Package: controlledburn (via r-universe)

October 24, 2026

Title Sparse Geometry Rasterization with Exact Coverage Fractions

Version 0.3.1

Description Rasterize polygon, line, and point geometry without materializing any pixel values. For polygons it computes exact coverage fractions for polygon-grid intersections, returning sparse tables: run-length encoded interior cells and individually weighted boundary cells. The scanline algorithm is O(perimeter) in both time and memory, using the scan line fill attributed to Wylie et al. (1967) <doi:10.1145/1465611.1465619>. Exact coverage fractions use the 'exactextract' algorithm by Daniel Baston, vendored from the 'exactextractr' package <doi:10.32614/CRAN.package.exactextractr>. Derived from the 'fasterize' package by Noam Ross.

License Apache License (== 2.0)

Copyright See file COPYRIGHTS

Encoding UTF-8

LinkingTo cpp11

Imports wk

Suggests geos, testthat (>= 3.0.0), knitr, rmarkdown, fasterize, terra, raster, ggplot2

URL https://github.com/hypertidy/controlledburn

BugReports https://github.com/hypertidy/controlledburn/issues

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

Depends R (>= 3.5)

NeedsCompilation yes

Author Michael Sumner [aut, cre] (ORCID: <https://orcid.org/0000-0002-2471-7511>), Noam Ross [aut]

(ORCID: <https://orcid.org/0000-0002-2136-0000>), EcoHealth Alliance [cph] (Original fasterize scanline logic), Jim Hester [ctb] (Wrote CollectorList.h for early version), Daniel Baston [cph] (exactextract C++ algorithm (vendored)), ISciences, LLC [cph] (exactextract C++ algorithm (vendored))

Maintainer Michael Sumner <mdsumner@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-10-24 11:00:21 UTC

RemoteUrl https://github.com/cran/controlledburn

RemoteRef HEAD

RemoteSha 2fc55adbf26c4d629fcaaa58aa810f1c111ed59c

Contents

burn 2
burn_sparse 4
crop_burn 5
extract_burn 6
materialise_chunk 7

Index **9**

---

burn *Sparse rasterization of polygon, line, and point geometry*

Description

Rasterizes geometry into a sparse output schema, with one type-pure table per geometry kind. Polygons produce run-length-encoded interior cells (`$runs`) and weighted boundary cells (`$edges`, with a coverage `fraction` in `[0, 1]`). Lines produce one record per touched cell with the absolute `length` of line within the cell, in CRS units (`$lines`). Points produce one record per touched cell, with no measure column (`$points`). See the unified geometry rasterization design doc in `inst/docs-design/`.

Usage

```
burn(
  x,
  extent = NULL,
  dimension = NULL,
  resolution = NULL,
  mode = c("coverage", "approx")
)

burn_scanline(x, extent = NULL, dimension = NULL, resolution = NULL)
```

Arguments

`x` geometry input, one of:

- a `geos_geometry` vector (from geos)
- an `sfc` geometry column (from sf)
- a `wk_wkb` vector (from wk)
- a list of raw vectors containing WKB

`extent` numeric vector `c(xmin, xmax, ymin, ymax)` defining the raster extent. If `NULL` (default), derived from the bounding box of `x`.

`dimension` integer vector `c(ncol, nrow)` defining the grid dimensions. If `NULL` (default), fitted to the extent with at most 256 cells along the longer axis, preserving aspect ratio. Mutually exclusive with `resolution`.

`resolution` numeric, cell size (scalar for square cells, or `c(dx, dy)`). If supplied, `dimension` is computed from `extent / resolution`. Mutually exclusive with `dimension`.

`mode` character, one of `"coverage"` (default) or `"approx"`. `"coverage"` computes exact coverage fractions for polygon boundary cells (the `$edges` table). `"approx"` uses the cell-center rule (fasterize semantics): a boundary cell is included as a full run cell iff the polygon edge crosses to the left of the cell center. No `$edges` are produced for polygons. Faster (skips traversal math), but approximate. Lines and points are unaffected by mode.

Details

Mixed geometry-kind input (e.g. a vector containing both polygons and lines) is permitted and produces tables populated for whichever kinds are present. Each table's measure column means exactly one thing — polygon `$edges$fraction` is dimensionless, `$lines$length` is in CRS units, points have no measure — so combining them in a single materialized matrix would silently mix units: run separate burns for each kind before `materialize_chunk()` (which sums whatever it is given) or `extract_burn()` with `fun = "sum"` (which refuses mixed kinds).

GeometryCollection input warns and is skipped. Curved types (CircularString, CompoundCurve, etc.) must be linearised by the caller before calling.

Memory usage is O(perimeter) rather than O(bounding-box area), so no `tile_size` parameter is required (in contrast to `burn_sparse()`, which is polygon-only and tile-bounded).

Value

A list with class `"controlledburn"` containing:

`runs` data.frame with columns `row`, `col_start`, `col_end`, `id` — run-length encoded polygon interior cells (always full coverage). Empty for line/point input.

`edges` data.frame with columns `row`, `col`, `fraction`, `id` — polygon boundary cells with partial coverage; `fraction` is in (0, 1). Empty for line/point input, or when `mode = "approx"`.

`lines` data.frame with columns `row`, `col`, `length`, `id` — line cells; `length` is the absolute length of the line within the cell, in CRS units. Empty for polygon/point input.

`points` data.frame with columns `row`, `col`, `id` — point cells; no measure column (a point is either in a cell or it isn't). Empty for polygon/line input.

`extent` the raster extent (`c(xmin, xmax, ymin, ymax)`)

`dimension` the grid dimensions (`c(ncol, nrow)`)

Row and column indices are 1-based. Row 1 is the top (ymax) row. The `id` column is a 1-based index into the input geometry vector.

The measure is computed in the CRS the geometry and grid are specified in — *not* the geodesic measure on the Earth. Geometry in degrees produces area in square degrees and length in degrees. For geodesic measures, project to an appropriate CRS first.

Examples

```
if (requireNamespace("geos", quietly = TRUE)) {
  library(geos)

  # Polygon: produces $runs (interior) and $edges (fraction in [0, 1])
  poly <- as_geos_geometry("POLYGON ((0.5 0.5, 2.5 0.5, 2.5 2.5, 0.5 2.5, 0.5 0.5))")
  r <- burn(poly, extent = c(0, 3, 0, 3), dimension = c(3, 3))

  # Approx mode: fasterize-style cell-center rule, runs only
  r <- burn(poly, extent = c(0, 3, 0, 3), dimension = c(3, 3), mode = "approx")

  # Line: produces $lines with absolute length in CRS units
  line <- as_geos_geometry("LINESTRING (0.5 0.5, 2.5 2.5)")
  r <- burn(line, extent = c(0, 3, 0, 3), dimension = c(3, 3))

  # Point: produces $points with row/col only (no measure column)
  pt <- as_geos_geometry("POINT (1.5 1.5)")
  r <- burn(pt, extent = c(0, 3, 0, 3), dimension = c(3, 3))

  # Defaults: extent from bbox, 256-cell fitted grid
  r <- burn(poly)

  # By resolution
  r <- burn(poly, resolution = 0.01)
}
```

`burn_sparse` *Deprecated: use burn() instead*

Description

`burn_sparse()` has been removed. Use `burn()` instead, which produces identical output for polygon input and also supports lines and points.

Usage

```
burn_sparse(...)
```

Arguments

...	ignored

Value

This function always errors; it never returns a value. Call `burn()` instead.

Examples

```
# burn_sparse() is deprecated and errors; use burn() instead
try(burn_sparse())
```

---

crop_burn — *Crop a controlledburn result to a target extent*

Description

Filters and clips the sparse tables (runs, edges, lines, points) to a sub-window defined by a target extent. Returns a new `controlledburn` object with offset row/col indices and snapped extent. No dense allocation — just data frame filtering.

Usage

```
crop_burn(x, target)
```

Arguments

`x`	A `controlledburn` object returned by `burn()`.
`target`	numeric vector `c(xmin, xmax, ymin, ymax)` defining the target window. Snapped outward to cell boundaries.

Value

A `controlledburn` object covering only the target window, with row/col indices re-based to 1.

Examples

```
if (requireNamespace("geos", quietly = TRUE)) {
  library(geos)
  poly <- as_geos_geometry("POLYGON ((1 1, 9 1, 9 9, 1 9, 1 1))")
  r <- burn(poly, extent = c(0, 10, 0, 10), dimension = c(100L, 100L))
  sub <- crop_burn(r, target = c(2, 5, 3, 7))
  sub
}
```

`extract_burn` *Extract values from a controlledburn result at point locations*

Description

Queries the sparse tables directly; no dense matrix is created. For points at cell centres this reproduces `materialize_chunk()` exactly: `extract_burn(x, xy, fun) == materialize_chunk(x, fun)[cbind(row, col)]`

Usage

```
extract_burn(x, xy, fun = c("id", "sum"))
```

Arguments

`x`	A `controlledburn` object returned by `burn()`.
`xy`	A two-column matrix or data.frame of x, y coordinates in the CRS of the burn.
`fun`	character, `"id"` (last geometry written wins, `NA` background) or `"sum"` (accumulated coverage/length/count, `0` background). Same semantics as `materialize_chunk()`.

Details

Interior runs are matched by row equality plus column interval containment; edges, lines, and points by exact (row, col) match. Cost is O(points + hits), independent of grid dimension.

Value

A numeric vector of length `nrow(xy)`. Points outside the grid extent are `NA` for both funs.

Examples

```
if (requireNamespace("geos", quietly = TRUE)) {
  library(geos)
  poly <- as_geos_geometry("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))")
  r <- burn(poly, extent = c(0, 10, 0, 10), dimension = c(10L, 10L))
  # value at three cell centres (fun = "id": geometry id, else NA)
  extract_burn(r, cbind(c(1.5, 5.5, 8.5), c(1.5, 5.5, 8.5)), fun = "id")
}
```

`materialise_chunk` *Materialise a controlledburn sparse result into a dense matrix*

Description

Converts the sparse run/edge/line tables from `burn()` into a dense matrix. With `target`, only the cells inside a sub-window of the burn grid are materialised: the sparse tables are first filtered and re-indexed by `crop_burn()` and only the sub-window is allocated, so a chunk of a very large burn grid can be realised without ever allocating the full grid.

Usage

```
materialise_chunk(x, fun = c("sum", "id"), target = NULL)

materialize_chunk(x, fun = c("sum", "id"), target = NULL)
```

Arguments

`x`	A `controlledburn` object returned by `burn()`.
`fun`	character, the pixel function: `"sum"` (default) Polygon runs contribute 1, edges contribute their coverage fraction, lines contribute length, points contribute 1. Values accumulate additively across overlapping geometries. Background is 0. Mixed geometry kinds are summed as-is, although the measures have different units. `"id"` Write the geometry id (`$runs$id`, `$edges$id`, etc.) into each cell. When geometries overlap, the last id wins. Background is `NA`.
`target`	optional numeric `c(xmin, xmax, ymin, ymax)`; when given, materialise only this sub-window of the burn grid.

Details

The sub-window is snapped outward to the burn grid's cell boundaries (see `crop_burn()`), so the returned matrix covers at least `target`. Use `crop_burn(x, target)` to recover the snapped grid's `extent` and `dimension` (for plotting with `rasterImage()`, say). The materialised chunk is identical to the corresponding sub-matrix of the full materialisation.

Value

A numeric matrix with dimensions `nrow x ncol` of the burn grid, or of the snapped `target` window.

Examples

```
if (requireNamespace("geos", quietly = TRUE)) {
  poly <- geos::as_geos_geometry("POLYGON ((1 1, 9 1, 9 9, 1 9, 1 1))")
  r <- burn(poly, extent = c(0, 10, 0, 10), dimension = c(100L, 100L))
  full <- materialise_chunk(r)
  chunk <- materialise_chunk(r, target = c(2, 5, 3, 7))
```

```
  dim(chunk)
  crop_burn(r, c(2, 5, 3, 7))$extent
}
```

Index

burn, 2
burn(), *4–7*
burn_scanline *(*burn*)*, 2
burn_sparse, 4
burn_sparse(), *3*

crop_burn, 5
crop_burn(), *7*

extract_burn, 6
extract_burn(), *3*

materialise_chunk, 7
materialize_chunk *(*materialise_chunk*)*, 7
materialize_chunk(), *3*, *6*