# Package: hydrofabric3D (via r-universe)

December 6, 2024

**Title** hydrofabric3D

**Version** 0.1.97

**Description** Cuts terrain based cross sections for a river network.

**License** Apache License (>= 2)

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

**Imports** geos, wk, terra, dplyr, tidyr, vctrs, sf, smoothr, zoo, nhdplusTools, hydroloom, fastmap, DescTools, purrr, ggplot2, rlang, lwgeom, rmapshaper, AHGestimation, methods

**Remotes** mikejohnson51/AHGestimation

**Depends** R (>= 2.10)

**LazyData** true

**Config/Needs/website** rmarkdown

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0), distill, pals, glue, patchwork

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Config/pak/sysreqs** cmake libgdal-dev gdal-bin libgeos-dev make libicu-dev libpng-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev libnode-dev libx11-dev zlib1g-dev

**Repository** https://owp-spatial.r-universe.dev

**RemoteUrl** https://github.com/lynker-spatial/hydrofabric3D

**RemoteRef** HEAD

**RemoteSha** b4e6f21ecc0dff00408458656c084a2a3a714e1a

# Contents

add_bank_attributes . . . 3
add_braid_ids . . . 4
add_cs_area . . . 5
add_cs_bathymetry . . . 5
add_cs_id_sequence . . . 6
add_intersects_ids . . . 6
add_length_col . . . 7
add_points_per_cs . . . 7
add_point_type_counts . . . 8
add_powerlaw_bankful_width . . . 9
add_relief . . . 9
add_tmp_id . . . 10
adjust_flagged_transects . . . 11
adjust_transect_lengths . . . 11
align_banks_and_bottoms . . . 12
braided_flowlines . . . 13
classify_points . . . 13
compare_cs_validity . . . 14
cross_section_pts . . . 15
cs_arrange . . . 15
cut_cross_sections . . . 16
cut_transect . . . 17
dem_based_points_per_cs . . . 18
diff_overlaps . . . 18
drop_incomplete_cs_pts . . . 19
extend_by_percent . . . 19
extend_transects_by_cs_attributes . . . 20
extend_transects_sides . . . 21
extend_transects_to_polygons . . . 22
extract_dem_values . . . 23
find_braids . . . 24
fix_braided_transects . . . 25
flag_transects_for_change . . . 26
flowlines . . . 27
flowlines_missing_depth . . . 27
geos_extend_line . . . 28
get_bank_attributes . . . 28
get_braid_list . . . 29
get_cs_bottom_length . . . 30
get_points_per_cs . . . 30
get_point_type_counts . . . 31
get_relief . . . 32
get_start_node . . . 33
get_transects . . . 33
get_transect_extension_distances_to_polygons . . . 34
get_unique_tmp_ids . . . 35

get_validity_tally . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
invalid_flowlines . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
is_braided . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
junction_flowlines . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
linestring . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
nextgen_braided_flowlines . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
plot_cs_pts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
prep_flowlines . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
pts_to_XY . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
renumber_cs_ids . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
rm_multiflowline_intersections . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
rm_multi_intersects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
rm_self_intersections . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
select_cs_pts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
select_transects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
shorten_multi_flowline_intersecting_extended_transects . . . . . . . . . . . . . . . 43
shorten_multi_transect_intersecting_extended_transects . . . . . . . . . . . . . . . 44
transects_missing_depth . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
transects_to_cs_pts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
trim_transects_to_polygons . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
validate_classified_cs_pts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
validate_classified_cs_pts_against_transects . . . . . . . . . . . . . . . . . . . . 47
validate_cs_pts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
validate_cs_pts_against_transects . . . . . . . . . . . . . . . . . . . . . . . . . . 48
validate_transects . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
validate_transects_against_flowlines . . . . . . . . . . . . . . . . . . . . . . . . 49

**Index** **50**

---

## add_bank_attributes *Adds attributes about the banks of each cross section in a dataframe of cross section points Function adds "bottom", "left_bank", "right_bank" columns that are the Z values of the "lowest" bottom point, and the "highest" left and right bank Z values, respectively. If there are And also a "valid_banks" column is added that is TRUE if the hy_id/cs_id set of cross section point has at least 1 bottom point with at least 1 left bank point AND 1 right bank point that are above the lowest "bottom" point.*

---

### Description

Adds attributes about the banks of each cross section in a dataframe of cross section points Function adds "bottom", "left_bank", "right_bank" columns that are the Z values of the "lowest" bottom point, and the "highest" left and right bank Z values, respectively. If there are And also a "valid_banks" column is added that is TRUE if the hy_id/cs_id set of cross section point has at least 1 bottom point with at least 1 left bank point AND 1 right bank point that are above the lowest "bottom" point.

## Usage

```
add_bank_attributes(classified_pts)
```

## Arguments

| | |
|---|---|
| `classified_pts` | sf or dataframe of points with "hy_id", "cs_id", and "point_type" columns. Output of hydrofabric3D::classify_pts() |

## Value

sf or dataframe with added "bottom", "left_bank", "right_bank", and "valid_banks" columns

---

# `add_braid_ids` *Find braids and add to a dataframe/sf dataframe Adds a 'braid_id' and 'is_multibraid' columns to an sf dataframe containing a crosswalk_id and sf linestring geometires*

---

## Description

Find braids and add to a dataframe/sf dataframe Adds a 'braid_id' and 'is_multibraid' columns to an sf dataframe containing a crosswalk_id and sf linestring geometires

## Usage

```
add_braid_ids(network, crosswalk_id = NULL, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| `network` | The network object representing the river network. |
| `crosswalk_id` | unique ID column name |
| `verbose` | Logical indicating whether to display verbose messages during the braid detection process. |

## Value

dataframe or sf dataframe with added braid_id

## `add_cs_area` *Adds a cs_area column to a set of cross section points*

### Description

Adds a cs_area column to a set of cross section points

### Usage

```
add_cs_area(cs_pts, crosswalk_id = NULL)
```

### Arguments

| | |
|---|---|
| `cs_pts` | dataframe or sf dataframe of CS points with crosswalk_id, cs_id, Z, and relative distance columns |
| `crosswalk_id` | character, unique ID column name |

### Value

cs_pts dataframe with added numeric 'cs_area' column

## `add_cs_bathymetry` *Given provide inchannel widths and depths to a set of cross section points and derive estimated shapes*

### Description

Takes in a point of cross section points with added top width (TW), depth (DEPTH), and dingman_r (DINGMAN_R) columns

### Usage

```
add_cs_bathymetry(cs_pts = NULL, crosswalk_id = NULL)
```

### Arguments

| | |
|---|---|
| `cs_pts` | dataframe or sf dataframe. Default is NULL |
| `crosswalk_id` | character, ID column |

### Value

dataframe or sf dataframe with AHG estimated points injected into the input cross section points

## `add_cs_id_sequence` *Add a 1:number of cross sections 'cs_id' for each crosswalk_id by cs_measure*

### Description

Add a 1:number of cross sections 'cs_id' for each crosswalk_id by cs_measure

### Usage

```
add_cs_id_sequence(x, crosswalk_id = NULL)
```

### Arguments

| | |
|---|---|
| `x` | dataframe, sf dataframe or tibble |
| `crosswalk_id` | character, unique ID column |

### Value

dataframe, sf dataframe or tibble with an added 'cs_id' column

## `add_intersects_ids` *Add an ID column from 'y' if it intersects with 'x'*

### Description

Add an ID column from 'y' if it intersects with 'x'

### Usage

```
add_intersects_ids(x, y, id_col)
```

### Arguments

| | |
|---|---|
| `x` | sf dataframe |
| `y` | sf dataframe |
| `id_col` | character, unique ID column name in 'y' |

### Value

sf dataframe of x with id_col added if it intersects with y

## `add_length_col` *Add a length column to a sf geometry dataframe*

### Description

Add a length column to a sf geometry dataframe

### Usage

```
add_length_col(x, length_col = NULL, add_unit_to_col = FALSE)
```

### Arguments

| | |
|---|---|
| `x` | sf dataframe |
| `length_col` | character, name to use for length column. Default is NULL which will use "geom_length" as the length column name |
| `add_unit_to_col` | logical, whether to try and extract units from the geometry and append these to the column name. Default is FALSE |

### Value

sf dataframe

## `add_points_per_cs` *Add a points per cross section column to an sf dataframe of linestrings given a DEM and min points value*

### Description

This function calculates and adds a column called 'points_per_cs' to an sf dataframe representing cross-sections (linestrings) based on a provided DEM and a minimum points value per cross section.

### Usage

```
add_points_per_cs(
  transects,
  points_per_cs = NULL,
  min_pts_per_cs = 10,
  dem = default_dem
)
```

### Arguments

`transects` An sf dataframe representing cross-sections (linestrings). With a required cs_lengthm column (length of cross section in meters)

`points_per_cs` numeric, number of points per cross section. Default is NULL

`min_pts_per_cs` An optional minimum points value per cross section. If not provided,

`dem` A SpatRaster object representing the Digital Elevation Model (DEM) or a character string referencing a remote resource. the function calculates it based on the length of cross-sections and the resolution of the DEM.

### Value

An updated sf dataframe with the 'points_per_cs' column added.

## `add_point_type_counts` *Add the count of each point type as a column to a dataframe of section points*

### Description

add_point_type_counts() will add columns to the input dataframe with the counts of every point_type for each hy_id/cs_id in the input dataframe of classified cross section points (output of classify_pts())

### Usage

```
add_point_type_counts(classified_pts, crosswalk_id = NULL)
```

### Arguments

`classified_pts` dataframe or sf dataframe, cross section points with a "hy_id", and "cs_id" columns as well as a 'point_type' column containing the values: "bottom", "left_bank", "right_bank", and "channel"

`crosswalk_id` character, ID column

### Value

dataframe or sf dataframe with "<point_type>_count" columns added

## `add_powerlaw_bankful_width` *Add powerlaw_bankful_width column*

### Description

Add powerlaw_bankful_width column

### Usage

```
add_powerlaw_bankful_width(df, total_drainage_area_sqkm_col, min_bf_width)
```

### Arguments

| | |
|---|---|
| `df` | dataframe |
| `total_drainage_area_sqkm_col` | character, column with the total downstrream drainage area in square kilometers (numeric column) |
| `min_bf_width` | numeric, minimum bankful width value |

### Value

character, column with the total downstrream drainage area in square kilometers (numeric column)

## `add_relief` *Add relief attributes to a dataframe of cross sections points Given a set of cross section points (derived from hydrofabric3D::cross_section_pts() and hydrofabric3D::classify_points()) add a "has_relief" logical value to data. The "has_relief" value is indicating whether a cross section "has relief". Relief is determined by checking each set of cross section points have a left OR right bank that has a depth difference from the bottom that isgreater than or equal to a percentage of the cross section length (e.g. Assuming a 'pct_of_length_for_relief' of 0.01 (1%) of a 100m cross section would have a relief depth threshold of 1m)*

### Description

Add relief attributes to a dataframe of cross sections points Given a set of cross section points (derived from hydrofabric3D::cross_section_pts() and hydrofabric3D::classify_points()) add a "has_relief" logical value to data. The "has_relief" value is indicating whether a cross section "has relief". Relief is determined by checking each set of cross section points have a left OR right bank that has a depth difference from the bottom that isgreater than or equal to a percentage of the cross section length (e.g. Assuming a 'pct_of_length_for_relief' of 0.01 (1%) of a 100m cross section would have a relief depth threshold of 1m)

### Usage

```
add_relief(
  classified_pts,
  crosswalk_id = NULL,
  pct_of_length_for_relief = 0.01
)
```

### Arguments

| | |
|---|---|
| `classified_pts` | sf or dataframe of points with "hy_id", "cs_id", "cs_lengthm", and "point_type" columns. Output of hydrofabric3D::classify_points() |
| `crosswalk_id` | character, ID column |
| `pct_of_length_for_relief` | numeric, percent of cs_lengthm to use as the threshold depth for classifying whether a cross section has "relief". Default is 0.01 (1% of the cross sections length). |

### Value

sf or dataframe with added "has_relief" columns or a dataframe of dataframe of unique hy_id/cs_id and "has_relief"

---

## add_tmp_id *Function to add a new "tmp_id" column to a dataframe from 2 other columns*

---

### Description

Internal convenience function for creating a tmp_id column from 2 other columns in a dataframe. Default is to use hy_id and cs_id columns to create a tmp_id = <hy_id>_<cs_id>.

### Usage

```
add_tmp_id(df, x = "hy_id", y = "cs_id")
```

### Arguments

| | |
|---|---|
| `df` | dataframe with x and y as columns |
| `x` | character, column name in df to make up the first part of the added tmp_id column (tmp_id = x_y). Default is "hy_id." |
| `y` | character, column name in df to make up the second part of the added tmp_id column (tmp_id = x_y). Default is "cs_id." |

### Value

The input dataframe with the "tmp_id" column added.

## `adjust_flagged_transects` *Update a flagged set of transects by shortening them by the given left_distance and right_distance Requires 'left_distance' and 'right_distance' columns to specify how much to adjust flagged transects by*

### Description

Update a flagged set of transects by shortening them by the given left_distance and right_distance Requires 'left_distance' and 'right_distance' columns to specify how much to adjust flagged transects by

### Usage

```
adjust_flagged_transects(x, crosswalk_id = NULL, reindex_cs_ids = FALSE)
```

### Arguments

| | |
|---|---|
| `x` | sf dataframe of transects |
| `crosswalk_id` | character, unique ID column |
| `reindex_cs_ids` | logical, whether to generate a new 1-n set of cs_ids or to return the original identifiers |

### Value

sf dataframe of transects with updated geometries

## `adjust_transect_lengths` *Extend/shrink an sf linestring dataframe by a specified lengths vector*

### Description

Extend/shrink an sf linestring dataframe by a specified lengths vector

### Usage

```
adjust_transect_lengths(
  x,
  crosswalk_id = NULL,
  dir = "left",
  length_col = NULL
)
```

### Arguments

`x` linestring sf dataframe, requires an

`crosswalk_id` character, unique ID column name

`dir` direction to extend/shrink transect from, either "left" or "right". Default is "left".

`length_col` character, name of the column in "x" that has the length of the linestring (meters)

### Value

sf dataframe with extended linestring geometries

## `align_banks_and_bottoms` *Align banks and smooth bottoms of cross section points*

### Description

Ensures the bottom of each cross section is lower then or equal to that one upstream. To do this, we traverse down the network making sure this condition is met, and, in cases where it isn’t, we will lower the in channel portion of the cross section to make it true.

### Usage

```
align_banks_and_bottoms(cs_pts, crosswalk_id)
```

### Arguments

`cs_pts` dataframe or sf dataframe of classified cross section points (output of classify_points())

`crosswalk_id` character, ID column

### Value

sf dataframe of cross section points with aligned banks and smoothed bottoms

## `braided_flowlines` *Braided Flowlines*

### Description

A dataset containing flowlines representing braided river sections. These flowlines are used in hydrologic models to simulate complex river networks.

### Usage

```
braided_flowlines
```

### Format

An object of class `sf` (inherits from `tbl_df`, `tbl`, `data.frame`) with 508 rows and 39 columns.

### Source

Generated using hydrofabric3D software.

## `classify_points` *Classify Cross Section Points (version 3) with NA removal Version 2 of cross section point classifier function, uses 1st and 2nd derivative of the depths to better classify channel points*

### Description

Classify Cross Section Points (version 3) with NA removal Version 2 of cross section point classifier function, uses 1st and 2nd derivative of the depths to better classify channel points

### Usage

```
classify_points(
  cs_pts,
  crosswalk_id = NULL,
  pct_of_length_for_relief = 0.01,
  na.rm = TRUE
)
```

### Arguments

`cs_pts` CS points, output of hydrofabric3D::cross_section_pts()

`crosswalk_id` character, ID column in cs_pts

`pct_of_length_for_relief`
numeric, percent of cross section length (cs_lengthm) to use as the threshold depth for classifying whether a cross section has "relief". If a cross section has at least X% of its length in depth, then it is classified as "having relief" (i.e. has_relief = TRUE). Value must be non negative number (greater than or equal to 0). Default is 0.01 (1% of the cross sections length).

`na.rm` logical, whether to remove cross section pts with any missing Z values (Z = NA). Default is TRUE.

### Value

sf object

---

## `compare_cs_validity` *Compare valid_banks and has_relief between 2 sets of cross section points*

---

### Description

Compare valid_banks and has_relief between 2 sets of cross section points

### Usage

```
compare_cs_validity(cs_pts1, cs_pts2, crosswalk_id = NULL)
```

### Arguments

`cs_pts1` dataframe or sf dataframe of CS pts

`cs_pts2` dataframe or sf dataframe of CS pts

`crosswalk_id` character unique ID

### Value

dataframe, tibble

## `cross_section_pts` *Get Points across transects with elevation values*

### Description

Get Points across transects with elevation values

### Usage

```
cross_section_pts(
  transects = NULL,
  crosswalk_id = NULL,
  points_per_cs = NULL,
  min_pts_per_cs = 10,
  dem = default_dem
)
```

### Arguments

| | |
|---|---|
| `transects` | character, Hydrographic LINESTRING Network file path |
| `crosswalk_id` | character, ID column |
| `points_per_cs` | the desired number of points per CS. If NULL, then approximately 1 per grid cell resultion of DEM is selected. |
| `min_pts_per_cs` | Minimum number of points per cross section required. |
| `dem` | the DEM to extract data from |

### Value

sf object cross section points along the 'cs' linestring geometries

## `cs_arrange` *Rearrange transects / cross sections in order from upstream to downstream*

### Description

Rearrange transects / cross sections in order from upstream to downstream

### Usage

```
cs_arrange(x, crosswalk_id = NULL, order_by = c("cs_id", "cs_measure"))
```

### Arguments

| | |
|---|---|
| `x` | dataframe, sf dataframe or tibble |
| `crosswalk_id` | character, unique ID column |
| `order_by` | character, either "cs_id" or "cs_measure" |

### Value

dataframe, sf dataframe or tibble with an added 'cs_id' column

## `cut_cross_sections` *Generate Cross Sections Across Hydrographic Network*

### Description

Generate Cross Sections Across Hydrographic Network

### Usage

```
cut_cross_sections(
  net,
  crosswalk_id = NULL,
  cs_widths = 100,
  num = 10,
  smooth = TRUE,
  densify = 2,
  rm_self_intersect = TRUE,
  fix_braids = FALSE,
  braid_threshold = NULL,
  braid_method = "crosswalk_id",
  precision = 1,
  add = FALSE,
  verbose = TRUE
)
```

### Arguments

| | |
|---|---|
| `net` | Hydrographic LINESTRING Network |
| `crosswalk_id` | Unique Identifier in net |
| `cs_widths` | numeric, Bankfull Widths (length of cross sections for each net element) |
| `num` | numeric, Number of transects per Net element |
| `smooth` | logical, whether to smooth linestring geometries or not. Default is TRUE. |
| `densify` | numeric, how many times more points should be added to linestrings. Default is 2. |
| `rm_self_intersect` | logical, whether to remove self intersecting transect linestrings |

`fix_braids` logical, whether to fix braided transect lines or not. If TRUE, linestrings that are part of a braided network are augmented. Default is FALSE.

`braid_threshold` numeric value, value of the total length of all flowlines in a braid. Only braids with total flowline lengths less than or equal to the threshold will be considered by function (i.e. determines that maximum braid size that fix_braid_transects() should operate on). Default is NULL, which will attempt to fix all the braid transects in the data

`braid_method` The method to determine the geometries to cut. Options are "crosswalk_id", "component", or "neighbor". Default is "crosswalk_id"

`precision` int, distance in meters. Only applicable when fix_braids = TRUE. This is the number of meters to approximate final cross section linestring length. Increasing this value will decrease runtime of cross section extension algorithm. Value you must be greater than 0. Default is 1

`add` logical indicating whether to add original 'net' data to the outputted transect lines. Default is FALSE.

`verbose` logical, whether to output messages or not. Default is TRUE, and messages will be given

### Value

sf object of transect linestrings

## `cut_transect` *Generate a Perpendicular Linestring of a Given Width*

### Description

Generate a Perpendicular Linestring of a Given Width

### Usage

```
cut_transect(edge, width)
```

### Arguments

`edge` geos_geometry LINESTRING

`width` Length of Perpendicular LINESTRING

### Value

GEOS object

## dem_based_points_per_cs

*Calculate the points per cross section based off length relative to a DEM Given the length of cross sections and a DEM, approximate the appropriate number of points for each cross section length*

### Description

Calculate the points per cross section based off length relative to a DEM Given the length of cross sections and a DEM, approximate the appropriate number of points for each cross section length

### Usage

```
dem_based_points_per_cs(cs_length, dem = default_dem)
```

### Arguments

| | |
|---|---|
| `cs_length` | numeric vector, lengths of each cross section (meters) |
| `dem` | A SpatRaster object representing the Digital Elevation Model (DEM) or a character string referencing a remote resource. the function calculates it based on the length of cross-sections and the resolution of the DEM. |

### Value

numeric vector of length cs_length, with the number of points per cs_length

## diff_overlaps

*Use sf::st_difference to resolve overlaps in polygons based on intersections with other polygons*

### Description

Use sf::st_difference to resolve overlaps in polygons based on intersections with other polygons

### Usage

```
diff_overlaps(x)
```

### Arguments

| | |
|---|---|
| `x` | sf dataframe |

### Value

sf dataframe with overlaps removed

## `drop_incomplete_cs_pts` *Remove entire cross sections that have any NA Z (depth) values*

### Description

Remove entire cross sections that have any NA Z (depth) values

### Usage

```
drop_incomplete_cs_pts(cross_section_pts, crosswalk_id = NULL)
```

### Arguments

| | |
|---|---|
| `cross_section_pts` | cs points dataframe, tibble, or sf dataframe |
| `crosswalk_id` | unique ID for flowline |

### Value

cross_section_pts dataframe / tibble / sf dataframe with removed cross sections

## `extend_by_percent` *Extend an sf linestring dataframe by a percent of the lines length*

### Description

Extend an sf linestring dataframe by a percent of the lines length

### Usage

```
extend_by_percent(x, crosswalk_id = NULL, pct = 0.5, length_col = NULL)
```

### Arguments

| | |
|---|---|
| `x` | linestring sf dataframe |
| `crosswalk_id` | character, unique ID column name |
| `pct` | numeric, percent of line to extend linestring by in both directions |
| `length_col` | character, name of the output length column name. Default is NULL which will create a length column name of "geom_length". |

### Value

sf dataframe with extended linestring geometries

## extend_transects_by_cs_attributes

*Extend transects for any transects with invalid cross section attributes*

### Description

Extend transects for any transects with invalid cross section attributes

### Usage

```
extend_transects_by_cs_attributes(
  transects = NULL,
  flowlines = NULL,
  crosswalk_id = NULL,
  scale = 0.5,
  keep_lengths = FALSE,
  keep_extension_distances = FALSE,
  reindex_cs_ids = FALSE,
  verbose = TRUE
)
```

### Arguments

| | |
|---|---|
| `transects` | sf dataframe of transect LINESTRING geometries |
| `flowlines` | sf dataframe of flowline LINESTRING geometries |
| `crosswalk_id` | character |
| `scale` | numeric percent of original transect length to extend (in both directions). Default is 0.5 or 50% of transects length (i.e. 25% increase in length in both directions). |
| `keep_lengths` | logical whether to keep a record of the original transect lengths or not, default is FALSE, original lengths are not kept |
| `keep_extension_distances` | logical whether to return the extension distance (left_distance and right_distance) columns with the output dataframe. Default is FALSE, left_distance and right_distance are NOT returned with the output. |
| `reindex_cs_ids` | logical, whether to reindex the cs_ids to ensure each crosswalk_id has cs_ids of 1-number of transects. Default is FALSE, which guarantees crosswalk_id/cs_ids remain untouched as they were given in the input data. Setting this to TRUE will make sure if any cross sections were removed from a crosswalk_id, then the cs_ids are renumbered so there are no gaps between cs_ids within a crosswalk_id |
| `verbose` | logical |

### Value

dataframe or sf dataframe of extended transects

## extend_transects_sides

*Given a set of transect lines, a flowline network, extend the transect lines out given distances from the left and right Flowlines are required to ensure valid transect intersection relationship is maintained*

### Description

Given a set of transect lines, a flowline network, extend the transect lines out given distances from the left and right Flowlines are required to ensure valid transect intersection relationship is maintained

### Usage

```
extend_transects_sides(
  transects,
  flowlines,
  crosswalk_id,
  cs_id = "cs_id",
  grouping_id = "mainstem",
  direction = "any"
)
```

### Arguments

`transects` sf dataframe of linestrings, requires crosswalk_id, cs_id, grouping_id columns and numeric ’extension_distance’ column indicating the distance to extend

`flowlines` sf dataframe of linestrings

`crosswalk_id` character, column name that connects features in transects to flowlines

`cs_id` character, column name that uniquely identifies transects within a flowline

`grouping_id` character, column name in both transects and flowlines that denotes which flowlines are grouped with which transects.

`direction` character, whether to extend transects individually from left and right sides, or to strictly extend a transect if BOTH the left and right extension are valid. Valid inputs are either "any", "any_by_specific_distances", or "both".

### Value

transects sf dataframe with extended transect geometries, left and right distance columns, and flags indicating if the transect was extended in the left and/or right directions

## extend_transects_to_polygons

*Give a set of transecct linestrings and poylgons and get the minimum distance to extend each transect line (from both directions, to try and reach the edge of a "polygons") Superseces old version of function (now named extend_transects_to_polygons2())*

### Description

Give a set of transecct linestrings and poylgons and get the minimum distance to extend each transect line (from both directions, to try and reach the edge of a "polygons") Superseces old version of function (now named extend_transects_to_polygons2())

### Usage

```
extend_transects_to_polygons(
  transect_lines,
  polygons,
  flowlines,
  crosswalk_id = NULL,
  grouping_id = "mainstem",
  max_extension_distance = 3000,
  tolerance = NULL,
  keep_lengths = FALSE,
  reindex_cs_ids = TRUE,
  verbose = TRUE
)
```

### Arguments

`transect_lines` Set of Sf linestrings to extend (only if the transect lines are ENTIRELLY within a polygons)

`polygons` set of sf polygons that transect lines should be exteneded

`flowlines` set of Sf linestrings

`crosswalk_id` character, flowline ID that matches flowlines with transect lines. This crosswalk_id must appear are a column in both flowlines and transect_lines.

`grouping_id` character, name of a column in flowlines that should be used to group each transect with 1 or more flowlines. That is, when transects are checked to make sure they don't intersect other transects or other flowlines, this group ID will distinguise which flowlines a transect should be checked against. The intersect_group_id must appear as a column in both flowlines and transect_lines dataframes

`max_extension_distance`
numeric, maximum distance (meters) to extend a transect line in either direction to try and intersect one of the "polygons". Default is 3000m

`tolerance` A minimum distance to use for simplification on polygons. Use a higher value for more simplification on the polygons. Default is NULL which will apply no simplification to polygons.

`keep_lengths` logical whether to keep a record of the original transect lengths or not, default is FALSE, original lengths are not kept

`reindex_cs_ids` logical, whether to reindex the cs_ids to ensure each crosswalk_id has cs_ids of 1-number of transects. Default is TRUE, which makes sure if any cross sections were removed from a crosswalk_id, then the cs_ids are renumbered so there are no gaps between cs_ids within a crosswalk_id. Setting this to FALSE will make sure crosswalk_id/cs_ids remain untouched as they were given in the input data.

`verbose` logical, whether to output messages or not. Default is TRUE, and messages will output

### Value

sf linestring, with extended transect lines

## `extract_dem_values` *Given a set of linestrings, extract DEM values at points along the linestring*

### Description

Given a set of linestrings, extract DEM values at points along the linestring

### Usage

```
extract_dem_values(transects, crosswalk_id = NULL, dem = NULL)
```

### Arguments

`transects` cross section sf object

`crosswalk_id` character, column name of unique flowline / transect ID

`dem` SpatRaster DEM or character pointing to remote DEM resource

### Value

sf dataframe with Z values extracted from DEM

## find_braids — *Find braided sections of a network and return the unique crosswalk_ids for each idenfied braid*

### Description

Find and uniquely identify braids in a network of flowlines, given a dataframe containing comid, fromnode, tonode and divergence as columns. 'find_braids()" identifies braids as cycles in the graph representation of the river network.

### Usage

```
find_braids(network, crosswalk_id = NULL, nested = TRUE, verbose = FALSE)
```

### Arguments

| | |
|---|---|
| `network` | The network object representing the river network. |
| `crosswalk_id` | unique ID column name |
| `nested` | Logical indicating whether the output dataframe should be nested, with each COMID having a list of all the braids it is a part of. If TRUE (Default), the braid_id column may contain multiple braid IDs for a given COMID. If FALSE, there may be duplicate COMIDs as a single COMID could be a part of multiple braids (braid_id) |
| `verbose` | Logical indicating whether to display verbose messages during the braid detection process. |

### Value

dataframe or sf dataframe with added braid_id

### Examples

```
 ## Not run:
net <- nhdplusTools::navigate_network(
 start       = 101,
 mode        = "UT",
 distance_km = 100
 )

# drop most of the columns in the network dataset
net <- dplyr::select(net, comid, divergence, totdasqkm, fromnode, tonode, terminalpa)

# get a dataframe of COMIDs and braid IDs
braids <- find_braids(network = net, crosswalk_id = "comid")


# returns original data with each braid_id represented
# by its individual COMIDs (may contain duplicate COMIDs)
```

```
nested_braids = find_braids(network   = net,
                       crosswalk_id = "comid",
                       nested    = FALSE
                       )

## End(Not run)
```

## fix_braided_transects *Fix transects found on braided river sections (latest)*

### Description

Fix transects found on braided river sections (latest)

### Usage

```
fix_braided_transects(
  network,
  transect_lines,
  crosswalk_id = NULL,
  braid_threshold = NULL,
  method = "crosswalk_id",
  precision = 1,
  rm_intersects = TRUE
)
```

### Arguments

`network` sf dataframe of hydrologic network, linestrings

`transect_lines` sf linestring dataframe, containing cross sections of flowlines in 'network' the output of "cut_cross_sections()" function

`crosswalk_id` character, unique ID column

`braid_threshold`
numeric value, value of the total length of all flowlines in a braid. Only braids with total flowline lengths less than or equal to the threshold will be considered by function (i.e. determines that maximum braid size that fix_braid_transects() should operate on). Default is NULL, which will attempt to fix all the braid transects in the data

`method` The method to determine the geometries to cut. Options are "crosswalk_id", "component", or "neighbor". Default is "crosswalk_id"

`precision` int, distance in meters to approximate final cross section linestring length. Value you must be greater than 0. Default is 1

`rm_intersects` logical, whether to remove transect linestrings that intersect with other parts of the network ('network'). Default is TRUE which will remove intersecting linestrings.

## Value

sf object of transect linestrings

`flag_transects_for_change` *Add a flagged and extension distance columns to set of transects with CS attributes based on new cross section points data*

## Description

Add a flagged and extension distance columns to set of transects with CS attributes based on new cross section points data

## Usage

```
flag_transects_for_change(
  x,
  crosswalk_id = NULL,
  points_per_cs = NULL,
  min_pts_per_cs = 10,
  dem = default_dem,
  pct_of_length_for_relief = 0.01,
  na.rm = TRUE
)
```

## Arguments

| | |
|---|---|
| `x` | sf dataframe of transects |
| `crosswalk_id` | character, unique ID column |
| `points_per_cs` | numeric |
| `min_pts_per_cs` | numeric |
| `dem` | character |
| `pct_of_length_for_relief` | numeric |
| `na.rm` | logical, whether to remove NAs from the given cross section points and any NA comparison points pulled from the dem. Default is TRUE |

## Value

sf dataframe of transects with updated geometries

## `flowlines` *Flowlines*

### Description

A dataset of primary flowlines for hydrologic and hydraulic modeling.

### Usage

```
flowlines
```

### Format

An object of class `sf` (inherits from `tbl_df`, `tbl`, `data.frame`) with 10 rows and 5 columns.

### Source

NOAA Office of Water Prediction.

## `flowlines_missing_depth` *Flowlines Missing Depth*

### Description

A dataset of flowlines missing depth information, which may require further analysis or imputation.

### Usage

```
flowlines_missing_depth
```

### Format

An object of class `sf` (inherits from `tbl_df`, `tbl`, `data.frame`) with 1 rows and 5 columns.

### Source

Derived from `flowlines`.

## geos_extend_line *Extend a geos_geometry linestring from, one or both ends, by a given distance (meters)*

### Description

Extend a geos_geometry linestring from, one or both ends, by a given distance (meters)

### Usage

```
geos_extend_line(line, distance, dir = "both", with_crs = TRUE)
```

### Arguments

| | |
|---|---|
| line | sf linestring or geos_geometry linestring to extend |
| distance | numeric value in meters or a vector of length 2 if 'end = "both"' where |
| dir | character, determines whether to extend the linestring from the 'tail', 'head' or 'both' ends |
| with_crs | logical, whether a CRS should be prescribed to extended output geos_geometry linestring |

### Value

geos_geometry linestring extended by 'distance' from either the 'head', 'tail' or 'both' ends of the original linestring

## get_bank_attributes *Get attributes about the banks of each cross section in a dataframe of cross section points Given a set of cross section points with point_type column, return a dataframe of the unique hy_id/cs_ids with the following calculated columns: "bottom", "left_bank", "right_bank" columns which are the Z values of the "lowest" bottom point, and the "highest" left and right bank Z values, respectively. And a "valid_banks" column indicating whether the hy_id/cs_id set of cross section point has at least a signle bottom point with at least 1 left bank point AND 1 right bank point that are above the lowest "bottom" point.*

### Description

Get attributes about the banks of each cross section in a dataframe of cross section points Given a set of cross section points with point_type column, return a dataframe of the unique hy_id/cs_ids with the following calculated columns: "bottom", "left_bank", "right_bank" columns which are the Z values of the "lowest" bottom point, and the "highest" left and right bank Z values, respectively. And a "valid_banks" column indicating whether the hy_id/cs_id set of cross section point has at least a signle bottom point with at least 1 left bank point AND 1 right bank point that are above the lowest "bottom" point.

### Usage

```
get_bank_attributes(classified_pts, crosswalk_id = NULL)
```

### Arguments

| | |
|---|---|
| `classified_pts` | sf or dataframe of points with "hy_id", "cs_id", and "point_type" columns. Output of hydrofabric3D::classify_pts() |
| `crosswalk_id` | character, ID column |

### Value

dataframe with each row being a unique hy_id/cs_id with "bottom", "left_bank", "right_bank", and "valid_banks" values for each hy_id/cs_id.

## `get_braid_list` *Create a list of braid IDs containing crosswalk_ids in each braid*

### Description

Find and uniquely identify braids in a network of flowlines, given an sf dataframe containing crosswalk_id and sf linestring geometries, 'find_braids()" identifies braids as cycles in the graph representation of the river network.

### Usage

```
get_braid_list(network, crosswalk_id = NULL, verbose = FALSE)
```

### Arguments

| | |
|---|---|
| `network` | The network object representing the river network. |
| `crosswalk_id` | unique ID column name |
| `verbose` | Logical indicating whether to display verbose messages during the braid detection process. |

### Value

list of braid IDs and COMIDs within each braid

### Examples

```
 ## Not run:
net <- nhdplusTools::navigate_network(
 start       = 101,
 mode        = "UT",
 distance_km = 100
 )
```

```
net <- dplyr::select(net, comid, divergence, totdasqkm, fromnode, tonode, terminalpa)

# get a dataframe of COMIDs and braid IDs
braids <- get_braid_list(network = net, crosswalk_id = "comid")

## End(Not run)
```

## get_cs_bottom_length *Calculate the length between the leftmost and rightmost bottom point in each cross section*

### Description

Calculate the length between the leftmost and rightmost bottom point in each cross section

### Usage

```
get_cs_bottom_length(cs_pts, crosswalk_id = NULL)
```

### Arguments

| | |
|---|---|
| cs_pts | dataframe, or sf dataframe of cross section points |
| crosswalk_id | character, ID column |

### Value

summarized dataframe of input cs_pts dataframe with a bottom_length value for each hy_id/cs_id

## get_points_per_cs *Calculate the points per cross section based off length*

### Description

Calculate the points per cross section based off length

### Usage

```
get_points_per_cs(
  cs_length,
  points_per_cs = NULL,
  min_pts_per_cs = 10,
  dem = default_dem
)
```

### Arguments

`cs_length` numeric vector, lengths of each cross section (meters)

`points_per_cs` numeric, number of points per cross section. Default is NULL

`min_pts_per_cs` An optional minimum points value per cross section. If not provided,

`dem` A SpatRaster object representing the Digital Elevation Model (DEM) or a character string referencing a remote resource. the function calculates it based on the length of cross-sections and the resolution of the DEM.

### Value

numeric vector, indicating the number of points per cross section for each cs_length

## `get_point_type_counts` *Get the count of each point type in a set of cross section points*

### Description

get_point_type_counts() will create a dataframe providing the counts of every point_type for each hy_id/cs_id in a set of classified cross section points (output of classify_pts())

### Usage

```
get_point_type_counts(classified_pts, crosswalk_id = NULL)
```

### Arguments

`classified_pts` dataframe or sf dataframe, cross section points with a "hy_id", and "cs_id" columns as well as a ’point_type’ column containing the values: "bottom", "left_bank", "right_bank", and "channel"

`crosswalk_id` character, ID column

### Value

dataframe or sf dataframe with hy_id, cs_id, and <point_type>_count columns for each point_type

## `get_relief` *Get relief attributes from a dataframe of cross sections points Generate a dataframe from a set of classified cross section points indicating whether a cross section "has relief". Relief is determined by checking each set of cross section points have a left OR right bank that has a depth difference from the bottom that is greater than or equal to a percentage of the cross section length (e.g. Assuming a 'pct_of_length_for_relief' of 0.01 (1%) of a 100m cross section would have a relief depth threshold of 1m)*

### Description

Get relief attributes from a dataframe of cross sections points Generate a dataframe from a set of classified cross section points indicating whether a cross section "has relief". Relief is determined by checking each set of cross section points have a left OR right bank that has a depth difference from the bottom that is greater than or equal to a percentage of the cross section length (e.g. Assuming a 'pct_of_length_for_relief' of 0.01 (1%) of a 100m cross section would have a relief depth threshold of 1m)

### Usage

```
get_relief(
  classified_pts,
  crosswalk_id = NULL,
  pct_of_length_for_relief = 0.01,
  detailed = FALSE
)
```

### Arguments

`classified_pts` sf or dataframe of points with "hy_id", "cs_id", "cs_lengthm", and "point_type" columns. Output of hydrofabric3D::classify_pts()

`crosswalk_id` character, ID column

`pct_of_length_for_relief` numeric, percent of cs_lengthm to use as the threshold depth for classifying whether a cross section has "relief". Default is 0.01 (1% of the cross sections length).

`detailed` logical, whether to return only a the "has_relief" column or include all derived relief based columns such as "max_relief" and the "pct_of_length_for_relief" used. Default is FALSE and returns a dataframe with only "hy_id", "cs_id", and "has_relief".

### Value

dataframe with each row being a unique hy_id/cs_id with a "has_relief" value for each hy_id/cs_id. If detailed = TRUE, then the output dataframe will include the following additional columns: "cs_lengthm", "max_relief", "pct_of_length_for_relief".

## `get_start_node` *Get a valid starting node from a graph*

### Description

Get a valid starting node from a graph

### Usage

```
get_start_node(graph, start = NULL)
```

### Arguments

| | |
|---|---|
| `graph` | dataframe, sf dataframe, with fromnode and tonode columns |
| `start` | character, node in 'fromnode' column of graph |

### Value

character, node

## `get_transects` *Generate Multiple cross section along a linestring*

### Description

Generate Multiple cross section along a linestring

### Usage

```
get_transects(line, bf_width, n)
```

### Arguments

| | |
|---|---|
| `line` | sf linestring or geos_geometry, original line element |
| `bf_width` | Bankfull Width (length of cross section) |
| `n` | number of cross sections |

### Value

sf dataframe with 'n' evenly spaced transect lines with cs_measures for each cross section geometry

## get_transect_extension_distances_to_polygons

*Get the left and right extension distances for a set of transects out to a set of polygons*

### Description

Get the left and right extension distances for a set of transects out to a set of polygons

### Usage

```
get_transect_extension_distances_to_polygons(
  transects,
  polygons,
  crosswalk_id,
  max_extension_distance,
  tolerance = NULL,
  verbose = TRUE
)
```

### Arguments

| | |
|---|---|
| transects | sf linestring dataframe |
| polygons | sf polygon dataframe |
| crosswalk_id | character |
| max_extension_distance | numeric |
| tolerance | A minimum distance to use for simplification on polygons. Use a higher value for more simplification on the polygons. Default is NULL which will apply no simplification to polygons. |
| verbose | logical, whether to output messages or not. Default is TRUE, and messages will output |

### Value

data.frame or tibble

## `get_unique_tmp_ids` *Get a list of unique tmp_ids in a dataframe*

### Description

Dataframe can have "tmp_id" column already or the columns can be specified with 'x' and 'y' arguments

### Usage

```
get_unique_tmp_ids(df, x = "hy_id", y = "cs_id")
```

### Arguments

`df` dataframe with x and y as columns, with an optional "tmp_id" column, otherwise a tmp_id will be created from x_y

`x` The name of the column in df to make up the first part of the added tmp_id column (tmp_id = x_y). Default is hy_id.

`y` The name of the column in df to make up the second part of the added tmp_id column (tmp_id = x_y). Default is cs_id.

### Value

character vector of unique "tmp_id" values in the given dataframe

## `get_validity_tally` *Get a total count of the validity attributes*

### Description

Get a total count of the validity attributes

### Usage

```
get_validity_tally(x, crosswalk_id = NULL)
```

### Arguments

`x` dataframe or sf dataframe with crosswalk_id, has_relief, and valid_banks columns

`crosswalk_id` character unique ID column

### Value

dataframe or tibble

## `invalid_flowlines` *Invalid Flowlines*

### Description

A dataset of flowlines identified as invalid due to self-intersections or other topological errors.

### Usage

```
invalid_flowlines
```

### Format

An object of class `sf` (inherits from `tbl_df`, `tbl`, `data.frame`) with 2 rows and 5 columns.

### Source

Processed using `rm_self_intersections`.

## `is_braided` *Detect whether a braid exists in a hydrologic network Check if if a hydrologic network dataset contains any braids. If multiple discontinuous networks are within the 'network' data. The function will try to infer the distinct networks and then check for braids in each component (using find_connected_components()).*

### Description

Detect whether a braid exists in a hydrologic network Check if if a hydrologic network dataset contains any braids. If multiple discontinuous networks are within the 'network' data. The function will try to infer the distinct networks and then check for braids in each component (using find_connected_components()).

### Usage

```
is_braided(network, crosswalk_id = NULL, recycle = FALSE, verbose = FALSE)
```

### Arguments

| | |
|---|---|
| `network` | sf data.frame of linestrings with a unique <crosswalk_id> attribute. |
| `crosswalk_id` | unique ID column name |
| `recycle` | logical, whether the return logical vector should be recycled to the length of the number of unique networks (disconnected networks/outlets/terminalpa). If FALSE (default), the function returns TRUE if ANY of the networks contain a braid. Otherwise, if TRUE, the function attempts to distinguish the different/separate network components and returns a logical vector the length of the number of connected components in the network. |
| `verbose` | logical print status updates, if TRUE, messages will print. Default is FALSE. |

## Value

logical, If TRUE, at least one braid was detected in network, FALSE if no braids were found. If multiple components are found OR a terminal_id column is given, each unique network is checked for braiding (recycles to length of unique "terminal_id")

# `junction_flowlines` *Junction Flowlines*

## Description

A dataset of flowlines representing junctions in the river network, used for hydrodynamic connectivity analysis.

## Usage

```
junction_flowlines
```

## Format

An object of class `sf` (inherits from `tbl_df`, `tbl`, `data.frame`) with 5 rows and 5 columns.

## Source

Derived from `flowlines`.

# `linestring` *Flowlines linestring*

## Description

A dataset containing flowlines linestrings .

## Usage

```
linestring
```

```
linestring
```

## Format

An object of class `sf` (inherits from `tbl_df`, `tbl`, `data.frame`) with 325 rows and 5 columns.

An object of class `sf` (inherits from `tbl_df`, `tbl`, `data.frame`) with 325 rows and 5 columns.

## Source

Generated using hydrofabric3D software.

## nextgen_braided_flowlines *NextGen Braided Flowlines*

### Description

A dataset of braided flowlines compatible with the NextGen hydrologic prediction system.

### Usage

```
nextgen_braided_flowlines
```

### Format

An object of class `sf` (inherits from `tbl_df`, `tbl`, `data.frame`) with 183 rows and 11 columns.

### Source

Created using hydrofabric3D.

## plot_cs_pts *Plots an X-Y scatter plot of cross section points*

### Description

Plots an X-Y scatter plot of cross section points

### Usage

```
plot_cs_pts(
  cs_pts,
  crosswalk_id = NULL,
  x = "pt_id",
  y = "Z",
  color = NULL,
  size = 1,
  grid = FALSE,
  scales = "free_y"
)
```

### Arguments

`cs_pts` data.frame of cross section points with columns hy_id, cs_id and columns for X and Y axises (i.e. "pt_id", "Z")

`crosswalk_id` unique ID column name

`x` character name of column in cs_pts to use for X axis

`y` character name of column in cs_pts to use for Y axis

`color` character name of column in cs_pts to color points on plot

`size` numeric, size of the cs points, default is 1

`grid` logical, if TRUE then use facet_grid, otherwise use facet_wrap. Default is FALSE (uses facet_wrap)

`scales` Should scales be fixed ("fixed", the default), free ("free"), or free in one dimension ("free_x", "free_y")?

### Value

ggplot2 object

## `prep_flowlines` *Prepare flowlines have a more dense and/or smoother surface for cutting transects*

### Description

Prepare flowlines have a more dense and/or smoother surface for cutting transects

### Usage

```
prep_flowlines(flowlines, densify = NULL, smooth = FALSE, verbose = TRUE)
```

### Arguments

`flowlines` sf dataframe of flowline linestrings

`densify` numeric, if NULL, no densification happens. Default is NULL

`smooth` logical, whether to smooth linestrings

`verbose` logical

### Value

sf dataframe

## `pts_to_XY` *Convert an sf dataframe with a point geometry column to non spatial with XY columns*

### Description

Convert an sf dataframe with a point geometry column to non spatial with XY columns

### Usage

```
pts_to_XY(pts)
```

### Arguments

| | |
|---|---|
| `pts` | sf dataframe of points |

### Value

data.frame or tibble with added X and Y columns

## `renumber_cs_ids` *Fix IDs in a dataframe*

### Description

This function renumbers cross section IDs in a dataframe to ensure each crosswalk_id has cross sections numbered from 1 to the total number of cross sections on the crosswalk_id.

### Usage

```
renumber_cs_ids(df, crosswalk_id = NULL)
```

### Arguments

| | |
|---|---|
| `df` | A dataframe containing crosswalk_id and cs_id columns. |
| `crosswalk_id` | crosswalk_id character, name of primary ID column |

### Value

The input dataframe with renumbered cs_id values.

## rm_multiflowline_intersections

*Remove transect lines that intersect with more than one flowline*

### Description

Remove transect lines that intersect with more than one flowline

### Usage

```
rm_multiflowline_intersections(transects, flowlines)
```

### Arguments

| | |
|---|---|
| transects | sf linestring dataframe of transect lines |
| flowlines | sf linestring dataframe of flowlines |

### Value

sf linestring dataframe

## rm_multi_intersects

*Selectively removes intersecting transect lines Attempts to remove transects intersecting other transects by first removing transects that interesect the most other transects, then re checking intersection condition, and doing this until there are no multi intersections this gives the benefit of removing a transect line that intersects many other transects, potentially leaving those other transects with no extraneous intersections ONCE the MULTI intersecting transect is removed*

### Description

Selectively removes intersecting transect lines Attempts to remove transects intersecting other transects by first removing transects that interesect the most other transects, then re checking intersection condition, and doing this until there are no multi intersections this gives the benefit of removing a transect line that intersects many other transects, potentially leaving those other transects with no extraneous intersections ONCE the MULTI intersecting transect is removed

### Usage

```
rm_multi_intersects(x)
```

### Arguments

| | |
|---|---|
| x | sf dataframe of linestrings |

### Value

sf dataframe

## rm_self_intersections *Remove Self-Intersections*

### Description

A dataset of flowlines processed to remove self-intersections.

### Usage

```
rm_self_intersections
```

### Format

An object of class `function` of length 1.

### Source

Generated using `rm_self_intersections` function.

## select_cs_pts *Select standard cross section point columns Internal helper function for selecting cross section point columns aligning with standard data model for cross section points*

### Description

Select standard cross section point columns Internal helper function for selecting cross section point columns aligning with standard data model for cross section points

### Usage

```
select_cs_pts(cs_pts, crosswalk_id = NULL)
```

### Arguments

| | |
|---|---|
| `cs_pts` | dataframe, tibble, or sf dataframe |
| `crosswalk_id` | character, unique ID column |

### Value

dataframe, tibble, or sf dataframe with only relavent cross section point columns

## `select_transects` *Select standard transect columns Internal helper function for selecting transect columns aligning with standard data model for transect*

### Description

Select standard transect columns Internal helper function for selecting transect columns aligning with standard data model for transect

### Usage

```
select_transects(transects, crosswalk_id = NULL)
```

### Arguments

| | |
|---|---|
| `transects` | dataframe, tibble, or sf dataframe |
| `crosswalk_id` | character, unique ID column |

### Value

dataframe, tibble, or sf dataframe with only relavent transects columns

## `shorten_multi_flowline_intersecting_extended_transects` *Takes any transects that was extended and with multiple flowline intersections, and shortens them by the distance specified in the "extension_distance" column*

### Description

Takes any transects that was extended and with multiple flowline intersections, and shortens them by the distance specified in the "extension_distance" column

### Usage

```
shorten_multi_flowline_intersecting_extended_transects(
  x,
  flowlines,
  crosswalk_id = NULL
)
```

### Arguments

| | |
|---|---|
| `x` | sf dataframe of transects, requires a crosswalk_id, cs_id, cs_lengthm, extension_distance, and geometry column |
| `flowlines` | sf dataframe of flowline LINESTRINGS to compare to |
| `crosswalk_id` | character, unique ID column |

### Value

sf dataframe of transects with any transects that intersect multiple other transects being shortened by -extension_distance

---

## shorten_multi_transect_intersecting_extended_transects

*Takes any transects with multiple intersections that was extended, and shortens them by the distance specified in the "extension_distance" column*

---

### Description

Takes any transects with multiple intersections that was extended, and shortens them by the distance specified in the "extension_distance" column

### Usage

```
shorten_multi_transect_intersecting_extended_transects(x, crosswalk_id = NULL)
```

### Arguments

| | |
|---|---|
| `x` | sf dataframe of transects, requires a crosswalk_id, cs_id, cs_lengthm, extension_distance, and geometry column |
| `crosswalk_id` | character, unique ID column |

### Value

sf dataframe of transects with any transects that intersect multiple other transects being shortened by -extension_distance

---

## transects_missing_depth

*Transects Missing Depth*

---

### Description

A dataset of transects where depth information is unavailable, potentially impacting hydraulic model accuracy.

### Usage

```
transects_missing_depth
```

### Format

An object of class `sf` (inherits from `tbl_df`, `tbl`, `data.frame`) with 5 rows and 8 columns.

### Source

Derived from the transect processing pipeline.

## transects_to_cs_pts *Convert SF linestring transect lines into SF points with*

### Description

Convert SF linestring transect lines into SF points with

### Usage

```
transects_to_cs_pts(transects, points_per_cs)
```

### Arguments

| | |
|---|---|
| `transects` | sf linestring |
| `points_per_cs` | numeric vector of length ’transects’, indicating the number of points to get per transect |

### Value

sf point dataframe

## trim_transects_to_polygons *Trim a set of transects to the bounds of polygons*

### Description

Trim a set of transects to the bounds of polygons

### Usage

```
trim_transects_to_polygons(
  transect_lines,
  flowlines,
  polygons,
  crosswalk_id = NULL,
  dissolve = FALSE
)
```

### Arguments

| | |
|---|---|
| transect_lines | sf dataframe |
| flowlines | sf dataframe |
| polygons | sf dataframe |
| crosswalk_id | character unique ID |
| dissolve | logical, whether to dissolve polygon internal boundaries or not. Default is FALSE. |

### Value

sf dataframe

## validate_classified_cs_pts

*Validate Classified Cross Sections Points Ensure all cross section points are valid. This validates the points in the same manner as validate_cs_pts() but also checks that classification columns ('class', 'point_type', 'valid_banks', 'has_relief') exist.*

### Description

Validate Classified Cross Sections Points Ensure all cross section points are valid. This validates the points in the same manner as validate_cs_pts() but also checks that classification columns ('class', 'point_type', 'valid_banks', 'has_relief') exist.

### Usage

```
validate_classified_cs_pts(cs_pts, crosswalk_id = NULL)
```

### Arguments

| | |
|---|---|
| cs_pts | sf object, cross section points |
| crosswalk_id | character, column name of the crosswalk id |

### Value

logical, TRUE if cs_pts meet all required criteria, FALSE otherwise

## validate_classified_cs_pts_against_transects

*Validate Classified Cross Section Points Against Transects Ensure all cross section points are valid relative to a set of transects. This validates the points in the same manner as validate_cs_pts_against_transects() but also checks that classification columns ('class', 'point_type', 'valid_banks', 'has_relief') exist.*

### Description

Validate Classified Cross Section Points Against Transects Ensure all cross section points are valid relative to a set of transects. This validates the points in the same manner as validate_cs_pts_against_transects() but also checks that classification columns ('class', 'point_type', 'valid_banks', 'has_relief') exist.

### Usage

```
validate_classified_cs_pts_against_transects(
  cs_pts,
  transects,
  crosswalk_id = NULL
)
```

### Arguments

| | |
|---|---|
| `cs_pts` | sf object, cross section points |
| `transects` | sf object, transects |
| `crosswalk_id` | character, column name of the crosswalk id |

### Value

logical, TRUE if all validations pass, FALSE otherwise

## validate_cs_pts

*Validate Cross Sections Points Ensure all cross section points are valid*

### Description

Validate Cross Sections Points Ensure all cross section points are valid

### Usage

```
validate_cs_pts(cs_pts, crosswalk_id = NULL)
```

### Arguments

| | |
|---|---|
| `cs_pts` | sf object, cross section points |
| `crosswalk_id` | character, column name of the crosswalk id |

### Value

logical, TRUE if cs_pts meet all required criteria, FALSE otherwise

## `validate_cs_pts_against_transects` *Validate Cross Section Points Against Transects Ensure all cross section points are valid relative to a set of transects*

### Description

Validate Cross Section Points Against Transects Ensure all cross section points are valid relative to a set of transects

### Usage

```
validate_cs_pts_against_transects(cs_pts, transects, crosswalk_id = NULL)
```

### Arguments

| | |
|---|---|
| `cs_pts` | sf object, cross section points |
| `transects` | sf object, transects |
| `crosswalk_id` | character, column name of the crosswalk id |

### Value

logical, TRUE if all validations pass, FALSE otherwise

## `validate_transects` *Validate Transects*

### Description

Validate Transects

### Usage

```
validate_transects(transects, crosswalk_id = NULL)
```

## Arguments

| | |
|---|---|
| `transects` | sf object, transects |
| `crosswalk_id` | character, column name of the crosswalk id |

## Value

logical, TRUE if all validations pass, FALSE otherwise

---

# `validate_transects_against_flowlines` *Validate Transects Against Flowlines Ensure all transects are valid relative to a set of flowlines*

---

## Description

Validate Transects Against Flowlines Ensure all transects are valid relative to a set of flowlines

## Usage

```
validate_transects_against_flowlines(transects, flowlines, crosswalk_id = NULL)
```

## Arguments

| | |
|---|---|
| `transects` | sf object, transects |
| `flowlines` | sf object, flowlines |
| `crosswalk_id` | character, column name of the crosswalk id |

## Value

logical, TRUE if all validations pass, FALSE otherwise

# Index

∗ **datasets**
braided_flowlines, 13
flowlines, 27
flowlines_missing_depth, 27
invalid_flowlines, 36
junction_flowlines, 37
linestring, 37
nextgen_braided_flowlines, 38
rm_self_intersections, 42
transects_missing_depth, 44

add_bank_attributes, 3
add_braid_ids, 4
add_cs_area, 5
add_cs_bathymetry, 5
add_cs_id_sequence, 6
add_intersects_ids, 6
add_length_col, 7
add_point_type_counts, 8
add_points_per_cs, 7
add_powerlaw_bankful_width, 9
add_relief, 9
add_tmp_id, 10
adjust_flagged_transects, 11
adjust_transect_lengths, 11
align_banks_and_bottoms, 12

braided_flowlines, 13

classify_points, 13
compare_cs_validity, 14
cross_section_pts, 15
cs_arrange, 15
cut_cross_sections, 16
cut_transect, 17

dem_based_points_per_cs, 18
diff_overlaps, 18
drop_incomplete_cs_pts, 19

extend_by_percent, 19
extend_transects_by_cs_attributes, 20
extend_transects_sides, 21
extend_transects_to_polygons, 22
extract_dem_values, 23

find_braids, 24
fix_braided_transects, 25
flag_transects_for_change, 26
flowlines, 27
flowlines_missing_depth, 27

geos_extend_line, 28
get_bank_attributes, 28
get_braid_list, 29
get_cs_bottom_length, 30
get_point_type_counts, 31
get_points_per_cs, 30
get_relief, 32
get_start_node, 33
get_transect_extension_distances_to_polygons, 34
get_transects, 33
get_unique_tmp_ids, 35
get_validity_tally, 35

invalid_flowlines, 36
is_braided, 36

junction_flowlines, 37

linestring, 37

nextgen_braided_flowlines, 38

plot_cs_pts, 38
prep_flowlines, 39
pts_to_XY, 40

renumber_cs_ids, 40
rm_multi_intersects, 41
rm_multiflowline_intersections, 41

rm_self_intersections, 42

select_cs_pts, 42
select_transects, 43
shorten_multi_flowline_intersecting_extended_transects, 43
shorten_multi_transect_intersecting_extended_transects, 44

transects_missing_depth, 44
transects_to_cs_pts, 45
trim_transects_to_polygons, 45

validate_classified_cs_pts, 46
validate_classified_cs_pts_against_transects, 47
validate_cs_pts, 47
validate_cs_pts_against_transects, 48
validate_transects, 48
validate_transects_against_flowlines, 49