

Package ‘wkpool’

October 12, 2026

Title Vertex Pool Topology for Well-Known Geometry

Version 0.4.0

Description Establishes and maintains vertex pool topology for geometry handled by 'wk'. Segments are the atomic unit, vertices are shared via integer references into a pool. Topology is made discoverable via coincident vertex detection while not requiring modification of the input data. Topological data models follow principles described in Worboys and Duckham (2004, ISBN:978-0415283755). The edge-based topology geometry decomposed into vertices and directed edge pairs is a simplification of the quad-edge case in Guibas & Stolfi (1985) <[doi:10.1145/282918.282923](https://doi.org/10.1145/282918.282923)>.

License MIT + file LICENSE

URL <https://github.com/hypertidy/wkpool>

BugReports <https://github.com/hypertidy/wkpool/issues>

Encoding UTF-8

Imports vctr, wk (>= 0.9.4)

Suggests decido, knitr, rmarkdown, RTriangle, silicate, spelling, testthat (>= 3.0.0), traipse

VignetteBuilder knitr

LinkingTo cpp11

Config/testthat/edition 3

Language en-GB

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Michael Sumner [aut, cre, cph]

Maintainer Michael Sumner <mdsumner@gmail.com>

Repository CRAN

Date/Publication 2026-10-12 11:00:02 UTC

Contents

arcs_to_wkb	2
arc_node_summary	3
as_arcs	4
as_pslg	5
classify_cycles	6
cycles_to_wkt	7
cycle_signed_area	8
establish_topology	9
find_arcs	10
find_cycles	11
find_internal_boundaries	12
find_neighbours	12
find_nodes	13
find_shared_edges	14
merge_coincident	15
new_wkpool	16
new_wkpool_tri	17
plot.wkpool	19
plot.wkpool_tri	19
pool_combine	20
pool_compact	21
pool_holes	22
pool_triangles	23
pool_triangulate	24
reverse_cycle	25
segments_to_wkt	26
topology_report	27
vec_c	27
vertex_degree	28
wkpool	29
wkpool-accessors	30
wkpool-crs	31
wk_handle.wkpool	32
Index	34

arcs_to_wkb	<i>Convert arcs to WKT linestrings</i>
-------------	--

Description

Convert arcs to WKT linestrings

Usage

```
arcs_to_wkb(x, quotient = FALSE, ...)
```

```
arcs_to_wkt(x, quotient = FALSE)
```

Arguments

x	A wkpool (ideally after merge_coincident)
quotient	Walk the quotient graph: duplicated shared boundaries collapse to a single undirected edge (first occurrence keeps its direction), so a boundary shared by two features becomes ONE arc rather than one per feature, and its interior vertices are degree-2 pass-throughs. This is the TopoJSON-style arc decomposition of a polygon layer. Default FALSE walks every segment.
...	Passed to <code>wk::as_wkb()</code>

Details

Each arc (maximal segment sequence between nodes) becomes a linestring.

Geometry is assembled natively via `wk::wk_linestring()` (WKB first, WKT derived from it): coordinates are emitted at full double precision, z is carried when the pool has it, and the pool's crs and geodesic flag are attached to the output.

Value

A wk_wkt vector of LINESTRING geometries

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
merged <- merge_coincident(pool)
arcs_to_wkt(merged)
```

arc_node_summary	<i>Summarize arc-node structure</i>
------------------	-------------------------------------

Description

Summarize arc-node structure

Usage

```
arc_node_summary(x, quotient = FALSE)
```

Arguments

x	A wkpool (ideally after merge_coincident)
quotient	Walk the quotient graph: duplicated shared boundaries collapse to a single undirected edge (first occurrence keeps its direction), so a boundary shared by two features becomes ONE arc rather than one per feature, and its interior vertices are degree-2 pass-throughs. This is the TopoJSON-style arc decomposition of a polygon layer. Default FALSE walks every segment.

Value

List with counts and degree distribution

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
merged <- merge_coincident(pool)
arc_node_summary(merged)
```

as_arcs	<i>Convert arcs to a wkpool of arc segments</i>
---------	---

Description

Convert arcs to a wkpool of arc segments

Usage

```
as_arcs(x, arc_id = TRUE, quotient = FALSE)
```

Arguments

x	A wkpool (ideally after merge_coincident)
arc_id	Logical: add .arc column to track arc membership?
quotient	Walk the quotient graph: duplicated shared boundaries collapse to a single undirected edge (first occurrence keeps its direction), so a boundary shared by two features becomes ONE arc rather than one per feature, and its interior vertices are degree-2 pass-throughs. This is the TopoJSON-style arc decomposition of a polygon layer. Default FALSE walks every segment.

Value

A wkpool with segments grouped by arc

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
merged <- merge_coincident(pool)
as_arcs(merged)
```

as_pslg

*Convert wkpool to a PSLG (planar straight line graph)***Description**

Convert wkpool to a PSLG (planar straight line graph)

Usage

```
as_pslg(x, ...)
```

Arguments

x	A wkpool (will be passed through merge_coincident if not already)
...	passed to merge_coincident

Details

Produces a planar straight line graph for constrained Delaunay triangulation. The vertex pool maps directly to P, and segment indices map directly to S (both 1-indexed). Shared boundaries appear once per feature.

A PSLG does not record which side of a segment is inside. For RTriangle, pass hole points from [pool_holes\(\)](#) as H. The triangles come back as vertex and index matrices, and [new_wkpool_tri\(\)](#) puts them on a vertex pool again; see [vignette\("triangulation", package = "wkpool"\)](#).

For ear cutting that keeps the feature of each triangle, see [pool_triangulate\(\)](#) (experimental).

Value

A list with P (vertex matrix) and S (segment matrix), the form used by RTriangle::pslg()

Examples

```
x <- wk::as_wkb("POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))")
pool <- establish_topology(x)
as_pslg(pool)
```

classify_cycles	<i>Classify cycles as outer rings or holes based on winding</i>
-----------------	---

Description

Classify cycles as outer rings or holes based on winding

Usage

```
classify_cycles(x, convention = c("sf", "ogc"))
```

Arguments

x	A wkpool (ideally after merge_coincident)
convention	Which winding convention to use when the pool has no path provenance: "sf" (default) or "ogc" • sf: negative area = outer, positive = hole • ogc: positive area = outer, negative = hole

Details

When the pool carries path provenance, ring roles are structural: the first ring of each part is the exterior and later rings are its holes, regardless of winding, and convention is not consulted. The signed area still reports the observed winding. Pools without provenance are classified by winding under the requested convention.

Value

A data frame with cycle index, signed area, and type (outer/hole); when path provenance is present, also .path and .feature

Examples

```
x <- wk::as_wkb(c(
  paste0(
    "MULTIPOLYGON (((0 0, 0 1, 0.75 1, 1 0.8, 0.5 0.7, ",
    "0.8 0.6, 0.69 0, 0 0), (0.2 0.2, 0.5 0.2, ",
    "0.5 0.4, 0.3 0.6, 0.2 0.4, 0.2 0.2)))",
    "MULTIPOLYGON (((0.69 0, 0.8 0.6, 1.1 0.63, 1.23 0.3, 0.69 0)))"
  ))

pool <- establish_topology(x)
merged <- merge_coincident(pool)
classify_cycles(merged)
```

<code>cycles_to_wkt</code>	<i>Convert cycles to WKT polygons</i>
----------------------------	---------------------------------------

Description

Convert cycles to WKT polygons

Usage

```
cycles_to_wkt(x, feature = TRUE, convention = c("sf", "ogc"))
cycles_to_wkb(x, feature = TRUE, convention = c("sf", "ogc"), ...)
```

Arguments

<code>x</code>	A wkpool (ideally after <code>merge_coincident</code>)
<code>feature</code>	Logical: attempt to reconstruct original features? If TRUE, groups cycles by .feature and nests holes in outers. If FALSE, each cycle becomes a separate polygon.
<code>convention</code>	Winding convention: "sf" (default) or "ogc"; only consulted for pools without path provenance
<code>...</code>	Passed to <code>wk::as_wkb()</code>

Details

Converts cycles back to polygons. When `feature = TRUE` and the pool carries path provenance (minted by `establish_topology()`), the original feature structure is reconstructed exactly: rings are grouped into their original part (exterior first, then that part's holes) and parts into their original feature (POLYGON or MULTIPOLYGON), with winding emitted as stored and convention not consulted. Pools without provenance fall back to a heuristic that groups rings by feature and nests all holes with the first outer.

Geometry is assembled natively via `wk::wk_polygon()` and `wk::wk_collection()` (WKB first, WKT derived from it): coordinates are emitted at full double precision, rings are closed by `wk`, `z` is carried when the pool has it, and the pool's crs and geodesic flag are attached to the output.

Value

A `wk_wkt` vector of POLYGON geometries

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
merged <- merge_coincident(pool)
```

`cycles_to_wkt(merged)`

<code>cycle_signed_area</code>	<i>Calculate signed area of a cycle</i>
--------------------------------	---

Description

Calculate signed area of a cycle

Usage

`cycle_signed_area(cycle, pool)`

Arguments

- | | |
|--------------------|---|
| <code>cycle</code> | Integer vector of .vx IDs forming a closed cycle |
| <code>pool</code> | Vertex pool data frame (from <code>pool_vertices</code>) |

Details

Uses the shoelace formula to compute signed area. The sign indicates winding direction. For typical geographic data (Simple Features convention):

- Negative area = outer ring
- Positive area = hole

This is intrinsic to the geometry: we observe winding, not declare it.

Value

Numeric signed area.

Examples

```
x <- wk::as_wkb("POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))")
pool <- establish_topology(x)
merged <- merge_coincident(pool)
cycles <- find_cycles(merged)
cycle_signed_area(cycles[[1]], pool_vertices(merged))
```

establish_topology	<i>Establish topology from any wk-handleable geometry</i>
--------------------	---

Description

Establish topology from any wk-handleable geometry

Usage

```
establish_topology(x, ...)
```

Arguments

x	Any geometry wk can handle (sf, sfc, wkb, wkt, geos, s2, xy, ...)
...	reserved for future use

Details

Vertices are minted as-is from the input coordinates. No snapping or deduplication is performed - the pool represents the exact truth of the input geometry. Use `merge_coincident()` afterwards to discover/establish shared topology.

Segments track their feature origin via `.feature` attribute, enabling discovery of shared boundaries and neighbour relations.

Segments also carry a minted `.path` id - one id per input ring or linestring - with a sidecar table (see `pool_paths()`) recording each path's feature, part and ring identity as captured from `wk::wk_coords()`. This provenance makes cycles, ring roles and feature structure recoverable exactly, in input order and input winding: see `find_cycles()`, `classify_cycles()` and `cycles_to_wkt()`.

The coordinate reference system of the input (as reported by `wk::wk_crs()`) and its geodesic flag (`wk::wk_is_geodesic()`) are captured on the pool and travel with it through subsetting, merging and compaction, and back out through the round-trip functions.

Value

A wkpool object (segments with vertex pool)

Examples

```
x <- wk::as_wkb(c(
  paste0(
    "MULTIPOLYGON (((0 0, 0 1, 0.75 1, 1 0.8, 0.5 0.7, ",
    "0.8 0.6, 0.69 0, 0 0), (0.2 0.2, 0.5 0.2, ",
    "0.5 0.4, 0.3 0.6, 0.2 0.4, 0.2 0.2)))",
    "MULTIPOLYGON (((0.69 0, 0.8 0.6, 1.1 0.63, 1.23 0.3, 0.69 0)))"
  ))
pool <- establish_topology(x)
```

find_arcs	<i>Find arcs (maximal segment sequences between nodes)</i>
-----------	--

Description

Find arcs (maximal segment sequences between nodes)

Usage

```
find_arcs(x, quotient = FALSE)
```

Arguments

x	A wkpool (ideally after merge_coincident)
quotient	Walk the quotient graph: duplicated shared boundaries collapse to a single undirected edge (first occurrence keeps its direction), so a boundary shared by two features becomes ONE arc rather than one per feature, and its interior vertices are degree-2 pass-throughs. This is the TopoJSON-style arc decomposition of a polygon layer. Default FALSE walks every segment.

Details

Arcs are maximal paths through degree-2 vertices. They start and end at nodes (degree != 2) or form closed loops through degree-2 vertices.

Value

A list of integer vectors, each containing .vx IDs forming an arc

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
merged <- merge_coincident(pool)
find_arcs(merged)
```

find_cycles	<i>Find closed cycles (rings) in segment graph</i>
-------------	--

Description

Find closed cycles (rings) in segment graph

Usage

```
find_cycles(x)
```

Arguments

`x` A wkpool (ideally after `merge_coincident`)

Details

When the pool carries path provenance (minted by `establish_topology()`), a cycle is a path whose segment chain closes: rings are recovered exactly, in input order and input winding, robust to segment reordering, and the result carries a path attribute mapping each cycle to its path id. Pools without provenance fall back to walking the segment graph in storage order, which relies on segments being consecutive within each ring.

Each cycle is a vector of vertex IDs in traversal order. The cycle is closed (first vertex connects back to last via a segment).

Value

A list of integer vectors, each containing .vx IDs forming a closed cycle

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
merged <- merge_coincident(pool)
find_cycles(merged)
```

```
find_internal_boundaries
```

Find internal boundaries (edges shared by exactly 2 features, opposite direction)

Description

Find internal boundaries (edges shared by exactly 2 features, opposite direction)

Usage

```
find_internal_boundaries(x)
```

Arguments

x A wkpool after merge_coincident

Details

Internal boundaries are edges where one feature has segment (v0 -> v1) and another feature has (v1 -> v0), i.e. they share the edge but traverse it in opposite directions. This is the defining characteristic of a shared polygon boundary.

Value

A wkpool containing only internal boundary segments

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
merged <- merge_coincident(pool)
find_internal_boundaries(merged)
```

```
find_neighbours
```

Build adjacency from shared edges

Description

Build adjacency from shared edges

Usage

```
find_neighbours(x, type = c("edge", "vertex"))
```

Arguments

x A wkpool after merge_coincident
type "edge" for features sharing an edge, "vertex" for features sharing any vertex

Value

Data frame of feature pairs that are neighbours

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
merged <- merge_coincident(pool)
find_neighbours(merged)
```

find_nodes	<i>Find nodes (vertices where degree != 2)</i>
------------	--

Description

Find nodes (vertices where degree != 2)

Usage

```
find_nodes(x, quotient = FALSE)
```

Arguments

x A wkpool (ideally after merge_coincident)
quotient Count in the quotient graph: duplicated shared boundaries (the same undirected vertex pair carried by more than one feature) count once. A vertex interior to a shared boundary has quotient degree 2 - the degree-2 invariant holds at this level - while true junctions (three or more distinct edges) stay nodes. Default FALSE counts every segment.

Details

Nodes are branch points (degree 3+) or endpoints (degree 1). Degree-2 vertices are pass-through points within an arc.

Value

Integer vector of .vx IDs that are nodes

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
merged <- merge_coincident(pool)
find_nodes(merged)
```

find_shared_edges	<i>Find shared edges between features</i>
-------------------	---

Description

Find shared edges between features

Usage

```
find_shared_edges(x)
```

Arguments

x A wkpool (ideally after merge_coincident)

Details

After merging coincident vertices, edges that were duplicated across features (e.g., shared polygon boundaries) will reference the same vertex pair. This function finds them and reports which features share each edge.

Value

Data frame of edges shared by multiple features

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
merged <- merge_coincident(pool)
find_shared_edges(merged)
```

merge_coincident	<i>Merge coincident vertices in a pool</i>
------------------	--

Description

Merge coincident vertices in a pool

Usage

```
merge_coincident(x, tolerance = 0, period_x = NULL)
```

Arguments

x	A wkpool, or a wkpool_tri (see pool_triangulate())
tolerance	Numeric tolerance for coordinate matching. Default 0 means exact match only.
period_x	NULL (default) for no wrapping, or a single positive number: x is treated as periodic with this period when deciding which vertices coincide (for example 360 for longitude in degrees). See 'Periodic x' below.

Details

With tolerance = 0, only exactly identical coordinates are merged - identity is the double-precision bit pattern (grouped at C level via vctrs), not a formatted representation, so values that differ only beyond printable precision remain distinct. The first occurrence becomes the canonical vertex; coordinates are never moved.

With a nonzero tolerance, coordinates are snapped to a grid of cell size tolerance for matching (the canonical vertex keeps its original coordinates). Note this is a snap grid, not a distance guarantee: two vertices within tolerance of each other that straddle a grid-cell boundary do not merge.

When the pool carries z, vertices are keyed on all coordinate columns: equal x/y with different z remain distinct vertices.

This is where you discover shared boundaries between polygons, network connectivity, mesh topology, etc.

Value

A wkpool (or wkpool_tri) with shared vertices merged (fewer unique vertices)

Periodic x

With period_x set, vertices are keyed on $x \% \text{period_x}$ (then on the tolerance snap grid, if any, wrapped again so that the grid cell at the period boundary is the same cell as the one at zero). This fuses the two copies of a seam, for example $x = -180$ and $x = 180$ for longitude, and with period_x = 360 data in a 0 to 360 convention fuses at 0 and 360 in the same way. Once fused, a seam is an ordinary internal boundary: [find_shared_edges\(\)](#), [find_internal_boundaries\(\)](#) and the quotient argument of [find_arcs\(\)](#) treat it like any other shared boundary.

The period is never inferred: wkpool carries the crs without interpreting it, so it cannot know that x is longitude. The decision to fuse is made here, explicitly.

Coordinates are still never moved: the canonical vertex keeps the x of its first occurrence (say -180), and segments from the other side of the seam now reference it. The topology is correct on the cylinder, but geometry emitted from the pool (for example by `cycles_to_wkb()`) has coordinate jumps of about one period across the seam. Consumers that understand wrapping (s2, great-circle densification) are fine; a planar consumer needs the seam split again downstream.

Segments are identified by their two vertices, so on the cylinder a segment cannot say which way round it goes. Two segments whose ends fuse to the same pair but which run opposite ways around become the same edge, for example the edge from -180 to 0 in one feature and from 0 to 180 in another. This needs a segment that spans at least half the period, so densify long segments before merging with `period_x`.

Only x is periodic. Poles (where all x coincide on the sphere), periodic y, and any detection from the crs are out of scope.

Feature provenance (`.feature`) is preserved - segments keep their original feature assignment even after vertex merging.

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
topology_report(pool)
merged <- merge_coincident(pool)
topology_report(merged)

# two hemispheres, each with its own copy of the antimeridian
g <- wk::as_wkb(c(
  "POLYGON ((-180 -60, 0 -60, 0 60, -180 60, -180 -60))",
  "POLYGON ((0 -60, 180 -60, 180 60, 0 60, 0 -60))"
))
pool <- establish_topology(g)
nrow(pool_vertices(merge_coincident(pool)))
fused <- merge_coincident(pool, period_x = 360)
nrow(pool_vertices(fused))
find_shared_edges(fused)
```

new_wkpool

Construct a wkpool from validated components

Description

A low-level constructor for wkpool objects, for use by packages that build pools directly (for example after refining or transforming an existing pool). Inputs are checked against the wkpool invariants: every segment endpoint must be a vertex id present in the pool.

Usage

```
new_wkpool(
  vertices,
  vx0,
  vx1,
  feature = NULL,
  path = NULL,
  paths = NULL,
  crs = NULL,
  geodesic = FALSE
)
```

Arguments

vertices	A data.frame with columns .vx (integer vertex ids), x, y, and optionally z.
vx0, vx1	Integer vectors of segment start/end vertex ids, each value present in vertices\$.vx.
feature	Optional integer vector of feature ids, one per segment.
path	Optional integer vector of path ids, one per segment. A path is a maximal run of segments minted from one input ring or linestring; it is the provenance that lets cycles, ring roles and feature structure be recovered exactly.
paths	Optional data.frame describing the paths: columns .path, .feature, .part, .ring (as captured from <code>wk::wk_coords()</code> identifiers by <code>establish_topology()</code>).
crs	A CRS object (commonly an authority string such as "EPSG:4326"), carried but never interpreted, per wk's CRS propagation model. Use NULL for none, or <code>wk::wk_crs_inherit()</code> .
geodesic	TRUE if segments should be interpreted as geodesics when coordinates are spherical, FALSE otherwise, NA to inherit.

Value

A wkpool object.

Examples

```
v <- data.frame(.vx = 1:3, x = c(0, 1, 1), y = c(0, 0, 1))
new_wkpool(v, vx0 = c(1L, 2L), vx1 = c(2L, 3L), crs = "EPSG:4326")
```

new_wkpool_tri

Construct a triangle pool

Description

A low-level constructor for wkpool_tri objects: a vector of triangles whose corners are vertex ids into a shared vertex pool, the same pool structure a `wkpool` uses for segments.

Usage

```
new_wkpool_tri(
  vertices,
  vx0,
  vx1,
  vx2,
  feature = NULL,
  part = NULL,
  paths = NULL,
  crs = NULL,
  geodesic = FALSE
)
```

Arguments

<code>vertices</code>	A data.frame with columns <code>.vx</code> (integer vertex ids), <code>x</code> , <code>y</code> , and optionally <code>z</code> .
<code>vx0, vx1, vx2</code>	Integer vectors of triangle corner vertex ids, each value present in <code>vertices\$.vx</code> .
<code>feature</code>	Optional integer vector of feature ids, one per triangle.
<code>part</code>	Optional integer vector of part ids, one per triangle (the polygon within a multi-polygon feature, as identified by wk::wk_coords()).
<code>paths</code>	Optional data.frame describing the input paths (see pool_paths()), carried from the segment pool.
<code>crs</code>	A CRS object (commonly an authority string such as "EPSG:4326"), carried but never interpreted, per wk's CRS propagation model. Use NULL for none, or wk::wk_crs_inherit() .
<code>geodesic</code>	TRUE if segments should be interpreted as geodesics when coordinates are spherical, FALSE otherwise, NA to inherit.

Details

The `wkpool_tri` class is experimental: how a triangle records the polygon part it came from (currently a `.part` field) may change in a later release.

Value

A `wkpool_tri` object.

Examples

```
v <- data.frame(.vx = 1:4, x = c(0, 1, 1, 0), y = c(0, 0, 1, 1))
new_wkpool_tri(v, vx0 = c(1L, 1L), vx1 = c(2L, 3L), vx2 = c(3L, 4L))
```

plot.wkpool	<i>Plot a wkpool object</i>
-------------	-----------------------------

Description

Draws segments coloured by feature membership.

Usage

```
## S3 method for class 'wkpool'
plot(x, col = NULL, ...)
```

Arguments

x	A wkpool object.
col	Colour(s) for segments. If NULL (default), segments are coloured by feature using a built-in palette.
...	Further arguments passed to plot.default() .

Value

Invisibly returns x.

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
plot(pool)
```

plot.wkpool_tri	<i>Plot a triangle pool</i>
-----------------	-----------------------------

Description

Draws every triangle with one call to [graphics::polygon\(\)](#), indexing the vertex pool directly (triangles separated by NA). This avoids building geometry: on 40,000 triangles it draws in a fraction of a second, where casting to wk polygons and plotting those takes over a minute.

Usage

```
## S3 method for class 'wkpool_tri'
plot(x, col = NULL, border = "black", add = FALSE, xlab = "x", ylab = "y", ...)
```

Arguments

x	A wkpool_tri.
col	Fill colour(s), recycled over triangles. If NULL (default), triangles are coloured by feature using the same palette as <code>plot.wkpool()</code> .
border	Border colour(s) for triangle edges, NA for none.
add	If TRUE, draw on the current plot.
xlab, ylab	Axis labels when add = FALSE.
...	Further arguments passed to <code>plot.default()</code> when add = FALSE.

Value

Invisibly returns x.

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0)), (0.2 0.3, 0.2 0.5, 0.4 0.5, 0.2 0.3))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
if (requireNamespace("decido", quietly = TRUE)) {
  tri <- pool_triangulate(establish_topology(x))
  plot(tri)
  plot(tri, col = "grey", border = NA)
}
```

pool_combine

Combine wkpool objects

Description

Combine wkpool objects

Usage

```
pool_combine(...)
```

Arguments

... wkpool objects to combine

Value

A single wkpool with merged pools and remapped segments

Examples

```
x <- wk::as_wkb("POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))")
y <- wk::as_wkb("POLYGON ((2 0, 3 0, 3 1, 2 1, 2 0))")
pool_a <- establish_topology(x)
pool_b <- establish_topology(y)
pool_combine(pool_a, pool_b)
```

pool_compact

*Compact a pool by removing unreferenced vertices***Description**

Compact a pool by removing unreferenced vertices

Usage

```
pool_compact(x)
```

Arguments

x A wkpool

Value

A wkpool with only referenced vertices, .vx remapped

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
merged <- merge_coincident(pool)
compact <- pool_compact(merged)
nrow(pool_vertices(compact))
```

pool_holes

Get hole points for constrained triangulation

Description

pool_holes() and hole_points() are the same function: pool_holes() is the name that matches the other pool_*() accessors, and hole_points() is kept for existing code.

Usage

```
pool_holes(x, convention = c("sf", "ogc"))
```

```
hole_points(x, convention = c("sf", "ogc"))
```

Arguments

x	A wkpool (ideally after merge_coincident)
convention	Which winding convention to use when the pool has no path provenance: "sf" (default) or "ogc"

Details

For use with RTriangle::triangulate(). Each row is a point inside a hole, which tells the triangulator to exclude that region.

The point is found on a horizontal line across the hole: the middle of the widest stretch of that line that starts at the hole's boundary and crosses no segment of the pool. It is inside the hole even when the hole is not convex, where the centroid of the ring's vertices can fall outside it (and a triangulator would then erase the polygon around the hole instead), and it is clear of any island in the hole.

A hole that is exactly filled by another polygon in the pool (an enclave in a coverage, such as a lake island that is its own feature) is not a hole of the pool and is not returned.

When the pool carries path provenance, holes are identified structurally (non-first rings of their part) and convention is not consulted.

Value

A matrix of hole points, one point inside each hole cycle, or NULL if no holes

Examples

```
polygons_with_holes <- wk::as_wkb(c(
  paste0(
    "MULTIPOLYGON (((0 0, 0 1, 0.75 1, 1 0.8, 0.5 0.7, ",
    "0.8 0.6, 0.69 0, 0 0), (0.2 0.2, 0.5 0.2, ",
    "0.5 0.4, 0.3 0.6, 0.2 0.4, 0.2 0.2)))"),
  "MULTIPOLYGON (((0.69 0, 0.8 0.6, 1.1 0.63, 1.23 0.3, 0.69 0)))"
))
```

```

x <- establish_topology(polygons_with_holes)
merged <- merge_coincident(x)
pool_holes(merged)
if (requireNamespace("RTriangle", quietly = TRUE)) {
  pslg <- as_pslg(merged)
  holes <- pool_holes(merged)
  tri <- RTriangle::triangulate(
    RTriangle::pslg(P = pslg$P, S = pslg$S, H = holes)
  )
}

```

pool_triangles	<i>Access the triangles of a triangle pool</i>
----------------	--

Description

Access the triangles of a triangle pool

Usage

```

pool_triangles(x)

tri_segments(x)

## S3 method for class 'wkpool_tri'
wk_handle(handleable, handler, ...)

```

Arguments

x	A wkpool_tri.
handleable	A wkpool_tri.
handler	A wk handler .
...	Passed on.

Details

A wkpool_tri is wk-handleable: it presents as one POLYGON per triangle (the handled vector has the same length as the triangle vector), with the pool's crs and geodesic flag attached.

Value

- pool_triangles(): a data frame with columns .vx0, .vx1, .vx2 and, when present, .feature and .part.
- tri_segments(): a [wkpool\(\)](#) of the triangle edges, three per triangle in triangle order and counter-clockwise, on the same vertex pool, with the triangle's feature. Edges used by one triangle are on the boundary of the triangulated region, edges used by two are internal (after [merge_coincident\(\)](#), so that adjacent triangles share vertex ids).

Examples

```
v <- data.frame(vx = 1:4, x = c(0, 1, 1, 0), y = c(0, 0, 1, 1))
tri <- new_wkpool_tri(v, vx0 = c(1L, 1L), vx1 = c(2L, 3L), vx2 = c(3L, 4L))
pool_triangles(tri)
tri_segments(tri)
```

pool_triangular	<i>Triangulate the polygons in a pool</i>
-----------------	---

Description

Triangulate each polygon of a `wkpool()` (built by `establish_topology()`) and return the triangles as a `wkpool_tri` on the same vertex pool.

Usage

```
pool_triangular(x, engine = c("decido"), ...)
```

Arguments

<code>x</code>	A <code>wkpool</code> with path provenance (from <code>establish_topology()</code>).
<code>engine</code>	Triangulation engine, currently only "decido".
<code>...</code>	Unused.

Details

The `wkpool_tri` class is experimental: how a triangle records the polygon part it came from (currently a `.part` field) may change in a later release.

The "decido" engine is ear cutting (`decido:earcut()`), which never creates vertices: every triangle corner is an existing vertex of the pool, so the vertex table is passed through unchanged. Each polygon (each part of a multipolygon) is triangulated separately, with its first ring as the boundary and later rings as holes, using the path provenance recorded by `establish_topology()`. Paths that are not closed rings (linestrings) are ignored.

The pool can be triangulated before or after `merge_coincident()`. On a raw pool each ring's closing coordinate is its own vertex; it is dropped from the ring and never used by a triangle. Merging the triangle pool later with `merge_coincident()` gives adjacent features shared triangle corners.

Triangles are counter-clockwise.

For constrained Delaunay triangulation, with Steiner points added for quality or area, pass `as_pslg()` to `RTriangle`; see `vignette("triangulation", package = "wkpool")`.

Value

A `wkpool_tri`.

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0)), (0.2 0.2, 0.2 0.4, 0.4 0.4, 0.2 0.2))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
if (requireNamespace("decido", quietly = TRUE)) {
  tri <- pool_triangulate(pool)
  tri
  pool_triangles(tri)
  wk::as_wkt(tri)
}
```

reverse_cycle

*Reverse a cycle's winding direction***Description**

Reverse a cycle's winding direction

Usage

```
reverse_cycle(cycle)
```

Arguments

cycle Integer vector of .vx IDs forming a closed cycle

Details

Flips the traversal direction of a cycle without changing coordinates. Useful for converting between winding conventions.

Value

The same vertices in reversed order (opposite winding)

Examples

```
x <- wk::as_wkb("POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))")
pool <- establish_topology(x)
merged <- merge_coincident(pool)
cycles <- find_cycles(merged)
reverse_cycle(cycles[[1]])
```

segments_to_wkt	<i>Convert wkpool segments to WKT</i>
-----------------	---------------------------------------

Description

Convert wkpool segments to WKT

Usage

```
segments_to_wkt(x, type = c("multilinestring", "linestring", "point"))
segments_to_wkb(x, type = c("multilinestring", "linestring", "point"), ...)
```

Arguments

x	A wkpool
type	Output type: "linestring" (segments as paths), "multilinestring" (all segments), or "point" (vertices only)
...	Passed to wk::as_wkb()

Details

Geometry is assembled natively via [wk::wk_linestring\(\)](#) and [wk::wk_collection\(\)](#) (WKB first, WKT derived from it): coordinates are emitted at full double precision, z is carried when the pool has it, and the pool's crs and geodesic flag are attached to the output.

Value

A wk_wkt vector

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
segments_to_wkt(pool)
segments_to_wkt(pool, type = "linestring")
```

topology_report	<i>Report topology diagnostics</i>
-----------------	------------------------------------

Description

Report topology diagnostics

Usage

```
topology_report(x, tolerance = 1e-08, period_x = NULL)
```

Arguments

x	A wkpool
tolerance	Tolerance for "near miss" detection
period_x	NULL, or the period of x, as in merge_coincident() ; duplicates and near misses are then counted with x wrapped.

Value

List of diagnostic information

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
topology_report(pool)
```

vec_c	<i>Combine many wkpool vectors into one vector</i>
-------	--

Description

This is non-functional, wkpool does not currently support [vec_c\(\)](#).

Usage

```
## S3 method for class 'wkpool'
vec_c(..., .ptype = NULL)
```

Arguments

- ... Vectors to coerce.
- .ptype If NULL, the default, the output type is determined by computing the common type across all elements of ...
Alternatively, you can supply .ptype to give the output known type. If `getOption("vctrs.no_guessing")` is TRUE you must supply this value: this is a convenient way to make production code demand fixed types.

Details

Attempts to combine wkpool vectors with `vec_c` will suggest using `[pool_combine()]` instead.

Value

nothing, used for a message side-effect (see Details)

See Also

[pool_combine\(\)](#)

vertex_degree	<i>Calculate vertex degree (number of segments touching each vertex)</i>
---------------	--

Description

Calculate vertex degree (number of segments touching each vertex)

Usage

`vertex_degree(x, quotient = FALSE)`

Arguments

- x A wkpool (ideally after `merge_coincident`)
- quotient Count in the quotient graph: duplicated shared boundaries (the same undirected vertex pair carried by more than one feature) count once. A vertex interior to a shared boundary has quotient degree 2 - the degree-2 invariant holds at this level - while true junctions (three or more distinct edges) stay nodes. Default FALSE counts every segment.

Value

Named integer vector: names are `.vx`, values are degree

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
merged <- merge_coincident(pool)
vertex_degree(merged)
```

 wkpool

Create a wkpool from a vertex table and a segment table

Description

The user-level constructor: give a vertex table and a segment table (as returned by [pool_vertices\(\)](#) and [pool_segments\(\)](#)) and get a wkpool back. It is a thin wrapper around [new_wkpool\(\)](#), which checks that every segment endpoint is a vertex id present in the pool.

Usage

```
wkpool(vertices, segments, paths = NULL, crs = NULL, geodesic = FALSE)
```

Arguments

vertices	A data.frame with columns .vx (integer vertex ids), x, y, and optionally z.
segments	A data.frame with integer columns .vx0 and .vx1 (vertex ids), and optionally .feature and .path.
paths	Optional data.frame describing the paths: columns .path, .feature, .part, .ring (as captured from wk::wk_coords() identifiers by establish_topology()).
crs	A CRS object (commonly an authority string such as "EPSG:4326"), carried but never interpreted, per wk's CRS propagation model. Use NULL for none, or wk::wk_crs_inherit() .
geodesic	TRUE if segments should be interpreted as geodesics when coordinates are spherical, FALSE otherwise, NA to inherit.

Value

A wkpool object.

See Also

[new_wkpool\(\)](#) for the low-level constructor, [establish_topology\(\)](#) to build a pool from wk geometry.

Examples

```
v <- data.frame(.vx = 1:4, x = c(0, 1, 1, 0), y = c(0, 0, 1, 1))
s <- data.frame(.vx0 = 1:4, .vx1 = c(2:4, 1L), .feature = 1L)
pool <- wkpool(v, s)
pool

# round trip through the accessors
wkpool(pool_vertices(pool), pool_segments(pool))
```

wkpool-accessors

*Access components of a wkpool object***Description**

Extract the vertex pool, segment table, or feature vector from a wkpool.

Usage

```
pool_vertices(x)

pool_segments(x)

pool_feature(x)

pool_path(x)

pool_paths(x)
```

Arguments

x A wkpool object.

Value

- `pool_vertices()`: A data frame with columns `.vx` (vertex ID), `x`, `y`, and optionally `z`.
- `pool_segments()`: A data frame with columns `.vx0`, `.vx1`, and optionally `.feature` and `.path`.
- `pool_feature()`: An integer vector of feature IDs, or `NULL` if no feature information is present.
- `pool_path()`: An integer vector of path IDs (one per segment), or `NULL` if no path provenance is present.
- `pool_paths()`: A data frame describing each path (`.path`, `.feature`, `.part`, `.ring`, as captured from `wk::wk_coords()` identifiers), or `NULL`.

Examples

```
x <- wk::as_wkb(c(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  "POLYGON ((1 0, 2 0, 2 1, 1 1, 1 0))"
))
pool <- establish_topology(x)
pool_vertices(pool)
pool_segments(pool)
pool_feature(pool)
```

 wkpool-crs

CRS and geodesic handling for wkpool

Description

A wkpool participates in wk's CRS propagation model: the CRS is an opaque object carried in an attribute, never interpreted, and checked for equality when pools are combined. Likewise the geodesic flag records whether segments should be interpreted as geodesics when coordinates are spherical. Both are captured from the input by [establish_topology\(\)](#), survive subsetting, merging and compaction, and are restored onto geometry produced by [segments_to_wkt\(\)](#), [arcs_to_wkt\(\)](#), [cycles_to_wkt\(\)](#) and their WKB counterparts.

Usage

```
## S3 method for class 'wkpool'
wk_crs(x)

## S3 method for class 'wkpool'
wk_set_crs(x, crs)

## S3 method for class 'wkpool'
wk_is_geodesic(x)

## S3 method for class 'wkpool'
wk_set_geodesic(x, geodesic)
```

Arguments

x	A wkpool object.
crs	A CRS object (commonly an authority string such as "EPSG:4326"), or NULL.
geodesic	TRUE, FALSE, or NA (inherit).

Value

- `wk_crs()`: the CRS object, or NULL.
- `wk_set_crs()`, `wk_set_geodesic()`: x with the attribute replaced.
- `wk_is_geodesic()`: TRUE, FALSE, or NA.

Examples

```
x <- wk::wkt("LINESTRING (0 0, 1 1)", crs = "EPSG:4326")
pool <- establish_topology(x)
wk::wk_crs(pool)
wk::wk_crs(wk::wk_set_crs(pool, "EPSG:3031"))
```

wk_handle.wkpool	<i>Handle wkpool objects with wk handlers</i>
------------------	---

Description

A wkpool is wk-handleable: it presents as one LINESTRING per segment (so the handled vector has the same length as the pool vector), with the pool's crs and geodesic flag attached. This means the wk ecosystem's generic conversions work directly on a pool: `wk::as_wkb()`, `wk::as_wkt()`, `wk::wk_coords()`, `geos::as_geos_geometry()`, plotting, and anything else built on [wk::wk_handle\(\)](#).

Usage

```
## S3 method for class 'wkpool'
wk_handle(handleable, handler, ...)
```

Arguments

handleable	A wkpool object.
handler	A wk handler .
...	Passed on.

Details

Vertices are emitted at full double precision, indexed directly from the pool (no text round trip). If the pool carries z, coordinates are emitted as XYZ.

Arcs and cycles remain explicit conversions ([arcs_to_wkt\(\)](#), [cycles_to_wkt\(\)](#) and friends): the handled form is deliberately the segment vector itself.

Value

The result of the handler.

Examples

```
x <- wk::wkt(
  "POLYGON ((0 0, 1 0, 1 1, 0 1, 0 0))",
  crs = "EPSG:4326"
)
pool <- establish_topology(x)
wk::as_wkt(pool)
```

wk_handle.wkpool

33

```
wk::wk_coords(pool)
wk::wk_crs(wk::as_wkb(pool))
```

Index

`arc_node_summary`, 3
`arcs_to_wkb`, 2
`arcs_to_wkt` (`arcs_to_wkb`), 2
`arcs_to_wkt`(), 31, 32
`as_arcs`, 4
`as_pslg`, 5
`as_pslg`(), 24

`classify_cycles`, 6
`classify_cycles`(), 9
`cycle_signed_area`, 8
`cycles_to_wkb` (`cycles_to_wkt`), 7
`cycles_to_wkb`(), 16
`cycles_to_wkt`, 7
`cycles_to_wkt`(), 9, 31, 32

`decido::earcut`(), 24

`establish_topology`, 9
`establish_topology`(), 7, 11, 17, 24, 29, 31

`find_arcs`, 10
`find_arcs`(), 15
`find_cycles`, 11
`find_cycles`(), 9
`find_internal_boundaries`, 12
`find_internal_boundaries`(), 15
`find_neighbours`, 12
`find_nodes`, 13
`find_shared_edges`, 14
`find_shared_edges`(), 15

`graphics::polygon`(), 19

`hole_points` (`pool_holes`), 22

`merge_coincident`, 15
`merge_coincident`(), 23, 24, 27

`new_wkpool`, 16
`new_wkpool`(), 29

`new_wkpool_tri`, 17
`new_wkpool_tri`(), 5

`plot.default`(), 19, 20
`plot.wkpool`, 19
`plot.wkpool`(), 20
`plot.wkpool_tri`, 19
`pool_combine`, 20
`pool_combine`(), 28
`pool_compact`, 21
`pool_feature` (`wkpool-accessors`), 30
`pool_holes`, 22
`pool_holes`(), 5
`pool_path` (`wkpool-accessors`), 30
`pool_paths` (`wkpool-accessors`), 30
`pool_paths`(), 9, 18
`pool_segments` (`wkpool-accessors`), 30
`pool_segments`(), 29
`pool_triangles`, 23
`pool_triangulate`, 24
`pool_triangulate`(), 5, 15
`pool_vertices` (`wkpool-accessors`), 30
`pool_vertices`(), 29

`reverse_cycle`, 25

`segments_to_wkb` (`segments_to_wkt`), 26
`segments_to_wkt`, 26
`segments_to_wkt`(), 31

`topology_report`, 27
`tri_segments` (`pool_triangles`), 23

`vec_c`, 27
`vec_c`(), 27
`vertex_degree`, 28

`wk handler`, 23, 32
`wk::as_wkb`(), 3, 7, 26
`wk::wk_collection`(), 7, 26
`wk::wk_coords`(), 9, 17, 18, 29, 30

`wk::wk_crs()`, [9](#)
`wk::wk_crs_inherit()`, [17](#), [18](#), [29](#)
`wk::wk_handle()`, [32](#)
`wk::wk_is_geodesic()`, [9](#)
`wk::wk_linestring()`, [3](#), [26](#)
`wk::wk_polygon()`, [7](#)
`wk_crs.wkpool (wkpool-crs)`, [31](#)
`wk_handle.wkpool`, [32](#)
`wk_handle.wkpool_tri (pool_triangles)`,
[23](#)
`wk_is_geodesic.wkpool (wkpool-crs)`, [31](#)
`wk_set_crs.wkpool (wkpool-crs)`, [31](#)
`wk_set_geodesic.wkpool (wkpool-crs)`, [31](#)
`wkpool`, [17](#), [29](#)
`wkpool()`, [23](#), [24](#)
`wkpool-accessors`, [30](#)
`wkpool-crs`, [31](#)