series, and three complementary output-size reductions (magnitude quantization, tile deduplication by shared file offsets, and a monotone value transform). A worked configuration builds a global zoom-18 raster in under one gigabyte of resident memory and emits a roughly 600 MB file where a dense encoding would require approximately 275 GB.

Keywords. Defensive publication, prior art, Cloud-Optimized GeoTIFF, space-filling curve, Morton / Z-order code, external sort, streaming k-way merge, windowed median, single-pass overview pyramid, tile deduplication by shared file offsets, tile-access logs, log analysis, web map tiles, constant-memory raster construction, OpenStreetMap.

Purpose. This document is published to establish prior art. The techniques described here, and obvious variations of them, are disclosed for free public use; this publication is intended to prevent anyone from obtaining patent rights that would restrict that use.

Reference implementation. https://github.com/brawer/osmviews (`cmd/osmviews-builder`), released under the MIT license; see §6.

1 Problem

Consider a data set of *tile-access counts*: for a web map tiled in the usual `zoom/x/y` scheme, a stream of records stating how many times each tile was requested in a given period (e.g. one day). Such logs are published, for example, by the OpenStreetMap Foundation.

The goal is to convert a rolling window of such logs (e.g. the most recent 52 weekly periods) into a geospatial raster in which each cell holds an aggregated, area-normalized "view density" for the corresponding location, at a spatial resolution equivalent to zoom level 18 — roughly 150 m per cell at the equator — with coverage of the entire globe, and delivered as a Cloud-Optimized GeoTIFF (COG) so that clients can read arbitrary sub-regions and zoom levels over HTTP range requests.

The naïve approach — allocate the full dense zoom-18 array, accumulate into it, then write it — is infeasible on commodity hardware. A global zoom-18 raster has $2^{18} \times 2^{18} \approx 6.9 \times 10^{10}$ cells; at four bytes per cell that is approximately 275 GB, before overviews. The aggregation also requires holding, per cell, enough of the time series to compute a robust statistic such as a median. Neither fits in memory, and materializing either on disk in dense form is wasteful because the overwhelming majority of the planet (oceans, deserts, ice) has little or no data.

The techniques below build such a raster in a single streaming pass, with peak memory that scales with the *depth* of the tile tree (a small constant, e.g. 11 or 19) rather than its *breadth* (the cell count), and with an output file whose size tracks the information content of the data rather than the dimensions of the grid.

2 Overview of the approach

1. Encode each `zoom/x/y` tile as an integer *tile key* whose numeric order is a depth-first pre-order traversal of the tile quad-tree (§3.1).
2. For each time period, sort that period's records by tile key using an external (disk-backed) sort, and coalesce equal keys; store the sorted, coalesced period as a compressed file (§3.2).
3. Merge the sorted period files with a streaming k-way merge, so that a single globally key-ordered record stream is produced, in which every period's records for a given tile arrive consecutively (§3.2).

4. Consume that stream. For each tile, collect its per-period values and reduce them to one aggregated value with a robust statistic that tolerates missing periods, computed directly from the ordered run (§3.3).
5. Paint aggregated values into 256×256 raster tiles held in a tree. Because the stream is in pre-order, only the chain of raster tiles from the root to the tile currently being filled is ever live; completed raster tiles are evicted immediately (§3.4).
6. On eviction, subsample the completed raster tile into its parent, building the overview pyramid inline during the same pass (§3.5).
7. When writing each raster tile, apply three complementary size reductions: quantize low-magnitude tiles to a common value; store each distinct tile body once and make other tiles reference the same file bytes; and store a monotone transform of the value (§3.6).
8. Assemble the COG: image file directories first, tile bodies staged in a temporary file, offsets back-patched (§3.7).

Taken individually, several ingredients are well known — Morton / Z-order curves, external sorting, k-way merges, mipmap generation. The disclosure is the specific integration that yields constant-memory, single-pass construction of a *globally complete, high-zoom, overview-complete COG from a time series*, together with the size-reduction techniques of §3.6, in particular tile deduplication by shared file offsets (§3.6.2).

3 Detailed description

3.1 Level-embedding space-filling tile key

Each tile (z, x, y) is mapped to a fixed-width unsigned integer key as follows. Reserve the least-significant b bits for the zoom level z ($b = 5$ holds any $z \leq 31$). Above those, interleave the coordinate bits two per level, most-significant first: bits $z-1$ of x and y (the root-quadrant choice) fill the top key-bit pair, bits $z-2$ the next, and so on down to bits 0 in the lowest pair. Unused low bits are zero. The coordinate field is then $2z$ bits wide, so a 64-bit key stays collision-free for $z \leq 29$; the reference implementation accepts $z \leq 24$.

This is an interleaving (Z-order / Morton) code with two properties that the method depends on:

- **Numeric order equals pre-order traversal.** For two keys, the comparison proceeds level by level from the root. A tile's key is numerically smaller than the keys of all tiles strictly inside it, and an entire sub-tree occupies one contiguous key interval. Sorting keys ascending therefore yields a depth-first, pre-order enumeration of the quad-tree.
- **Cheap tree operations.** "Is tile A an ancestor of tile B", "truncate B to level k", and "successor of A in pre-order at maximum level m" are all constant-time bit manipulations of the key.

Logs contain tiles at many zoom levels, so the record stream is *mixed-level*; embedding the level in the key — rather than interleaving x and y alone at a fixed level — is what makes such a stream sortable into a single valid traversal.

Variations. Any bijective encoding with the pre-order property works: the level bits may occupy the high end instead of the low end; a Hilbert-curve ordering may be substituted where its locality is preferred and the ancestor test is adjusted; trees of branching factor other than four (e.g. binary "tile pyramids", octrees for volumetric data) generalize directly by changing the bits-per-level.

3.2 Two-stage ordering: per-period external sort, cross-period streaming merge

Per period. The records of one period are parsed to (tile key, count) pairs and fed to an external (disk-backed) sort on that key. The sort spills runs to temporary files and merges them, so peak memory is bounded by its configured buffer regardless of the number of distinct tiles. The sorted output is streamed once more to coalesce equal keys (a tile requested on several days of the period becomes one record with the summed count) and written to a compressed file. These per-period files are cached (locally and, optionally, in object storage) and reused on subsequent runs, so that a periodic rebuild only sorts the newest period(s).

Across periods. To aggregate the window, the N sorted period files are combined by a streaming k-way merge: maintain a min-heap of N cursors ordered by tile key, then repeatedly emit the smallest and advance its cursor. The result is one record stream, globally ordered by tile key, in which **all N periods' records for a given tile arrive consecutively**, followed by all records for the next tile, in quad-tree pre-order. Working memory for the merge is the heap plus one buffered record per period — constant in the cell count and linear only in the (small, fixed) window size N.

An optional per-period scalar weight may be applied to counts as they are read from each cursor (for example, to extrapolate a period with missing days to a full period by scaling by the ratio of expected to observed days); this does not affect ordering.

In an implementation, the merge and the paint stage run as a two-stage pipeline: a single **producer** runs the merge, decompressing the period files as it reads them, while a single **consumer** (§3.3–§3.7) paints the raster. The two are coupled by a bounded buffer, so decompression and painting overlap and the buffer adds only a constant to working memory; each stage is single-threaded.

3.3 Per-cell aggregation from the ordered stream

The consumer reads the merged stream in order and accumulates the run of records that share a tile key into a small buffer (at most N values). When the key changes, it reduces the buffer to one value.

The **windowed median** is a robust reduction that tolerates missing periods and is computable directly from the ordered run: treat periods with no record for this tile as zeros, conceptually prepended to the sorted list of present values. Let c be the number of present values (periods that have a record for this tile). The median is then the element at index $\lfloor N/2 \rfloor$ of the

length-N conceptual list, i.e. the present value at index $\lfloor N/2 \rfloor - (N - c)$, or zero if that index is negative (the tile appears in fewer than half the periods). Because the k-way merge breaks tile-key ties by ascending count, the buffered present values already arrive sorted, so no per-cell sort is needed. The median suppresses transient spikes, seasonality, and the occasional noisy weighted period.

The reduced value is then divided by the tile's true surface area (which, in Web Mercator, varies with latitude) to obtain an area-normalized density.

Variations. Any order statistic (a different quantile, a trimmed mean), or a decay-weighted mean, may be substituted; the point is that the statistic is obtained from the length-c ascending run plus the count of absent periods ($N - c$), without ever holding all N periods for all cells.

3.4 Bounded raster working set via pre-order consumption

The output raster is produced in fixed-size **raster tiles** of $T \times T$ cells (e.g. $T = 256$). The full-resolution level is a grid of $2^L \times 2^L$ raster tiles, where $L = \text{zoom} - \log_2(T)$ (e.g. $L = 10$ for zoom 18, $T = 256$). A raster tile is a $T \times T$ array of the cell type plus a pointer to its parent raster tile.

The consumer maintains only a **path of raster tiles from the root to the one currently being filled** — at most $L + 1$ of them. When a record arrives whose target raster tile is not contained in the current one, the pre-order property guarantees that no later record can fall inside the current raster tile or inside any ancestor that does not contain the new target. Therefore those raster tiles are *final*. The consumer:

1. evicts each finalized raster tile (compress its body, append it to the output staging file, record its offset and length; see §3.5 and §3.6);
2. walks down toward the new target, allocating the chain of intermediate raster tiles, and for every sibling sub-tree that is skipped entirely, emits a single uniform raster tile (§3.6.1) carrying a fill value (zero, or a nearest-neighbor carry-forward);
3. allocates the new leaf raster tile.

Consequently, the peak live raster memory is $(L + 1) \cdot T^2 \cdot \text{sizeof(cell)}$ — for $L = 10$, $T = 256$, four-byte cells, about 2.8 MiB — **independent of how many cells the raster has**. Peak process memory in a worked implementation is about 800 MiB, dominated by language-runtime overhead, compression buffers, and the N open period readers (a decompressor and one buffered record each), not by the raster.

Regions never mentioned by any record are emitted as uniform raster tiles during a final sweep, so the raster has no holes: it covers the whole grid.

3.5 Single-pass inline overview generation

When a leaf or interior raster tile is finalized and about to be evicted, it is first subsampled $2 \times 2 \to 1$ into the appropriate quadrant of its parent raster tile (e.g. by max-pooling, which preserves visible hotspots; average- or min-pooling are alternatives). Because eviction happens in pre-order and a parent is only finalized after all four of its children, the entire overview pyramid (levels $L - 1$ down to 0) is produced **within the same single pass** over the input, with no separate downsampling stage and no re-reading of the full-resolution data.

3.6 Output-size reduction

Three techniques act together so that the file size tracks the data's information content rather than the grid dimensions.

3.6.1 Magnitude quantization

Before compressing a raster tile, test whether every cell, *rounded to the storage precision* (e.g. to the nearest integer density), equals the same value. Because the great majority of the planet has a density far below one unit, huge contiguous areas round to a single value (typically zero). A raster tile that is uniform after rounding is handled as a uniform tile keyed by that rounded value. This both removes the need to compress most tiles and feeds the deduplication of §3.6.2.

3.6.2 Tile deduplication by shared file offsets

Maintain, per pyramid level, a map from *tile body content* to the file offset and length at which a tile with that content was first written. In the reference implementation the key is the shared rounded value of a uniform tile (§3.6.1); hashing the compressed body would extend the same mechanism to non-uniform tiles. When a later tile has content already in the map, **do not write its body again**: instead set its entry in the format's tile-offset and tile-bytecount arrays to point at the *same* byte range as the earlier tile.

The result is that each pyramid level of the file stores exactly one compressed body for "all-zero", one for "all-one", and so on, and the hundreds of thousands of ocean tiles at that level all reference it. It is unusual for multiple image tiles to resolve to one shared byte range — most encoders never do it — but the TIFF 6.0 specification permits it (the `TileOffsets` and `TileByteCounts` arrays are just arrays of pointers and lengths; nothing requires them to be distinct or monotone), and mainstream readers (e.g. GDAL, `tifffile`, browser GeoTIFF libraries) decode it correctly.

A corollary: the file's structural metadata must **not** advertise a strict tile ordering or contiguous tile layout (for COGs, the `BLOCK_ORDER` ghost option is deliberately omitted), because such a declaration would forbid tiles from sharing bytes.

Reader-side complement. A client that decodes tiles on demand should key its decoded-tile cache by each tile's byte offset in the container rather than by grid position. Because the technique above makes many grid positions resolve to the same byte range, an offset-keyed cache collapses every reference to a shared body — for example all of the ocean tiles at a given level — into a single entry, so scanning a large uniform region costs one cache slot instead of repeatedly evicting useful tiles. The client libraries of §6 do this.

3.6.3 Monotone value transform

Cell values are stored as a fixed monotone transform of the density, for example $\ln(1 + \text{density})$. This preserves the ordering of cells (so any threshold or ranking is unchanged) while compressing a very skewed dynamic range (roughly 0 to 2.3×10^5) into a near-linear small range (roughly 0 to 12), which improves both the compressibility of tile bodies and the out-of-the-box behavior of visualization tools. The transform's maximum over the raster is recorded in a standard tag so that readers can invert or re-normalize it.

3.7 Cloud-Optimized GeoTIFF assembly

A COG places all the image file directories (IFDs) and a small block of structural "ghost" metadata ahead of the tile data, so that a reader learns the file's structure from a short prefix and then fetches only the tiles it needs with HTTP range requests. But the tile offsets are not known until the tiles have been written. The method therefore:

1. streams compressed tile bodies to a temporary file during the pass (§3.4), recording each body's provisional offset within that file and its length;
2. writes the file header, the ghost metadata, and all IFDs (full-resolution level and every overview level), with placeholder tile-offset values;
3. for each pyramid level in turn, appends that level's tile-offset array followed by its tile bodies, drawn from the temporary file and now at their final absolute offsets; the per-level tile-bytecount arrays follow at the end of the file;
4. seeks back and patches every IFD's tile-offset and tile-bytecount pointer, and the inter-IFD "next IFD" pointers, to the final values.

Classic (32-bit) TIFF is sufficient: the size reductions of §3.6 keep the output below 4 GiB. Standard georeferencing tags place the grid in the target projection; provenance tags record the input date range, the producing software version, and the date of the most recent input period.

4 Resulting properties

- **Peak memory constant in cell count.** Determined by tree depth L, window size N (one decompressing reader per period), and compression buffers. A worked configuration builds a global zoom-18 raster with pyramid in about 800 MiB resident, well under 1 GiB.
- **Single pass.** The input is read once; overviews are a by-product; no stage re-reads the full-resolution raster.
- **Output size tracks information, not dimensions.** A worked configuration emits ≈ 600 MB — the exact figure drifts from week to week with the input — where a dense zoom-18 encoding would be ≈ 275 GB.
- **Incremental.** Re-running for a shifted window resorts only the new period(s); everything else is reused from cache.
- **Standards-compliant output.** A valid Cloud-Optimized GeoTIFF readable by unmodified mainstream tools, including the shared-offset tiles of §3.6.2.
- **Commodity hardware.** No cluster, no large-memory machine, no GPU.

5 Variations and generalizations

The disclosure covers, without limitation, the following variations:

- **Any raster produced by aggregating a key-ordered time series into a tiled pyramid**, not only view-density maps but also population, emissions, elevation change, sensor coverage, edit activity, and the like.
- **Any spatial tree.** Quad-trees (this document), binary tile pyramids, octrees / 3D tiles for volumetric or point data, or non-square tilings, by adjusting the bits-per-level of the key of §3.1 and the arity of the eviction/subsample step.
- **Any projection or CRS**, geographic or projected; the area-normalization of §3.3 is projection-specific but optional.
- **Any robust per-cell reduction** computed from an ascending run plus an absent-period count: other quantiles, trimmed or winsorized means, decay- or recency-weighted means, min, max.
- **Any pooling rule** for overview generation: max, min, mean, median, mode, or a learned kernel.
- **Any container format** whose tile index is an array of (offset, length) pairs and does not mandate distinctness — TIFF, BigTIFF, GeoTIFF, or a bespoke format — for the deduplication of §3.6.2; use BigTIFF in place of classic TIFF when §3.6 does not keep the file under 4 GiB.
- **Any lossless codec** for tile bodies; **any monotone transform** (logarithm, square root, μ-law, a fitted CDF) for §3.6.3.
- **Any spill-to-disk external sort and any k-way merge** for §3.2, including merging directly from remote object storage.
- **Gap fill** by zero, by the nearest painted ancestor, by interpolation, or by leaving an explicit nodata value.
- **Parallelization.** The per-period sorts of §3.2 are trivially parallelizable. The merge-and-paint pass of §3.3–§3.7 can be sharded by sub-tree: each shard covers a disjoint key range and quad-tree sub-tree, painting it and its overviews down to the shard root; a short serial tail then subsamples the shard roots up the remaining pyramid levels, concatenates the tile bodies (whose order is immaterial), and fixes up the IFD offsets. Because sub-trees over ocean carry almost no data while urban ones carry most of it, shard far more finely than the core count and balance the shards dynamically. Within a shard, only the pre-order walk and its eviction/subsample step are inherently sequential; input decompression (one worker per period file)

and tile-body compression (one per evicted tile) parallelize freely.

- **Deferred / range-served overviews.** Emitting fewer overview levels, or none, and relying on the reader.
- **Reader-side caching by shared offset.** A client that reads a container with deduplicated tiles (§3.6.2) keys its decoded-tile cache on each tile's file offset rather than its grid coordinates, so every reference to one shared body occupies a single cache entry.
- **Bounded reads from fixed tile geometry.** Because every tile decodes to exactly $T \times T \times$ sizeof(cell) bytes, a reader can cap the decompressor's output at that size and reject, when opening the file, any tile whose recorded compressed length exceeds a small margin over the largest plausible compressed tile — bounding memory use on adversarial input with no separate size field.

6 Reference implementation

A complete, working implementation in the Go programming language is publicly available under the MIT license at `https://github.com/brawer/osmviews` (directory `cmd/osmviews-builder`; the tile key is in `tile.go`, the two-stage ordering in `tilelogs.go` and `merge.go`, the bounded raster path and inline overviews in `paint.go`, and the size reductions and COG assembly in `raster.go`). Reader-side reference implementations — client libraries that query the resulting file and implement the offset-keyed cache of §3.6.2 — are published under the MIT license at `https://github.com/brawer/osmviews-rs` (Rust) and `https://github.com/brawer/osmviews-py` (Python). The commit history of these repositories predates or accompanies this publication.

Appendix: enumerated disclosed techniques

For the avoidance of doubt, the following are disclosed for free public use, alone and in combination:

1. Encoding mixed-level map tiles as integers whose ascending numeric order is a pre-order traversal of the tiling tree, by reserving low (or high) bits for the level and interleaving coordinate bits per level.
2. Aggregating a windowed time series of per-tile counts by (a) an external, disk-backed sort of each period on that key, then (b) a streaming k-way merge across periods, yielding one key-ordered stream in which all periods' records for a tile are consecutive.
3. Applying a per-period scalar weight to counts during the merge to normalize periods with missing sub-intervals.
4. Computing a per-cell windowed order statistic (e.g. median with zero-fill for absent periods) directly from the ascending per-cell run and the count of absent periods, without materializing the window for all cells.
5. Painting the aggregate into a tree of fixed-size raster tiles while holding only the root-to-current-leaf path, justified by the pre-order arrival of records, with immediate eviction of finalized raster tiles.
6. Generating the entire reduced-resolution overview pyramid inline, by subsampling each raster tile into its parent at eviction time, in the same single pass over the input.
7. Emitting skipped sub-trees and never-touched regions as uniform raster tiles so the output has global coverage with no holes.
8. Quantizing raster tiles to a common value at storage precision so that large low-magnitude regions collapse to identical tiles.
9. Deduplicating identical raster tiles by setting multiple entries of the container's tile-offset / tile-length index to point at a single shared byte range, and correspondingly omitting any strict-tile-order declaration from the container's structural metadata.
10. Storing a monotone transform of the cell value to compress dynamic range while preserving order, with the transform's extent recorded in metadata.
11. Assembling a header-first (Cloud-Optimized) GeoTIFF by staging tile bodies in a temporary file and back-patching offsets after the single pass.
12. The combination of 1–11 to build a globally complete, high-zoom, overview-complete Cloud-Optimized GeoTIFF from tile-access logs using working memory that is constant in the number of cells.
13. Reading such a container by caching decoded tiles keyed on their shared byte offset, so that all references to a deduplicated tile body occupy one cache entry.
14. Bounding a reader's memory use by capping decompression output at the fixed decoded tile size and rejecting, at open time, any tile whose recorded compressed length exceeds the largest plausible compressed tile.
15. The combination of 1–14 as an end-to-end scheme for producing such a container and reading it.