

# Package ‘wkpool’

May 8, 2026

**Title** Vertex Pool Topology for Well-Known Geometry

**Version** 0.3.0

**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

**RoxygenNote** 7.3.3

**Imports** vctrs, wk (>= 0.9.4)

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

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Language** en-GB

**NeedsCompilation** no

**Author** Michael Sumner [aut, cre]

**Maintainer** Michael Sumner <[mdsumner@gmail.com](mailto:mdsumner@gmail.com)>

**Repository** CRAN

**Date/Publication** 2026-03-05 19:10:02 UTC

## Contents

arcs_to_wkb . . . . .	2
arc_node_summary . . . . .	3
as_arcs . . . . .	4
as_pslg . . . . .	4
classify_cycles . . . . .	5
cycles_to_wkt . . . . .	6
cycle_signed_area . . . . .	7
establish_topology . . . . .	8
find_arcs . . . . .	9
find_cycles . . . . .	9
find_internal_boundaries . . . . .	10
find_neighbours . . . . .	11
find_nodes . . . . .	12
find_shared_edges . . . . .	12
hole_points . . . . .	13
merge_coincident . . . . .	14
plot.wkpool . . . . .	15
pool_combine . . . . .	16
pool_compact . . . . .	16
reverse_cycle . . . . .	17
segments_to_wkt . . . . .	18
topology_report . . . . .	18
vec_c . . . . .	19
vertex_degree . . . . .	20
wkpool-accessors . . . . .	20
<b>Index</b>	<b>22</b>

---

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

---

### Description

Convert arcs to WKT linestrings

### Usage

```
arcs_to_wkb(x, ...)
```

```
arcs_to_wkt(x)
```

### Arguments

x	A wkpool (ideally after merge_coincident)
...	Passed to <code>wk::as_wkb()</code>

**Details**

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

**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)
```

**Arguments**

x                    A wkpool (ideally after merge\_coincident)

**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)
```

**Arguments**

x	A wkpool (ideally after merge_coincident)
arc_id	Logical: add .arc column to track arc membership?

**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	<i>Convert wkpool to RTriangle pslg format</i>
---------	--

---

**Description**

Convert wkpool to RTriangle pslg format

**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 suitable for constrained triangulation with RTriangle. The vertex pool maps directly to P, and segment indices map directly to S (both 1-indexed).

Note: Hole detection is not currently supported. The returned pslg does not include hole points (H). For polygons with holes, you may need to identify hole points manually using `find_cycles()` and `cycle_signed_area()`.

**Value**

A list with P (vertex matrix) and S (segment matrix) for `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)
```

---

<code>classify_cycles</code>	<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**

- |                         |  |
|-------------------------|--|
| <code>x</code>          | A wkpool (ideally after <code>merge_coincident</code> )  |
| <code>convention</code> | Which winding convention to use: "sf" (default) or "ogc" <ul style="list-style-type: none"> <li>• sf: negative area = outer, positive = hole</li> <li>• ogc: positive area = outer, negative = hole</li> </ul> |

**Value**

A data frame with cycle index, signed area, and type (outer/hole)

**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)
```

---

cycles\_to\_wkt

*Convert cycles to WKT polygons*


---

**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**

x	A wkpool (ideally after merge_coincident)
feature	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.
convention	Winding convention: "sf" (default) or "ogc"
...	Passed to <code>wk::as_wkb()</code>

**Details**

Converts cycles back to polygons. When feature = TRUE, attempts to reconstruct original polygon structure by grouping rings by feature and nesting holes within their containing outer ring.

**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)
```

---

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

---

**Description**

Calculate signed area of a cycle

**Usage**

```
cycle_signed_area(cycle, pool)
```

**Arguments**

cycle	Integer vector of .vx IDs forming a closed cycle
pool	Vertex pool data frame (from pool_vertices)

**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      *Establish topology from any wk-handleable geometry*

---

## 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()` afterward to discover/establish shared topology.

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

## 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)
```

**Arguments**

x                    A wkpool (ideally after merge\_coincident)

**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**

Walks the segment graph to discover closed loops. Relies on segments being in consecutive order within each ring (as produced by establish\_topology).

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	<i>Build adjacency from shared edges</i>
-----------------	--

---

**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)
```

**Arguments**

x                    A wkpool (ideally after merge\_coincident)

**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)
```

---

hole\_points

*Get hole points for constrained triangulation*


---

**Description**

Get hole points for constrained triangulation

**Usage**

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

**Arguments**

x                    A wkpool (ideally after merge\_coincident)  
convention          Which winding convention to use: "sf" (default) or "ogc"

**Details**

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

**Value**

A matrix of hole points (centroids of hole cycles), 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)
hole_points(merged)
if (requireNamespace("RTriangle", quietly = TRUE)) {
  pslg <- as_pslg(merged)
  holes <- hole_points(merged)
  tri <- RTriangle::triangulate(
    RTriangle::pslg(P = pslg$P, S = pslg$S, H = holes)
  )
}
```

---

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

---

**Description**

Merge coincident vertices in a pool

**Usage**

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

**Arguments**

x	A wkpool
tolerance	Numeric tolerance for coordinate matching. Default 0 means exact match only.

**Details**

With tolerance = 0, only exactly identical coordinates are merged. The first occurrence becomes the canonical vertex.

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

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

**Value**

A wkpool with shared vertices merged (fewer unique vertices)

**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)
```

---

plot.wkpool

*Plot a wkpool object*


---

**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 <code>plot.default()</code> .

**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)
```

---

pool_combine	<i>Combine wkpool objects</i>
--------------	-------------------------------

---

**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	<i>Compact a pool by removing unreferenced vertices</i>
--------------	---

---

**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))
```

---

reverse_cycle	<i>Reverse a cycle's winding direction</i>
---------------	--

---

**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 <code>wk::as_wkb()</code>

**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)
```

**Arguments**

x                    A wkpool  
 tolerance          Tolerance for "near miss" detection

**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

---

*Combine many wkpool vectors into one vector*


---

**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)
```

**Arguments**

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

**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-accessors	<i>Access components of a wkpool object</i>
------------------	---

---

**Description**

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

**Usage**

```
pool_vertices(x)
```

```
pool_segments(x)
```

```
pool_feature(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`.
- `pool_feature()`: An integer vector of feature IDs, or `NULL` if no feature information is present.

**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)
```

# Index

arc\_node\_summary, 3  
arcs\_to\_wkb, 2  
arcs\_to\_wkt (arcs\_to\_wkb), 2  
as\_arcs, 4  
as\_pslg, 4

classify\_cycles, 5  
cycle\_signed\_area, 7  
cycles\_to\_wkb (cycles\_to\_wkt), 6  
cycles\_to\_wkt, 6

establish\_topology, 8

find\_arcs, 9  
find\_cycles, 9  
find\_internal\_boundaries, 10  
find\_neighbours, 11  
find\_nodes, 12  
find\_shared\_edges, 12

hole\_points, 13

merge\_coincident, 14

plot.default(), 15  
plot.wkpool, 15  
pool\_combine, 16  
pool\_combine(), 20  
pool\_compact, 16  
pool\_feature (wkpool-accessors), 20  
pool\_segments (wkpool-accessors), 20  
pool\_vertices (wkpool-accessors), 20

reverse\_cycle, 17

segments\_to\_wkb (segments\_to\_wkt), 18  
segments\_to\_wkt, 18

topology\_report, 18

vec\_c, 19

vec\_c(), 19  
vertex\_degree, 20

wk::as\_wkb(), 2, 6, 18  
wkpool-accessors, 20