Package: terrainr (via r-universe)

October 13, 2024

Type Package

Title Landscape Visualizations in R and 'Unity'

Version 0.7.5.9000

Description Functions for the retrieval, manipulation, and visualization of 'geospatial' data, with an aim towards producing '3D' landscape visualizations in the 'Unity' '3D' rendering engine. Functions are also provided for retrieving elevation data and base map tiles from the 'USGS' National Map <https://apps.nationalmap.gov/services/>.

License MIT + file LICENSE

URL https://docs.ropensci.org/terrainr/, https://github.com/ropensci/terrainr

BugReports https://github.com/ropensci/terrainr/issues

Imports base64enc, ggplot2 (>= 3.4.0), glue, grDevices, httr, magick (>= 2.5.0), methods, png, rlang, sf (>= 1.0-5), terra, unifir, units

Suggests brio, covr, jpeg, knitr, progress, progressr, rmarkdown, testthat, tiff

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Depends R (>= 2.10)

Repository https://cafri-labs.r-universe.dev

RemoteUrl https://github.com/ropensci/terrainr

RemoteRef HEAD

RemoteSha aa3ea674510ded4864aa910f93b440b7728e5b5c

Contents

`addbuff` *Add a uniform buffer around a bounding box for geographic coordinates*

Description

add_bbox_buffer calculates the great circle distance both corners of your bounding box are from the centroid and extends those by a set distance. Due to using Haversine/great circle distance, latitude/longitude calculations will not be exact.

set_bbox_side_length is a thin wrapper around add_bbox_buffer which sets all sides of the bounding box to (approximately) a specified length.

Both of these functions are intended to be used with geographic coordinate systems (data using longitude and latitude for position). For projected coordinate systems, a more sane approach is to use sf::st_buffer to add a buffer, or combine sf::st_centroid with the buffer to set a specific side length.

Usage

```
add_bbox_buffer(data, distance, distance_unit = "meters", error_crs = NULL)

## S3 method for class 'sf'
add_bbox_buffer(data, distance, distance_unit = "meters", error_crs = NULL)

## S3 method for class 'Raster'
add_bbox_buffer(data, distance, distance_unit = "meters", error_crs = NULL)

## S3 method for class 'SpatRaster'
add_bbox_buffer(data, distance, distance_unit = "meters", error_crs = NULL)

set_bbox_side_length(
  data,
  distance,
```

```
  distance_unit = "meters",
  error_crs = NULL
)

## S3 method for class 'sf'
set_bbox_side_length(
  data,
  distance,
  distance_unit = "meters",
  error_crs = NULL
)

## S3 method for class 'Raster'
set_bbox_side_length(
  data,
  distance,
  distance_unit = "meters",
  error_crs = NULL
)

## S3 method for class 'SpatRaster'
set_bbox_side_length(
  data,
  distance,
  distance_unit = "meters",
  error_crs = NULL
)
```

Arguments

`data`	The original data to add a buffer around. Must be either an `sf` or `SpatRaster` object.
`distance`	The distance to add or to set side lengths equal to.
`distance_unit`	The units of the distance to add to the buffer, passed to units::as_units.
`error_crs`	Logical: Should this function error if `data` has no CRS? If `TRUE`, function errors; if `FALSE`, function quietly assumes EPSG:4326. If `NULL`, the default, function assumes EPSG:4326 with a warning.

Value

An `sfc` object (from sf::st_as_sfc).

See Also

Other utilities: `calc_haversine_distance()`, `deg_to_rad()`, `get_centroid()`, `rad_to_deg()`

Examples

```
df <- data.frame(
```

```
  lat = c(44.04905, 44.17609),
  lng = c(-74.01188, -73.83493)
)

df_sf <- sf::st_as_sf(df, coords = c("lng", "lat"))
df_sf <- sf::st_set_crs(df_sf, 4326)

add_bbox_buffer(df_sf, 10)

df <- data.frame(
  lat = c(44.04905, 44.17609),
  lng = c(-74.01188, -73.83493)
)

df_sf <- sf::st_as_sf(df, coords = c("lng", "lat"))
df_sf <- sf::st_set_crs(df_sf, 4326)

set_bbox_side_length(df_sf, 4000)
```

`combine_overlays` *Combine multiple image overlays into a single file*

Description

This function combines any number of images into a single file, which may then be further processed as an image or transformed into an image overlay.

Usage

```
combine_overlays(
  ...,
  output_file = tempfile(fileext = ".png"),
  transparency = 0
)
```

Arguments

`...`	File paths for images to be combined. Note that combining TIFF images requires the `tiff` package be installed.
`output_file`	The path to save the resulting image to. Can be any format accepted by magick::image_read. Optionally, can be set to `NULL`, in which case this function will return the image as a `magick` object instead of writing to disk.
`transparency`	A value indicating how much transparency should be added to each image. If less than 1, interpreted as a proportion (so a value of 0.1 results in each image becoming 10% more transparent); if between 1 and 100, interpreted as a percentage (so a value of 10 results in each image becoming 10% more transparent.) A value of 0 is equivalent to no additional transparency.

Value

If `output_file` is not null, `output_file`, invisibly. If `output_file` is null, a `magick` image object.

See Also

Other data manipulation functions: `georeference_overlay()`, `merge_rasters()`, `raster_to_raw_tiles()`, `vector_to_overlay()`

Other overlay creation functions: `georeference_overlay()`, `vector_to_overlay()`

Other visualization functions: `geom_spatial_rgb()`, `raster_to_raw_tiles()`, `vector_to_overlay()`

Examples

```
## Not run:
# Generate points and download orthoimagery
mt_elbert_points <- data.frame(
  lat = runif(100, min = 39.11144, max = 39.12416),
  lng = runif(100, min = -106.4534, max = -106.437)
)

mt_elbert_sf <- sf::st_as_sf(mt_elbert_points, coords = c("lng", "lat"))
sf::st_crs(mt_elbert_sf) <- sf::st_crs(4326)

output_files <- get_tiles(
  mt_elbert_sf,
  output_prefix = tempfile(),
  services = c("ortho")
)

# Merge orthoimagery into a single file
ortho_merged <- merge_rasters(
  input_rasters = output_files[1],
  output_raster = tempfile(fileext = ".tif")
)

# Convert our points into an overlay
mt_elbert_overlay <- vector_to_overlay(mt_elbert_sf,
  ortho_merged[[1]],
  size = 15,
  color = "red",
  na.rm = TRUE
)

# Combine the overlay with our orthoimage
ortho_with_points <- combine_overlays(
  ortho_merged[[1]],
  mt_elbert_overlay
)

## End(Not run)
```

`geom_spatial_rgb` *Plot RGB rasters in ggplot2*

Description

`geom_spatial_rgb` and `stat_spatial_rgb` allow users to plot three-band RGB rasters in ggplot2, using these layers as background base maps for other spatial plotting. Note that unlike ggplot2::geom_sf, this function does *not* force ggplot2::coord_sf; for accurate mapping, add ggplot2::coord_sf with a `crs` value matching your input raster as a layer.

Usage

```
geom_spatial_rgb(
  mapping = NULL,
  data = NULL,
  stat = "spatialRGB",
  position = "identity",
  ...,
  hjust = 0.5,
  vjust = 0.5,
  interpolate = FALSE,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  scale = NULL
)

stat_spatial_rgb(
  mapping = NULL,
  data = NULL,
  geom = "raster",
  position = "identity",
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  scale = NULL,
  ...
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. In addition to the three options described in ggplot2::geom_raster, there are two additional methods:

	If a SpatRaster object (see terra::rast), this function will coerce the raster to a data frame and assume the raster bands are in RGB order (while allowing for, but ignoring, a fourth alpha band). If a length-1 character vector, this function will attempt to load the object via terra::rast.
stat	The statistical transformation to use on the data for this layer, either as a ggproto Geom subclass or as a string naming the stat stripped of the stat_ prefix (e.g. "count" rather than "stat_count")
position	Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use position_jitter), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.
...	Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.
hjust, vjust	horizontal and vertical justification of the grob. Each justification value should be a number between 0 and 1. Defaults to 0.5 for both, centering each pixel over its data location.
interpolate	If TRUE interpolate linearly, if FALSE (the default) don't interpolate.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders().
scale	Integer. Maximum (possible) value in the three channels. If NULL, attempts to infer proper values from data – if all RGB values are <= 1 then 1, <= 255 then 255, and otherwise 65535.
geom	The geometric object to use to display the data, either as a ggproto Geom subclass or as a string naming the geom stripped of the geom_ prefix (e.g. "point" rather than "geom_point")

See Also

Other visualization functions: combine_overlays(), raster_to_raw_tiles(), vector_to_overlay()

Examples

```
## Not run:

simulated_data <- data.frame(
  id = seq(1, 100, 1),
  lat = runif(100, 44.04905, 44.17609),
  lng = runif(100, -74.01188, -73.83493)
)
```

```
simulated_data <- sf::st_as_sf(simulated_data, coords = c("lng", "lat"))
simulated_data <- sf::st_set_crs(simulated_data, 4326)

output_tiles <- get_tiles(simulated_data,
  services = c("ortho"),
  resolution = 120
)

merged_ortho <- tempfile(fileext = ".tif")
merge_rasters(output_tiles[["ortho"]], merged_ortho)

merged_stack <- terra::rast(merged_ortho)

library(ggplot2)

ggplot() +
  geom_spatial_rgb(
    data = merged_ortho,
    mapping = aes(
      x = x,
      y = y,
      r = red,
      g = green,
      b = blue
    )
  ) +
  geom_sf(data = simulated_data) +
  coord_sf(crs = 4326)

ggplot() +
  geom_spatial_rgb(
    data = merged_stack,
    mapping = aes(
      x = x,
      y = y,
      r = red,
      g = green,
      b = blue
    )
  ) +
  geom_sf(data = simulated_data) +
  coord_sf(crs = 4326)

## End(Not run)
```

`georeference_overlay` *Georeference image overlays based on a reference raster*

Description

This function georeferences an image overlay based on a reference raster, setting the extent and CRS of the image to those of the raster file. To georeference multiple images and merge them into a single file, see merge_rasters.

Usage

```
georeference_overlay(
  overlay_file,
  reference_raster,
  output_file = tempfile(fileext = ".tif")
)
```

Arguments

`overlay_file`	The image overlay to georeference. File format will be detected automatically from file extension; options include `jpeg/jpg`, `png`, and `tif/tiff`.
`reference_raster`	The raster file to base georeferencing on. The output image will have the same extent and CRS as the reference raster. Accepts anything that can be read by terra::rast
`output_file`	The path to write the georeferenced image file to. Must be a TIFF.

Value

The file path written to, invisibly.

See Also

Other data manipulation functions: `combine_overlays()`, `merge_rasters()`, `raster_to_raw_tiles()`, `vector_to_overlay()`

Other overlay creation functions: `combine_overlays()`, `vector_to_overlay()`

Examples

```
## Not run:
simulated_data <- data.frame(
  id = seq(1, 100, 1),
  lat = runif(100, 44.1114, 44.1123),
  lng = runif(100, -73.92273, -73.92147)
)

simulated_data <- sf::st_as_sf(simulated_data, coords = c("lng", "lat"))

downloaded_tiles <- get_tiles(simulated_data,
  services = c("elevation", "ortho"),
  georeference = FALSE
)
```

```
georeference_overlay(
  overlay_file = downloaded_tiles[[2]],
  reference_raster = downloaded_tiles[[1]],
  output_file = tempfile(fileext = ".tif")
)

## End(Not run)
```

---

get_tiles *A user-friendly way to get USGS National Map data tiles for an area*

Description

This function splits the area contained within a bounding box into a set of tiles, and retrieves data from the USGS National map for each tile. As of version 0.5.0, the method for lists has been deprecated.

Usage

```
get_tiles(
  data,
  output_prefix = tempfile(),
  side_length = NULL,
  resolution = 1,
  services = "elevation",
  verbose = FALSE,
  georeference = TRUE,
  projected = NULL,
  ...
)

## S3 method for class 'sf'
get_tiles(
  data,
  output_prefix = tempfile(),
  side_length = NULL,
  resolution = 1,
  services = "elevation",
  verbose = FALSE,
  georeference = TRUE,
  projected = NULL,
  ...
)

## S3 method for class 'sfc'
get_tiles(
  data,
```

```
  output_prefix = tempfile(),
  side_length = NULL,
  resolution = 1,
  services = "elevation",
  verbose = FALSE,
  georeference = TRUE,
  projected = NULL,
  ...
)

## S3 method for class 'Raster'
get_tiles(
  data,
  output_prefix = tempfile(),
  side_length = NULL,
  resolution = 1,
  services = "elevation",
  verbose = FALSE,
  georeference = TRUE,
  projected = NULL,
  ...
)

## S3 method for class 'SpatRaster'
get_tiles(
  data,
  output_prefix = tempfile(),
  side_length = NULL,
  resolution = 1,
  services = "elevation",
  verbose = FALSE,
  georeference = TRUE,
  projected = NULL,
  ...
)

## S3 method for class 'list'
get_tiles(
  data,
  output_prefix = tempfile(),
  side_length = NULL,
  resolution = 1,
  services = "elevation",
  verbose = FALSE,
  georeference = TRUE,
  projected = NULL,
  ...
)
```

Arguments

`data`	An `sf` or `SpatRaster` object; tiles will be downloaded for the full extent of the provided object.
`output_prefix`	The file prefix to use when saving tiles.
`side_length`	The length, in meters, of each side of tiles to download. If `NULL`, defaults to the maximum side length permitted by the least permissive service requested.
`resolution`	How many meters are represented by each pixel? The default value of 1 means that 1 pixel = 1 meter, while a value of 2 means that 1 pixel = 2 meters, and so on.
`services`	A character vector of services to download data from. Current options include "3DEPElevation", "USGSNAIPPlus", and "nhd". Users can also use short codes to download a specific type of data without specifying the source; current options for short codes include "elevation" (equivalent to "3DEPElevation"), "ortho" (equivalent to "USGSNAIPPlus), and "hydro" ("nhd"). Short codes are not guaranteed to refer to the same source across releases. Short codes are converted to their service name and then duplicates are removed, so any given source will only be queried once per tile.
`verbose`	Logical: should tile retrieval functions run in verbose mode?
`georeference`	Logical: should tiles be downloaded as PNGs without georeferencing, or should they be downloaded as georeferenced TIFF files? This option does nothing when only elevation data is being downloaded.
`projected`	Logical: is `data` in a projected coordinate reference system? If `NULL`, the default, inferred from sf::st_is_longlat.
`...`	Additional arguments passed to hit_national_map_api. These can be used to change default query parameters or as additional options for the National Map services. See below for more details.

Value

A list of the same length as the number of unique services requested, containing named vectors of where data files were saved to. Returned invisibly.

Available Datasources

The following services are currently available (with short codes in parentheses where applicable). See links for API documentation.

- 3DEPElevation (short code: elevation)
- USGSNAIPPlus (short code: ortho)
- USGSNAIPImagery
- nhd (short code: hydro)
- govunits
- contours
- geonames

- NHDPlus_HR
- structures
- transportation
- wbd ("short code": watersheds)
- ecosystems
- USGSTopo
- USGSShadedReliefOnly
- USGSImageryOnly
- USGSHydroCached
- USGSTNMBlank

Additional Arguments

The ... argument can be used to pass additional arguments to the National Map API or to edit the hard-coded defaults used by this function. More information on common arguments to change can be found in hit_national_map_api. Note that ... can also be used to change the formats returned by the server, but that doing so while using this function will likely cause the function to error (or corrupt the output data). To download files in different formats, use hit_national_map_api.

See Also

Other data retrieval functions: `hit_national_map_api()`

Examples

```
## Not run:
simulated_data <- data.frame(
  id = seq(1, 100, 1),
  lat = runif(100, 44.04905, 44.17609),
  lng = runif(100, -74.01188, -73.83493)
)

simulated_data <- sf::st_as_sf(simulated_data, coords = c("lng", "lat"))

get_tiles(simulated_data, tempfile())

## End(Not run)
```

make_manifest *Transform rasters and write manifest file for import into Unity*

Description

These functions crop input raster files into smaller square tiles and then converts them into either .png or .raw files which are ready to be imported into the Unity game engine. make_manifest also writes a "manifest" file and importer script which may be used to automatically import the tiles into Unity.

Usage

```
make_manifest(
  heightmap,
  overlay = NULL,
  output_prefix = "import",
  manifest_path = "terrainr.manifest",
  importer_path = "import_terrain.cs"
)

transform_elevation(heightmap, side_length = 4097, output_prefix = "import")

transform_overlay(overlay, side_length = 4097, output_prefix = "import")
```

Arguments

heightmap	File path to the heightmap to transform.
overlay	File path to the image overlay to transform. Optional for make_manifest.
output_prefix	The file path to prefix output tiles with.
manifest_path	File path to write the manifest file to.
importer_path	File name to write the importer script to. Set to NULL to not copy the importer script. Will overwrite any file at the same path.
side_length	Side length, in pixels, of each output tile. If the raster has dimensions not evenly divisible by side_length, tiles will be generated with overhanging pieces set to 0 units of elevation or RGB 0 (pure black). Side lengths not equal to 2^x + 1 (for x <= 12) will cause a warning, as tiles must be this size for import into Unity.

Value

manifest_path, invisibly.

Examples

```
## Not run:
if (!isTRUE(as.logical(Sys.getenv("CI")))) {
  simulated_data <- data.frame(
    id = seq(1, 100, 1),
    lat = runif(100, 44.04905, 44.17609),
    lng = runif(100, -74.01188, -73.83493)
  )
  simulated_data <- sf::st_as_sf(simulated_data, coords = c("lng", "lat"))
  output_files <- get_tiles(simulated_data)
  temptiff <- tempfile(fileext = ".tif")
  merge_rasters(output_files["elevation"][[1]], temptiff)
  make_manifest(temptiff, output_prefix = tempfile(), importer_path = NULL)
}

## End(Not run)
```

`make_unity` *Initialize terrain inside of a Unity project.*

Description

Initialize terrain inside of a Unity project.

Usage

```
make_unity(
  project,
  heightmap,
  overlay = NULL,
  side_length = 4097,
  scene_name = "terrainr_scene",
  action = TRUE,
  unity = find_unity()
)
```

Arguments

`project`	The directory path of the Unity project to create terrain inside.
`heightmap`	The file path for the raster to transform into terrain.
`overlay`	Optionally, a file path for an image overlay to layer on top of the terrain surface. Leave as NULL for no overlay.
`side_length`	The side length, in map units, for the terrain tiles. Must be equal to 2^x + 1, for any x between 5 and 12.
`scene_name`	The name of the Unity scene to create the terrain in.

action	Boolean: Execute the unifir "script" and create the Unity project? If FALSE, returns a non-executed script.
unity	The location of the Unity executable to create projects with. By default, will be auto-detected by unifir::find_unity

Value

An object of class "unifir_script", containing either an executed unifir script (if action = TRUE) or a non-executed script object (if action = FALSE).

Examples

```
## Not run:
if (!isTRUE(as.logical(Sys.getenv("CI")))) {
  simulated_data <- data.frame(
    id = seq(1, 100, 1),
    lat = runif(100, 44.04905, 44.17609),
    lng = runif(100, -74.01188, -73.83493)
  )
  simulated_data <- sf::st_as_sf(simulated_data, coords = c("lng", "lat"))
  output_files <- get_tiles(simulated_data)
  temptiff <- tempfile(fileext = ".tif")
  merge_rasters(output_files["elevation"][[1]], temptiff)
  make_unity(file.path(tempdir(), "unity"), temptiff)
}

## End(Not run)
```

---

merge_rasters *Merge multiple raster files into a single raster*

Description

Some functions like get_tiles return multiple separate files when it can be useful to have a single larger raster instead. This function is a thin wrapper over sf::gdal_utils, making it easy to collapse those multiple raster files into a single TIFF.

Usage

```
merge_rasters(
  input_rasters,
  output_raster = tempfile(fileext = ".tif"),
  options = character(0),
  overwrite = FALSE,
  force_fallback = FALSE
)
```

Arguments

input_rasters	A character vector containing the file paths to the georeferenced rasters you want to use.
output_raster	The file path to save the merged georeferenced raster to.
options	Optionally, a character vector of options to be passed directly to sf::gdal_utils. If the fallback is used and any options (other than "-overwrite") are specified, this will issue a warning.
overwrite	Logical: overwrite output_raster if it exists? If FALSE and the file exists, this function will fail with an error. The behavior if this argument is TRUE and "-overwrite" is passed to options directly is not stable.
force_fallback	Logical: if TRUE, uses the much slower fallback method by default. This is used for testing purposes and is not recommended for use by end users.

Value

output_raster, invisibly.

See Also

Other data manipulation functions: combine_overlays(), georeference_overlay(), raster_to_raw_tiles(), vector_to_overlay()

Examples

```
## Not run:
simulated_data <- data.frame(
  lat = c(44.10379, 44.17573),
  lng = c(-74.01177, -73.91171)
)

simulated_data <- sf::st_as_sf(simulated_data, coords = c("lng", "lat"))

img_files <- get_tiles(simulated_data)
merge_rasters(img_files[[1]])

## End(Not run)
```

---

raster_to_raw_tiles *Crop a raster and convert the output tiles into new formats.*

Description

This function has been deprecated as of terrainr 0.5.0 in favor of the new function, make_manifest. While it will be continued to be exported until at least 2022, improvements and bug fixes will only be made to the new function. Please open an issue if any features you relied upon is missing from the new function!

Usage

```
raster_to_raw_tiles(input_file, output_prefix, side_length = 4097, raw = TRUE)
```

Arguments

input_file	File path to the input TIFF file to convert.
output_prefix	The file path to prefix output tiles with.
side_length	The side length, in pixels, for the .raw tiles.
raw	Logical: Convert the cropped tiles to .raw? When FALSE returns a .png.

Details

This function crops input raster files into smaller square tiles and then converts them into either .png or .raw files which are ready to be imported into the Unity game engine.

Value

Invisibly, a character vector containing the file paths that were written to.

See Also

Other data manipulation functions: combine_overlays(), georeference_overlay(), merge_rasters(), vector_to_overlay()

Other visualization functions: combine_overlays(), geom_spatial_rgb(), vector_to_overlay()

Examples

```
## Not run:
if (!isTRUE(as.logical(Sys.getenv("CI")))) {
  simulated_data <- data.frame(
    id = seq(1, 100, 1),
    lat = runif(100, 44.04905, 44.17609),
    lng = runif(100, -74.01188, -73.83493)
  )
  simulated_data <- sf::st_as_sf(simulated_data, coords = c("lng", "lat"))
  output_files <- get_tiles(simulated_data)
  temptiff <- tempfile(fileext = ".tif")
  merge_rasters(output_files["elevation"][[1]], temptiff)
  raster_to_raw_tiles(temptiff, tempfile())
}

## End(Not run)
```

vector_to_overlay *Turn spatial vector data into an image overlay*

Description

This function allows users to quickly transform any vector data into an image overlay, which may then be imported as a texture into Unity.

Usage

```
vector_to_overlay(
  vector_data,
  reference_raster,
  output_file = NULL,
  transparent = "#ffffff",
  ...,
  error_crs = NULL
)
```

Arguments

vector_data	The spatial vector data set to be transformed into an overlay image. Users may provide either an sf object or a length 1 character vector containing a path to a file readable by sf::read_sf.
reference_raster	The raster file to produce an overlay for. The output overlay will have the same extent and resolution as the input raster. Users may provide either a Raster* object or a length 1 character vector containing a path to a file readable by terra::rast.
output_file	The path to save the image overlay to. If NULL, saves to a tempfile.
transparent	The hex code for a color to be made transparent in the final image. Set to FALSE to not set any colors to transparent.
...	Arguments passed to ... in either ggplot2::geom_point (for point vector data), ggplot2::geom_line (for line data), or ggplot2::geom_polygon (for all other data types).
error_crs	Logical: Should this function error if data has no CRS? If TRUE, function errors; if FALSE, function quietly assumes EPSG:4326. If NULL, the default, function assumes EPSG:4326 with a warning.

Value

output_file, invisibly.

See Also

Other data manipulation functions: combine_overlays(), georeference_overlay(), merge_rasters(), raster_to_raw_tiles()

Other overlay creation functions: combine_overlays(), georeference_overlay()

Other visualization functions: combine_overlays(), geom_spatial_rgb(), raster_to_raw_tiles()

Examples

```
## Not run:

# Generate points to download raster tiles for
set.seed(123)
simulated_data <- data.frame(
  id = seq(1, 100, 1),
  lat = runif(100, 44.1114, 44.1123),
  lng = runif(100, -73.92273, -73.92147)
)

# Create an sf object from our original simulated data

simulated_data_sf <- sf::st_as_sf(simulated_data, coords = c("lng", "lat"))
sf::st_crs(simulated_data_sf) <- sf::st_crs(4326)

# Download data!

downloaded_tiles <- get_tiles(simulated_data_sf, tempfile())

merged_file <- merge_rasters(
  downloaded_tiles[[1]],
  tempfile(fileext = ".tif")
)


# Create an overlay image
vector_to_overlay(simulated_data_sf, merged_file[[1]], na.rm = TRUE)

## End(Not run)
```

Index