

Package ‘controlledburn’

October 28, 2026

Title Sparse Geometry Rasterization with Exact Coverage Fractions

Version 0.3.2

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 $O(\text{perimeter})$ in both time and memory, using the scan line fill attributed to Wylie et al. (1967) <[doi:10.1145/1465611.1465619](https://doi.org/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](https://doi.org/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 graphics, grDevices, stats, vaster, wk, ximage

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

URL <https://hypertidy.github.io/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 CRAN

Date/Publication 2026-10-28 10:20:02 UTC

Contents

burn	2
burn_sparse	4
crop_burn	5
extract_burn	6
materialise_chunk	7
overview_burn	8
plot.controlledburn	9
summary.controlledburn	10

Index	12
--------------	-----------

burn	<i>Sparse rasterization of polygon, line, and point geometry</i>
------	--

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

<code>x</code>	geometry input, one of: • a <code>geos_geometry</code> vector (from <code>geos</code>) • an <code>sfc_geometry</code> column (from <code>sf</code>) • a <code>wk_wkb</code> vector (from <code>wk</code>) • a list of raw vectors containing WKB
<code>extent</code>	numeric vector <code>c(xmin, xmax, ymin, ymax)</code> defining the raster extent. If <code>NULL</code> (default), derived from the bounding box of <code>x</code> .
<code>dimension</code>	integer vector <code>c(ncol, nrow)</code> defining the grid dimensions. If <code>NULL</code> (default), fitted to the extent with at most 256 cells along the longer axis, preserving aspect ratio. Mutually exclusive with <code>resolution</code> .
<code>resolution</code>	numeric, cell size (scalar for square cells, or <code>c(dx, dy)</code>). If supplied, <code>dimension</code> is computed from <code>extent / resolution</code> . Mutually exclusive with <code>dimension</code> .
<code>mode</code>	character, one of "coverage" (default) or "approx". "coverage" computes exact coverage fractions for polygon boundary cells (the <code>\$edges</code> 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 <code>\$edges</code> 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 $O(\text{perimeter})$ rather than $O(\text{bounding-box area})$, 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	<i>Crop a controlledburn result to a target extent</i>
-----------	--

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	<i>Extract values from a controlledburn result at point locations</i>
--------------	---

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

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

Details

Interior runs are matched by row equality plus column interval containment; edges, lines, and points by exact (row, col) match. Cost is $O(\text{points} + \text{hits})$, 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	<i>Materialise a controlledburn sparse result into a dense matrix</i>
-------------------	---

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

<code>x</code>	A controlledburn object returned by <code>burn()</code> .
<code>fun</code>	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 (<code>\$runs\$id</code> , <code>\$edges\$id</code> , etc.) into each cell. When geometries overlap, the last id wins. Background is NA.
<code>target</code>	optional numeric <code>c(xmin, xmax, ymin, ymax)</code> ; 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
  }

```

 overview_burn

Coarse overview of a burn

Description

Aggregate a `burn()` result onto a grid of square tiles of block cells and return the mean coverage per tile (in $[0, 1]$ for non-overlapping polygons). Tiles cut off by the grid boundary are normalised by their actual cell count. Cost is proportional to the number of records plus the number of tiles: a run that spans several tiles contributes partial lengths to its two end tiles, and the fully covered tiles between them are filled with a difference array, so nothing is expanded to cells whatever the block size.

Usage

```
overview_burn(x, tiles = 256L, block = NULL)
```

Arguments

<code>x</code>	A controlledburn object returned by <code>burn()</code> .
<code>tiles</code>	Target number of tiles along the longer grid axis; ignored if <code>block</code> is given.
<code>block</code>	Tile size in cells. <code>block = 1</code> gives the exact coverage, the same as <code>materialise_chunk()</code> for polygons, with NA for 0.

Details

The result is a matrix in raster order (row 1 is the top), ready for `ximage::ximage(ov, extent = attr(ov, "extent"))`.

Value

A numeric matrix (tile rows by tile columns) with attributes `extent` (`c(xmin, xmax, ymin, ymax)` of the tile grid, which extends past the burn grid on the right and bottom when `block` does not divide its dimension) and `block`. Untouched tiles are NA.

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(1000L, 1000L))
  ov <- overview_burn(r, tiles = 20)
  dim(ov)
  attr(ov, "extent")
}

```

plot.controlledburn *Plot a burn*

Description

Draw a [burn\(\)](#) result in map coordinates (asp = 1) with the grid extent as the frame. Two renderings are available:

Usage

```
## S3 method for class 'controlledburn'
plot(
  x,
  what = c("auto", "exact", "overview"),
  marginals = TRUE,
  col = grDevices::hcl.colors(64, "YlOrRd", rev = TRUE),
  by_id = FALSE,
  tiles = 256L,
  threshold = 50000,
  content = FALSE,
  main = NULL,
  ...
)
```

Arguments

<code>x</code>	A controlledburn object returned by burn() .
<code>what</code>	"auto", "exact", or "overview".
<code>marginals</code>	Draw the row and column marginal strips. The layout is sized from the extent's aspect ratio so the strips span exactly the map; leftover device space is left blank.
<code>col</code>	Colour ramp (low to high coverage) for "overview" and for edge shading in "exact".
<code>by_id</code>	In "exact" mode, colour cells by geometry id instead of by coverage (edge cells take their fraction as opacity).
<code>tiles</code>	Passed to overview_burn() for the "overview" rendering.
<code>threshold</code>	Record count above which "auto" switches to "overview".
<code>content</code>	Draw the content extent (bounding box of touched cells) as a dashed rectangle.
<code>main</code>	Optional title, drawn in the outer margin above the strips.
<code>...</code>	Passed to graphics::plot() for the main panel.

Details

"exact" every record is drawn: runs as rectangles, edge cells shaded by coverage fraction, line cells shaded by length (relative to the cell diagonal), points as cells. Cost is one rectangle per record, so this is the picture of the sparse representation itself.

"overview" the coarse mean-coverage matrix from `overview_burn()` drawn with `ximage::ximage()`. Cost does not depend on grid size.

"auto" (default) picks "exact" when the total record count is at most threshold, else "overview". Either way the optional marginal strips are exact: covered cells per grid row (right) and per grid column (top), from `summary.controlledburn()`.

Value

x, invisibly, with the summary computed for the marginals attached as attribute summary.

Examples

```
if (requireNamespace("geos", quietly = TRUE)) {
  poly <- geos::as_geos_geometry(c("POLYGON ((1 1, 9 1, 9 9, 1 9, 1 1))",
                                   "POLYGON ((2 2, 4 3, 3 5, 2 2))"))
  r <- burn(poly, extent = c(0, 10, 0, 10), dimension = c(40L, 40L))
  plot(r)
  plot(r, by_id = TRUE, marginals = FALSE)
  plot(r, what = "overview", tiles = 10)
}
```

summary.controlledburn

Summarise a burn without materialising it

Description

Compute a compact description of a `burn()` result in time proportional to the number of records in its sparse tables, never to the number of grid cells, so it is cheap even for a national grid of billions of cells.

Usage

```
## S3 method for class 'controlledburn'
summary(object, tiles = 256L, ...)

## S3 method for class 'summary.controlledburn'
print(x, ascii = FALSE, ...)
```

Arguments

object	A controlledburn object returned by burn() .
tiles	Target number of tiles along the longer grid axis for the overview, passed to overview_burn() .
...	Ignored.
x	A summary.controlledburn object.
ascii	Also print the overview as text.

Details

The pieces are:

overview a coarse matrix of mean coverage per tile, see [overview_burn\(\)](#). When the grid is small enough that one tile is one cell, this is the exact coverage.

row_marginal, col_marginal exact covered-cell totals per grid row and per grid column, at full resolution.

content the extent and 1-based row/col range of touched cells, or NULL for an empty burn.

area per-geometry totals in cells: interior (run cells), boundary (summed edge fractions), lines (summed length, in extent units), points (count).

fractions quantiles of the edge coverage fractions.

records row counts of the sparse tables.

Coverage here means: run cells 1 each, edge cells their fraction, line and point cells 1 each (presence).

The number of geometries is taken as the largest id present, so geometries after the last one that touched the grid are not counted.

Value

A list of class summary.controlledburn.

Examples

```
if (requireNamespace("geos", quietly = TRUE)) {
  poly <- geos::as_geos_geometry(c("POLYGON ((1 1, 9 1, 9 9, 1 9, 1 1))",
                                   "POLYGON ((2 2, 4 3, 3 5, 2 2))"))
  r <- burn(poly, extent = c(0, 10, 0, 10), dimension = c(100L, 100L))
  s <- summary(r)
  s
  s$area
}
```

Index

burn, [2](#)
burn(), [4–11](#)
burn_scanline (burn), [2](#)
burn_sparse, [4](#)
burn_sparse(), [3](#)

crop_burn, [5](#)
crop_burn(), [7](#)

extract_burn, [6](#)
extract_burn(), [3](#)

graphics::plot(), [9](#)

materialise_chunk, [7](#)
materialise_chunk(), [8](#)
materialize_chunk (materialise_chunk), [7](#)
materialize_chunk(), [3, 6](#)

overview_burn, [8](#)
overview_burn(), [9–11](#)

plot.controlledburn, [9](#)
print.summary.controlledburn
 (summary.controlledburn), [10](#)

summary.controlledburn, [10](#)
summary.controlledburn(), [10](#)

ximage::ximage(), [10](#)