Package: moveVis (via r-universe)

May 19, 2026

Type Package

Title Movement Data Visualization

Version 1.0.1

Depends R (>= 3.5.0)

Date 2025-08-22

Description Tools to visualize movement data (e.g. from GPS tracking) and temporal changes of environmental data (e.g. from remote sensing) by creating video animations.

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.2

Imports move2, terra, sf, s2, lwgeom, units, basemaps, lubridate, ggplot2, rlang, ggnewscale, patchwork, gifski, av, pbapply, magrittr, methods, stats

BugReports https://www.github.com/16eagle/moveVis/issues

SystemRequirements ImageMagick, FFmpeg, libav

URL https://movevis.org

Suggests parallel, mapview, leaflet, testthat

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libmagick++-dev gsfonts libavfilter-dev libpng-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev libclang-dev

Repository https://16eagle.r-universe.dev

Date/Publication 2025-08-22 13:14:28 UTC

RemoteUrl https://github.com/16EAGLE/moveVis

RemoteRef HEAD

RemoteSha cf88cdcc0a44bd48c7f0bd35bc3f40f6d3f94283

Contents

moveVis-package . . . 2
[.moveVis . . . 5
[[.moveVis . . . 5
add_colourscale . . . 6
add_gg . . . 8
add_labels . . . 12
add_northarrow . . . 13
add_progress . . . 15
add_scalebar . . . 16
add_text . . . 18
add_timestamps . . . 20
align_move . . . 21
animate_frames . . . 23
c.moveVis . . . 25
deprecated . . . 26
frames_graph . . . 26
frames_spatial . . . 29
get_frametimes . . . 35
join_frames . . . 36
length.moveVis . . . 38
move_data . . . 39
print.moveVis . . . 39
raster_data . . . 40
rev.moveVis . . . 41
settings . . . 41
suggest_formats . . . 43
tail.moveVis . . . 44
view_spatial . . . 44
whitestork_data . . . 46

Index **47**

`moveVis-package` *Tools to visualize movement data in R*

Description

`moveVis` provides tools to visualize movement data (e.g. from GPS tracking) and temporal changes of environmental data (e.g. from remote sensing) by creating video animations. The `moveVis` package is closely connected to the `move2` package and builds up on `ggplot2` grammar of graphics.

Details

The package includes the following functions, sorted by the order they would be applied to create an animation from movement data:

Preparing movement tracks:

- `align_move` aligns single and multi-individual movement data to a uniform time scale with a uniform temporal resolution needed for creating an animation from it. Use this function to prepare your movement data for animation depending on the temporal resolution that suits your data.

Creating frames:

- `get_maptypes` returns available map services and types that can be used with `frames_spatial`. This function is reexported from the basemaps package.
- `frames_spatial` creates `moveVis` frames from movement and map/raster data, displaying movement-environment interactions spatio-temporally. Frames are returned as an object of class `moveVis` and can be subsetted, viewed (see `render_frame`), modified (see `add_gg` and associated functions) and animated (see `animate_frames`).
- `frames_graph` creates `moveVis` frames displaying movement-environment interaction graphs. Frames can be viewed or modified individually and animated using `animate_frames`.

Adapting frames:

- `add_gg` adds `ggplot2` expressions (e.g. to add layers such as points, polygons, lines, or to change scales etc.) to frames created with `frames_spatial` or `frames_graph`.
- `add_labels` adds character labels such as title or axis labels to frames created with `frames_spatial` or `frames_graph`.
- `add_scalebar` adds a scalebar to frames created with `frames_spatial` or `frames_graph`.
- `add_northarrow` adds a north arrow to frames created with `frames_spatial` or `frames_graph`.
- `add_progress` adds a progress bar to frames created with `frames_spatial` or `frames_graph`.
- `add_timestamps` adds timestamps to frames created with `frames_spatial` or `frames_graph`.
- `add_text` adds static or dynamically changing text to frames created with `frames_spatial` or `frames_graph`.
- `add_colourscale` adjusts the colour scales of frames created with `frames_spatial` and custom map imagery.
- `join_frames` side-by-side two or more sets of frames of equal lengths into one set of frames using `wrap_plots`, e.g. to combine spatial frames returned by `frames_spatial` with graph frames returned by `frames_graph`.
- `get_frametimes` extracts the timestamps associated with each frame of frames created using `frames_spatial` or `frames_graph` and returns them as a vector.

Animating frames (as GIF or video):

- `suggest_formats` returns a selection of suggested file formats that can be used with `out_file` of `animate_frames` on your system.
- `animate_frames` creates an animation from `moveVis` frames, e.g as `.gif` or `.mov` video file.

Viewing movement tracks:

- render_frame renders an individual frame. It yields the same result as if an individual frame is extracted using [[]].
- view_spatial displays movement tracks on an interactive mapview or leaflet map.

Methods:

- [.moveVis extracts individual frames or a sequence of frames from a moveVis frames object.
- [[.moveVis renders an individual frame, like render_frame does.
- c combines multiple moveVis frames.
- tail and head return n last or first frames of a moveVis frames object.
- length returns the length of moveVis frames, i.e. number of frames.
- print shows basic information about a moveVis frames object, i.e. number of frames, extent and more.
- rev reverses the order of a moveVis frames object.

Processing settings:

- use_multicore enables multi-core usage for computational expensive processing steps.
- use_disk enables the usage of disk space for creating frames, which can prevent memory overload when creating frames for very large animations.

Example data:

- move_data, a move2 object representing coordinates and acquisition times of three simulated movement tracks, covering a location nearby Lake of Constance, Germany.
- whitestork_data, a data.frame and a move2 object, both representing coordinates and acquisition times of 15 White Storks, migrating from Lake of Constance, SW Germany, to Africa.
- example_data, two SpatRasterDatasets, representing simulated NDVI images and an NDVI-derived classification covering the Lake of Constance area, as well as invented dates and times that simulate acquisition times.

The majority of these functions can be used with the forward pipe operator %>%, which is re-exported by moveVis.

Author(s)

Jakob Schwalb-Willmann. Maintainer: Jakob Schwalb-Willmann, moveVis@schwalb-willmann.de

See Also

Useful links:

- https://movevis.org
- Report bugs at https://www.github.com/16eagle/moveVis/issues

[.moveVis *Extract moveVis frames*

Description

Method for extracting individual frames or a sequence of frames from a `moveVis` frames object.

Usage

```
## S3 method for class 'moveVis'
x[i, ...]
```

Arguments

`x`	an object of class `moveVis`.
`i`	numeric, index number or sequence of index numbers of the frame(s) to be extracted.
`...`	further arguments passed to or from other methods.

Value

A `moveVis` frames object.

[[.moveVis *Render an individual frame*

Description

This function renders an individual frame. It yields the same result as if an individual frame is extracted using default subsetting `[[]]`.

Usage

```
## S3 method for class 'moveVis'
x[[i, ...]]

render_frame(frames, i = length(frames))
```

Arguments

`x`	frames as an object of class moveVis.
`i`	numeric, index number of the frame to be rendered.
`...`	additional arguments, currently not used.
`frames`	frames as an object of class moveVis.

Examples

```
library(moveVis)
library(move2)
library(terra)

data("move_data", package = "moveVis")
r <- readRDS(example_data(file = "raster_NDVI.rds"))

# align movement
m <- align_move(move_data, res = units::set_units(4, "min"))

# create spatial frames
frames <- frames_spatial(m, r, r_type = "gradient", fade_raster = TRUE)

# viewing frames calling this function:
render_frame(frames) # displays the last frame
render_frame(frames, i = 100) # displays frame 100

# alternatively, you can simply use `[[` to do this:
frames[[100]] # displays frame 100
```

`add_colourscale` *Add scale to frames*

Description

This function adjusts the colour scales of frames created with `frames_spatial` and custom map imagery using `ggplot2`.

Usage

```
add_colourscale(
  frames,
  type,
  colours,
  labels = waiver(),
  na.colour = "grey50",
  na.show = TRUE,
  legend_title = NULL,
  verbose = TRUE
)
```

Arguments

`frames`	an object of class `moveVis` created with `frames_spatial`.
`type`	character, either `"gradient"` or `"discrete"`. Must be equal to the definition of argument `r_type` with which `frames` have been created (see `frames_spatial`).

colours	character, a vector of colours. If type = "discrete", number of colours must be equal to the number of classes contained in the raster imagery with which frames have been created. Optionally, the vector can be named to associate map values with colours and define the scale limits, e.g. c("-1" = "red", "0" = "blue", "1" = "green")
labels	character, a vector of labels with the same length as colours. Ignored, if type = "gradient".
na.colour	character, colour to use for missing values.
na.show	logical, whether to display NA values in discrete scaling. Ignored, if type = "gradient".
legend_title	character, a legend title.
verbose	logical, if TRUE, messages and progress information are displayed on the console (default).

Details

Instead of using this function, you can use add_gg to apply any ggplot2 on moveVis frames yourself.

Value

A frames object of class moveVis.

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial frames_graph animate_frames

Examples

```
library(moveVis)
library(move2)
library(terra)

data("move_data", package = "moveVis")
r <- readRDS(example_data(file = "raster_NDVI.rds"))

# align movement
m <- align_move(move_data, res = units::set_units(4, "min"))

## Not run:
# create spatial frames with frames_spatial:
frames <- frames_spatial(
  m, r, r_type = "gradient", fade_raster = TRUE
)
frames[[100]] # take a look at one of the frames
```

```
# change the colour scale of all frames
frames <- add_colourscale(
  frames, type = "gradient", colours = c("orange", "white", "darkgreen"),
  legend_title = "NDVI"
)
frames[[100]]

r <- terra::sds(lapply(r, function(x){
  y <- x
terra::values(y) <- round(terra::values(y)*10)
  return(y)
}))

# turn fade_raster to FALSE, since it makes no sense to temporally interpolate discrete classes
frames <- frames_spatial(
  m, r, r_type = "discrete",
  fade_raster = FALSE
)
frames[[100]]
# now, let's assign a colour per class value to frames
colFUN <- colorRampPalette(c("orange", "lightgreen", "darkgreen"))
cols <- colFUN(10)
frames <- add_colourscale(
  frames, type = "discrete", colours = cols, legend_title = "Classes"
)
frames[[100]]

## End(Not run)
```

---

add_gg *Add* ggplot2 *function to frames*

Description

This function applies ggplot2 functions (e.g. to add layers, change scales etc.) to frames created with frames_spatial or frames_graph.

Usage

```
add_gg(frames, gg, data = NULL, ..., verbose = T)
```

Arguments

frames	an object of class moveVis created with frames_spatial.
gg	ggplot2 expressions (see details), either as • an expression of one or a list of ggplot2 functions to be added to every frame,

- a list of such of the same length as `frames` to add different `ggplot2` expressions per frame

`data` optional data used by gg (see details), either

- an object of any class, e.g. a `data.frame`, used by gg that will be added to all frames,
- a list, e.g. of multiple `data.frames`, with length of `frames` to add different data to each frame.

`...` additional (non-iterated) objects that should be visible to gg.

`verbose` logical, if `TRUE`, messages and progress information are displayed on the console (default).

Details

Argument gg expects `ggplot2` functions handed over as expressions (see `expr`) to avoid their evaluation before they are called for the correct frame. Simply wrap your `ggplot2` function into `expr` and supply it to gg. To add multiple `ggplot2` functions to be applied on every frame, supply an expression containing a list of `ggplot2` functions (e.g. `expr(list(geom_label(...), geom_text(...)))`). This expression would be added to all frames. To add specific `ggplot2` functions per frame, supply a list of expressions of the same length as frames. Each expression may contain a list of `ggplot2` functions, if you want to add multiple functions per frame.

If `data` is used, the `ggplot2` expressions supplied with gg can use the object by the name `data` for plotting. If `data` is a list, it must be of the same length as `frames`. The list will be iterated, so that functions in gg will have access to the individual objects within the list by the name `data` per each frame. If the data you want to display is does not change with frames and may only be a character vector or similiar, you may not need `data`, as you can supply the needed values within the expression supplied through gg.

If you supply gg as a list of expressions for each frame and `data` as a list of objects (e.g. data.frames) for each frame, each frame will be manipulated with the corresponding `ggplot2` function and the corresponding data.

Value

A frames object of class `moveVis`.

Author(s)

Jakob Schwalb-Willmann

See Also

`frames_spatial` `frames_graph` `animate_frames`

Examples

```
library(moveVis)
library(move2)
library(terra)
library(sf)
```

```
library(ggplot2)

data("move_data", package = "moveVis")
r <- readRDS(example_data(file = "raster_NDVI.rds"))

# align movement
m <- align_move(move_data, res = units::set_units(4, "min"))

# let's create a data.frame containing corner coordinates
data <- cbind(
  x = c(8.96, 8.955, 8.959, 8.963, 8.968, 8.963, 8.96),
  y = c(47.725, 47.728, 47.729, 47.728, 47.725, 47.723, 47.725)
)

# make an sf object projected to Pseudo Mercator to match basemap crs
data <- list(data) %>%
  st_polygon() %>%
  st_geometry() %>%
  st_as_sf(crs = st_crs(4326)) %>%
  st_transform(crs = st_crs(3857))

# add the polygon to all frames

## Not run:
# create frames with add_gg additions
frames <- frames_spatial(
  m, map_service = "osm", map_type = "topographic", alpha = 0.5) %>%
  # add polygon
  add_gg(gg = expr(geom_sf(
    data = data, colour = "black", fill = "transparent",
    linetype = "dashed", lwd = 1)
  ), data = data) %>%
  # add coordinate system
  add_gg(gg = expr(
    coord_sf(datum = st_crs(3857), expand = F)
  ), data = data)

# take alook
frames[[100]]

# add_gg can also be used iteratively to manipulate each frame differently.
# let's create unique polygons per frame:

# create data.frame containing corner coordinates
data <- cbind(
  x = c(8.96, 8.955, 8.959, 8.963, 8.968, 8.963, 8.96),
  y = c(47.725, 47.728, 47.729, 47.728, 47.725, 47.723, 47.725)
)
# make a list from it by replicating it by the length of frames
data <- rep(list(data), length.out = length(frames))

# let's alter the corner coordinates so that each polygon is different
data <- lapply(data, function(x){
```

```
  y <- rnorm(nrow(x)-1, mean = 0.00001, sd = 0.0001)
  x + c(y, y[1])
})

#' # make sf object from the data list contents and project to pseudo mercator
data <- lapply(data, function(x){
  list(x) %>%
    st_polygon() %>%
    st_geometry() %>%
    st_as_sf(crs = st_crs(4326)) %>%
    st_transform( crs = st_crs(3857))
})

# apply add_gg
frames <- frames_spatial(
  m, map_service = "osm", map_type = "topographic", alpha = 0.5) %>%
  # add polygons
  add_gg(gg = expr(geom_sf(
    data = data,
    colour = "black",
    fill = "transparent",
    linetype = "dashed",
    lwd = 1)
  ), data = data) %>%
  # add coordinate system
  add_gg(gg = expr(coord_sf(
    datum = st_crs(3857), expand = F)
  ), data = data)

frames[[100]]

# animate frames to see how the polygons "flip"
animate_frames(frames, out_file = tempfile(fileext = ".mov"))

## End(Not run)
# you can use add_gg on any list of ggplot2 objects,
# also on frames made using frames_graph
frames <- frames_graph(
  m, r, r_type = "gradient",
  fade_raster = TRUE, graph_type = "hist", val_by = 0.01
)
frames[[100]]

# manipulate the labels, since they are very dense:
# just replace the current scale
frames <- add_gg(frames, expr(scale_x_continuous(
  breaks=seq(0,1,0.1),
  labels=seq(0,1,0.1), expand = c(0,0)))
)
frames[[100]]
```

add_labels *Add labels to frames*

Description

This function adds character labels such as titles or axis labels to frames created with frames_spatial.

Usage

```
add_labels(
  frames,
  title = waiver(),
  subtitle = waiver(),
  caption = waiver(),
  tag = waiver(),
  x = waiver(),
  y = waiver(),
  verbose = TRUE
)
```

Arguments

frames	an object of class moveVis created with frames_spatial.
title	character, frame title. If NULL, an existing title of frames is removed. If waiver() (default, see waiver), an existing title of frames is kept.
subtitle	character, frame subtitle. If NULL, an existing title of frames is removed. If waiver() (default, see waiver), an existing title of frames is kept.
caption	character, frame caption. If NULL, an existing title of frames is removed. If waiver() (default, see waiver), an existing title of frames is kept.
tag	character, frame tag. If NULL, an existing title of frames is removed. If waiver() (default, see waiver), an existing title of frames is kept.
x	character, label of the x axis. If NULL, an existing title of frames is removed. If waiver() (default, see waiver), an existing title of frames is kept.
y	character, label of the y axis. If NULL, an existing title of frames is removed. If waiver() (default, see waiver), an existing title of frames is kept.
verbose	logical, if TRUE, messages and progress information are displayed on the console (default).

Value

A frames object of class moveVis.

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial frames_graph animate_frames

Examples

```
library(moveVis)
library(move2)
library(terra)

data("move_data", package = "moveVis")
r <- readRDS(example_data(file = "raster_NDVI.rds"))

# align movement
m <- align_move(move_data, res = units::set_units(4, "min"))

# create frames and add labels
frames <- frames_spatial(
  m, r, r_type = "gradient", fade_raster = TRUE
)

# add labels
frames <- add_labels(
  frames,
  title = "Example animation using moveVis::add_labels()",
  subtitle = "Adding a subtitle to frames created using frames_spatial()",
  caption = "Projection: Geographical, WGS84. Sources: moveVis examples.",
  x = "Longitude", y = "Latitude"
)

# have a look at one frame
frames[[100]]
```

---

add_northarrow *Add north arrow to frames*

Description

This function adds a north arrow to frames created with frames_spatial.

Usage

```
add_northarrow(
  frames,
  height = 0.05,
  position = "bottomright",
  x = NULL,
  y = NULL,
  colour = "black",
```

```
  size = 1,
  label_text = "N",
  label_margin = 0.4,
  label_size = 5,
  verbose = TRUE
)
```

Arguments

frames	an object of class moveVis created with frames_spatial.
height	numeric, height of the north arrow in a range from 0 to 1 as the proportion of the overall height of the frame map.
position	character, position of the north arrow on the map. Either "bottomleft", "upperleft", "upperright", "bottomright". Ignored, if x and y are set.
x	numeric, position of the bottom left corner of the north arrow on the x axis. If not set, position is used to calculate the position of the north arrow.
y	numeric, position of the bottom left corner of the north arrow on the y axis. If not set, position is used to calculate the position of the north arrow.
colour	character, colour.
size	numeric, arrow size.
label_text	character, text below the north arrow.
label_margin	numeric, margin between label and north arrow as a proportion of the size of the north arrow.
label_size	numeric, label font size.
verbose	logical, if TRUE, messages and progress information are displayed on the console (default).

Details

For adding more sophisticated map features, I recommend ggspatial.

Value

A frames object of class moveVis.

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial frames_graph animate_frames

Examples

```
library(moveVis)
library(move2)
library(terra)

data("move_data", package = "moveVis")
r <- readRDS(example_data(file = "raster_NDVI.rds"))

# align movement
m <- align_move(move_data, res = units::set_units(4, "min"))

# create frames and add northarrow
frames <- frames_spatial(m, r, r_type = "gradient", fade_raster = TRUE) %>%
  add_northarrow()
frames[[100]]

# or in white at another position
frames <- frames_spatial(m, r, r_type = "gradient", fade_raster = TRUE) %>%
  add_northarrow(colour = "white", position = "bottomleft")
frames[[100]]
```

---

add_progress *Add progress bar to frames*

Description

This function adds a progress bar to frames created with `frames_spatial`.

Usage

```
add_progress(frames, colour = "grey", size = 1.8, y_nudge = 0, verbose = TRUE)
```

Arguments

`frames`	an object of class `moveVis` created with `frames_spatial`.
`colour`	character, progress bar colour.
`size`	numeric, progress bar line size (line width).
`y_nudge`	numeric, amount of vertical distance to move the progress bar (default: top of plot area).
`verbose`	logical, if `TRUE`, messages and progress information are displayed on the console (default).

Value

A frames object of class `moveVis`.

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial frames_graph animate_frames

Examples

```
library(moveVis)
library(move2)
library(terra)

data("move_data", package = "moveVis")
r <- readRDS(example_data(file = "raster_NDVI.rds"))

# align movement
m <- align_move(move_data, res = units::set_units(4, "min"))

# create frames and add northarrow
frames <- frames_spatial(m, r, r_type = "gradient", fade_raster = TRUE) %>%
  add_progress()
frames[[100]]

# or in white at another position
frames <- frames_spatial(m, r, r_type = "gradient", fade_raster = TRUE) %>%
  add_progress(colour = "red", size = 2.5)
frames[[100]]
```

---

add_scalebar *Add scalebar to frames*

Description

This function adds a scalebar to frames created with frames_spatial.

Usage

```
add_scalebar(
  frames,
  distance = NULL,
  height = 0.015,
  position = "bottomleft",
  x = NULL,
  y = NULL,
  colour = "black",
  label_margin = 1.2,
```

```
  units = "km",
  verbose = TRUE
)
```

Arguments

frames	an object of class moveVis created with frames_spatial.
distance	numeric, optional. Distance displayed by the scalebar (in either km or miles defined by argument units) By default, the displayed distance is calculated automatically.
height	numeric, height of the scalebar in a range from 0 to 1 as the proportion of the overall height of the frame map. Default is 0.015.
position	character, position of the scalebar on the map. Either "bottomleft", "upperleft", "upperright", "bottomright". Ignored, if x and y are set.
x	numeric, position of the bottom left corner of the scalebar on the x axis. If not set, position is used to calculate the position of the scalebar.
y	numeric, position of the bottom left corner of the scalebar on the y axis. If not set, position is used to calculate the position of the scalebar.
colour	character, colour of the distance labels. Default is "black".
label_margin	numeric, distance of the labels to the scalebar as a proportion of the height of the scalebar (e.g. if set to 2, the labels will be positioned with a distance to the scalebar of twice the scalebar height).
units	character, either "km" for kilometers or "miles" for miles.
verbose	logical, if TRUE, messages and progress information are displayed on the console (default).

Details

For adding more sophisticated map features, I recommend using ggspatial.

Value

A frames object of class moveVis.

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial frames_graph animate_frames

Examples

```
library(moveVis)
library(move2)
library(terra)

data("move_data", package = "moveVis")
r <- readRDS(example_data(file = "raster_NDVI.rds"))

# align movement
m <- align_move(move_data, res = units::set_units(4, "min"))

# create frames and add scaleebar
frames <- frames_spatial(m, r, r_type = "gradient", fade_raster = TRUE) %>%
  add_scalebar()
frames[[100]]

# or in white at another position
frames <- frames_spatial(m, r, r_type = "gradient", fade_raster = TRUE) %>%
  add_scalebar(colour = "white", position = "bottomright")
frames[[100]]

# or with another height
frames <- frames_spatial(m, r, r_type = "gradient", fade_raster = TRUE) %>%
  add_scalebar(colour = "white", position = "bottomright", height = 0.025)
frames[[100]]
```

---

add_text *Add text to frames*

Description

This function adds text to frames created with `frames_spatial`.

Usage

```
add_text(
  frames,
  labels,
  x,
  y,
  colour = "black",
  size = 3,
  type = "text",
  verbose = TRUE
)
```

Arguments

frames	an object of class moveVis created with frames_spatial.
labels	character, text to be added to frames. Either a single character value or a character vector of same length as frames.
x	numeric, position of text on the x scale. Either a single numeric value or a numeric vector of same length as frames.
y	numeric, position of text on the y scale. Either a single numeric value or a numeric vector of same length as frames.
colour	character, the text colour(s). Either a single character value or a character vector of same length as frames.
size	numeric, the text size(s). Either a single numeric value or a numeric vector of same length as frames.
type	character, either "text" to draw text or "label" to draw text inside a box.
verbose	logical, if TRUE, messages and progress information are displayed on the console (default).

Value

A frames object of class moveVis.

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial frames_graph animate_frames

Examples

```
library(moveVis)
library(move2)
library(terra)

data("move_data", package = "moveVis")
r <- readRDS(example_data(file = "raster_NDVI.rds"))

# align movement
m <- align_move(move_data, res = units::set_units(4, "min"))

# create frames and add text somewhere to all frames:
frames <- frames_spatial(m, r, r_type = "gradient", fade_raster = TRUE) %>%
  add_text("Water area", x = 8.959, y = 47.7305, colour = "white", size = 3)
frames[[100]]

# or use the ggplot2 "label" type:
frames <- frames_spatial(m, r, r_type = "gradient", fade_raster = TRUE) %>%
  add_text("Water area", x = 8.959, y = 47.7305, colour = "black", size = 3,
```

```
            type = "label")
frames[[100]]
```

`add_timestamps` *Add timestamps to frames*

Description

This function adds timestamps to frames created with `frames_spatial`.

Usage

```
add_timestamps(
  frames,
  x = NULL,
  y = NULL,
  format = "%Y-%m-%d %H:%M:%S",
  ...,
  verbose = TRUE
)
```

Arguments

`frames`	an object of class `moveVis` created with `frames_spatial`.
`x`	numeric, optional, position of timestamps on the x scale. By default, timestamps will be displayed in the top center.
`y`	numeric, optional, position of timestamps on the y scale.
`format`	character, optional, format of timestamps to be displayed in, passed to `strftime`).
`...`	optional, arguments passed to `add_text`, such as `colour`, `size`, `type`.
`verbose`	logical, if `TRUE`, messages and progress information are displayed on the console (default).

Value

A frames object of class `moveVis`.

Author(s)

Jakob Schwalb-Willmann

See Also

`frames_spatial` `frames_graph` `animate_frames`

Examples

```
library(moveVis)
library(move2)
library(terra)

data("move_data", package = "moveVis")
r <- readRDS(example_data(file = "raster_NDVI.rds"))

# align movement
m <- align_move(move_data, res = units::set_units(4, "min"))

# create frames and add timestamps as text
frames <- frames_spatial(m, r, r_type = "gradient", fade_raster = TRUE) %>%
  add_timestamps(type = "text")
frames[[100]]

# or use the ggplot2 "label" type:
frames <- frames_spatial(m, r, r_type = "gradient", fade_raster = TRUE) %>%
  add_timestamps(type = "label")
frames[[100]]
```

---

align_move *Align movement data*

Description

This function aligns movement data to a uniform time scale with a uniform temporal resolution throughout the complete movement sequence. This prepares the provided movement data to be usable by frames_spatial, which requires a uniform time scale and a consistent, unique temporal resolution for all supplied trajectories.

Usage

```
align_move(
  m,
  res = "minimum",
  start_end_time = NULL,
  fill_na_values = TRUE,
  ...,
  verbose = TRUE
)
```

Arguments

m	move2 object, which is allowed to contain irregular timestamps and diverging temporal resolutions.

res	either a units object representing the temporal resolution m should be aligned to, or a character being one of 'minimum', 'maximum', 'mean' or 'median' to indicate how the target resolution should be derived from m.
start_end_time	NULL (default) or a vector of two POSIXct times (one start time and one end time for alignment). If NULL, the start and end time are retrieved from m and used for alignment.
	• "minimum" to use the smallest temporal resolution of m (default)
	• "maximum" to use the largest temporal resolution of m
	• "mean" to use the rounded average temporal resolution of m
	• "median" to use the rounded median temporal resolution of m
fill_na_values	logical, whether to fill empty (NA) values of columns of m at interpolated locations (defaults to TRUE). Column values at interpolated locations are filled with the value of the temporally closest location.
...	deprecated arguments, including digit, unit and spaceMethod.
verbose	logical, if TRUE, messages and progress information are displayed on the console (default).

Value

move2 object, with aligned positions at uniform temporal scale computed from m, ready to be used by frames_spatial.

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial frames_graph

Examples

```
library(moveVis)
library(move2)
library(lubridate)

# example data
data("move_data")

# the tracks in move_data have irregular timestamps and sampling rates.
# print unique timestamps and time lags
unique(mt_time(move_data))
unique(mt_time_lags(move_data, units = "s"))

# use align_move to interpolate move_data to a uniform time scale and lag.
# e.g. setting a resolution of 4 minutes:
m <- align_move(m = move_data, res = units::set_units(4, "min"))
# check the result: records with attribute interpolate == TRUE have been added
# all trajectories are now aligend to a uniform 4-minute resolution:
```

```
unique(mt_time_lags(m, units = "min"))

# same with resolution of 1 hour:
m <- align_move(move_data, res = units::set_units(1, "hour"))
unique(mt_time_lags(m, units = "hour"))

# resolution of 15 seconds:
m <- align_move(move_data, res = units::set_units(15, "sec"))
unique(mt_time_lags(m, units = "sec"))

# you can set the start/end times if needed:
# first, let us retrieve the start and end times
start_end_time <- range(mt_time(m))
start_end_time

# I want the start time to be at 00 minutes and 00 seconds for the first track:
start_end_time <- round.POSIXt(start_end_time, units = "hours")

m <- align_move(
  move_data, res = units::set_units(4, "min"),
  start_end_time = start_end_time
)
mt_time(m)
```

`animate_frames` *Animate frames*

Description

`animate_frames` creates an animation from `moveVis` frames computed with `frames_spatial`, `frames_graph` or `join_frames`.

Usage

```
animate_frames(
  frames,
  out_file,
  fps = 25,
  width = 700,
  height = 700,
  res = 100,
  end_pause = 0,
  display = TRUE,
  overwrite = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

frames	an object of class moveVis created with frames_spatial.
out_file	character, the output file path, e.g. "/dir/to/file.mov". The file extension must correspond to a file format known by the available renderers of the running system. Use suggest_formats to get a vector of suggested known file formats.
fps	numeric, the number of frames to be displayed per second. Default is 2.
width	numeric, width of the output animation in pixels.
height	numeric, height of the output animation in pixels.
res	numeric, resolution of the output animation in ppi.
end_pause	numeric, defining how many seconds the last frame of the animation should be hold to add a pause at the the end of the animation. Default is 0 seconds to not add a pause.
display	logical, whether the animation should be displayed after rendering or not.
overwrite	logical, wether to overwrite an existing file, if out_file is already present.
verbose	logical, if TRUE, messages and progress information are displayed on the console (default).
...	additional arguments to be passed to the render function.

Details

An appropriate render function is selected depending on the file extension in out_file: For .gif files, gifski::save_gif is used, for any other (video) format, av::av_capture_graphics is used.

Value

None or the default image/video viewer displaying the animation

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial frames_graph join_frames

Examples

```
library(moveVis)
library(move2)
library(terra)

data("move_data", package = "moveVis")
r <- readRDS(example_data(file = "raster_NDVI.rds"))

# align movement
m <- align_move(move_data, res = units::set_units(4, "min"))
```

```
# create frames
frames <- frames_spatial(m, r, r_type = "gradient", fade_raster = TRUE) %>%
  add_colourscale(
    type = "gradient", colours = c("orange", "white", "darkgreen"),
    legend_title = "NDVI") %>%
  add_northarrow(position = "bottomleft") %>%
  add_scalebar(colour = "white", position = "bottomright") %>%
  add_progress() %>%
  add_timestamps(type = "label")

# check available formats
suggest_formats()

## Not run:
# animate frames as GIF
out_file <- tempfile(fileext = ".gif")
animate_frames(frames, out_file = out_file)
browseURL(out_file) # view animation

# animate frames as mov
out_file <- tempfile(fileext = ".mov")
animate_frames(frames, out_file = out_file)
browseURL(out_file) # view animation

## End(Not run)
```

`c.moveVis` *Combining moveVis frames*

Description

Method for combining multiple `moveVis` frames objects.

Usage

```
## S3 method for class 'moveVis'
c(...)
```

Arguments

`...`	two or more objects of class `moveVis`.

Value

A list of `moveVis` frames objects.

deprecated *Deprecated functions*

Description

Several functions have been deprecated over the development cycles of moveVis.

Usage

```
subset_move(...)

df2move(...)

animate_move(...)

animate_raster(...)

animate_stats(...)

get_formats(...)

get_libraries(...)
```

Arguments

...	deprecated arguments.

Note

To install older versions of moveVis, see https://github.com/16EAGLE/moveVis/releases/ or visit the CRAN archive.

See Also

frames_spatial frames_graph join_frames animate_frames

---

frames_graph *Create frames of movement-environment interaction graphs for animation*

Description

frames_graph creates a list of ggplot2 graphs displaying movement-environment interaction. Each object represents a single frame. Each frame can be viewed or modified individually. The returned list of frames can be animated using animate_frames.

Usage

```
frames_graph(
  m,
  r,
  r_type = "gradient",
  fade_raster = FALSE,
  crop_raster = TRUE,
  return_data = FALSE,
  graph_type = "flow",
  path_size = 1,
  path_colours = NA,
  path_legend = TRUE,
  path_legend_title = "Names",
  val_min = NULL,
  val_max = NULL,
  val_by = 0.1,
  ...,
  verbose = T
)
```

Arguments

m	move2 object of uniform time scale and time lag as returned by align_move. Can contain a column named colour to control path colours (see details below).
r	terra object, either a SpatRaster (mono-temporal) or a SpatRasterDataset (multi-temporal). In case of the latter, times of 'r' must be defined as 'POSIXct' (see time and details below).
r_type	character, either "gradient" or "discrete". Ignored, if r contains three bands, which are treated as RGB.
fade_raster	logical, if TRUE, r is interpolated over time. If FALSE, r elements are assigned to those frames closest to the equivalent times of r.
crop_raster	logical, whether to crop rasters in r to frame extents before plotting or not.
return_data	logical, if TRUE, instead of a list of frames, a data.frame containing the values extracted from r_list per individual, location and time is returned. This data.frame can be used to create your own multi- or mono-temporal ggplot2 movement-environemnt interaction graphs.
graph_type	character, defines the type of multi-temporal graph that should be drawn as frames. Currently supported graphs are: • "flow", a time flow graph with frame time on the x axis and values of the visited cell at x on the y axis per individual track • "hist", a cumulative histogram with cell values on the x axis and time-cumulative counts of visits on the y axis per individual track.
path_size	numeric, size of each path.
path_colours	character, a vector of colours. Must be of same length as number of individual tracks in m and refers to the order of tracks in m. If undefined (NA) and m contains a column named colour, colours provided within m are used (see details).

Othwersie, colours are selected from a standard rainbow palette per individual track.

`path_legend` logical, wether to add a path legend from `m` or not. Legend tracks and colours will be ordered by the tracks' temporal apperances, not by their order in `m`.

`path_legend_title` character, path legend title. Default is `"Names"`.

`val_min` numeric, minimum value of the value axis. If undefined, the minimum is collected automatically.

`val_max` numeric, maximum value of the value axis. If undefined, the maximum is collected automatically.

`val_by` numeric, increment of the value axis sequence. Default is 0.1. If `graph_type = "discrete"`, this value should be an integer of 1 or greater.

`...` additional arguments, currently unused.

`verbose` logical, if `TRUE`, messages and progress information are displayed on the console (default).

Details

To later on side-by-side join spatial frames created using `frames_spatial` with frames created with `frames_graph` for animation, equal inputs must have been used for both function calls for each of the arguments `m`, `r_list`, `r_times` and `fade_raster`.

If argument `path_colours` is not defined (set to `NA`), path colours can be defined by adding a character column named `colour` to `m`, containing a colour code or name per row (e.g. `"red"`. This way, for example, column `colour` for all rows belonging to individual A can be set to `"green"`, while column `colour` for all rows belonging to individual B can be set to `"red"`. Colours could also be arranged to change through time or by behavioral segments, geographic locations, age, environmental or health parameters etc. If a column name `colour` in `m` is missing, colours will be selected automatically. Call `colours()` to see all available colours in R.

Value

An object of class `moveVis`. If `return_data` is `TRUE`, a `data.frame` is returned (see `return_data`).

Author(s)

Jakob Schwalb-Willmann

See Also

`frames_spatial` `join_frames` `animate_frames`

Examples

```
library(moveVis)
library(move2)
library(terra)

data("move_data", package = "moveVis")
```

```
r <- readRDS(example_data(file = "raster_NDVI.rds"))

# align movement
m <- align_move(move_data, res = units::set_units(4, "min"))

# create frames
frames <- frames_graph(
  m, r, r_type = "gradient", fade_raster = TRUE, graph_type = "flow"
)
# take a look
frames[[100]]

# make a histogram graph:
frames <- frames_graph(
  m, r, r_type = "gradient", fade_raster = TRUE, graph_type = "hist"
)
frames[[100]]

# change the value interval:
frames <- frames_graph(
  m, r, r_type = "gradient", fade_raster = TRUE, graph_type = "hist",
  val_by = 0.01
)
frames[[100]]

# manipulate the labels, as they are quite dense by replacing the scale
frames <- add_gg(frames, rlang::expr(
  scale_x_continuous(breaks=seq(0,1,0.1), labels=seq(0,1,0.1), expand = c(0,0)))
)
frames[[100]]

# to make your own graphs, use frames_graph to return data instead of frames
graph_data <- frames_graph(
  m, r, r_type = "gradient", fade_raster = TRUE, return_data = TRUE
)
```

`frames_spatial` *Create frames of spatial movement maps for animation*

Description

`frames_spatial` creates frames from movement and map/raster data. If no custom raster data is provided, a basemap is pulled from a map tile service using the `basemaps` package. Frames are returned as an object of class `moveVis` and can be subsetted, viewed (see `render_frame`), modified (see `add_gg` and associated functions) and animated (see `animate_frames`).

Usage

```
frames_spatial(
  m,
```

```
  r = NULL,
  r_type = "gradient",
  fade_raster = FALSE,
  crop_raster = TRUE,
  map_service = "osm",
  map_type = "streets",
  map_res = 1,
  map_token = NULL,
  map_dir = NULL,
  margin_factor = 1.1,
  equidistant = NULL,
  ext = NULL,
  crs = if (is.null(r)) st_crs(3857) else st_crs(terra::crs(r)),
  crs_graticule = st_crs(4326),
  path_size = 3,
  path_end = "round",
  path_join = "round",
  path_mitre = 10,
  path_arrow = NULL,
  path_colours = NA,
  path_alpha = 1,
  path_fade = FALSE,
  path_legend = TRUE,
  path_legend_title = "Names",
  tail_length = 19,
  tail_size = 1,
  tail_colour = "white",
  trace_show = FALSE,
  trace_size = tail_size,
  trace_colour = "white",
  cross_dateline = FALSE,
  ...,
  verbose = TRUE
)
```

Arguments

m	move2 object of uniform time scale and time lag as returned by align_move. Can contain a column named colour to control path colours (see details below).
r	terra object, either a SpatRaster (mono-temporal) or a SpatRasterDataset (multi-temporal). In case of the latter, times of 'r' must be defined as 'POSIXct' (see time and details below).
r_type	character, either "gradient" or "discrete". Ignored, if r contains three bands, which are treated as RGB.
fade_raster	logical, if TRUE, r is interpolated over time. If FALSE, r elements are assigned to those frames closest to the equivalent times of r.
crop_raster	logical, whether to crop rasters in r to frame extents before plotting or not.

map_service	character, a map service, e.g. "osm". Use get_maptypes for a list of available map services and types..
map_type	character, a map type, e.g. "streets". Use get_maptypes for available map services and types.
map_res	numeric, resolution of base map in range from 0 to 1.
map_token	character, mapbox authentification token for mapbox basemaps. Register at https://www.mapbox.com/ to get a mapbox token. Mapbox is free of charge after registration for up to 50.000 map requests per month. Ignored, if map_service = "osm".
map_dir	character, directory where downloaded basemap tiles can be stored. By default, a temporary directory is used. If you use moveVis often for the same area it is recommended to set this argument to a directory persistent throughout sessions (e.g. in your user folder), so that baesmap tiles that had been already downloaded by moveVis do not have to be requested again.
margin_factor	numeric, factor relative to the extent of m by which the frame extent should be increased around the movement area. Ignored, if ext is set.
equidistant	logical, whether to make the map extent equidistant (squared) with y and x axis measuring equal distances or not. Especially in polar regions of the globe it might be necessaray to set equidistant to FALSE to avoid strong stretches. By default (equidistant = NULL), equidistant is set automatically to FALSE, if ext is set, otherwise TRUE. Read more in the details.
ext	sf bbox in same CRS as m, optional. If set, frames are cropped to this extent. If not set, the extent is computed from m, optionally with a margin set by margin_factor.
crs	sf crs object. This is the CRS that is used for visualizing both movement and map data. Defaults to st_crs(3857) (Web Mercator), unless r is defined. In that case, st_crs(r) is used.
crs_graticule	sf crs object. This is the CRS that should be used to generate graticules. By default, geographic coordinates (Lon/Lat WGS84, EPSG:4326) is used.
path_size	numeric, size of each path.
path_end	character, either "round", "butt" or "square", indicating the path end style.
path_join	character, either "round", "mitre" or "bevel", indicating the path join style.
path_mitre	numeric, path mitre limit (number greater than 1).
path_arrow	arrow, path arrow specification, as created by grid::arrow().
path_colours	character, a vector of colours. Must be of same length as number of individual tracks in m and refers to the order of tracks in m. If undefined (NA) and m contains a column named colour, colours provided within m are used (see details). Othwersie, colours are selected from a standard rainbow palette per individual track.
path_alpha	numeric, defines alpha (transparency) of the path. Value between 0 and 1. Default is 1.
path_fade	logical, whether paths should be faded towards the last frame or not. Useful, if trace_show = TRUE and you want to hold the last frame using end_pause in animate_frames.

path_legend — logical, wether to add a path legend from m or not. Legend tracks and colours will be ordered by the tracks' temporal apperances, not by their order in m.

path_legend_title — character, path legend title. Default is "Names".

tail_length — numeric, length of tail per movement path.

tail_size — numeric, size of the last tail element. Default is 1.

tail_colour — character, colour of the last tail element, to which the path colour is faded. Default is "white".

trace_show — logical, whether to show the trace of the complete path or not.

trace_size — numeric, size of the trace. Default is same as tail_size.

trace_colour — character, colour of the trace. Default is "white". It is recommended to define the same colours for both trace_colour and tail_colour to enforce an uninterrupted colour transition form the tail to the trace.

cross_dateline — logical, whether tracks are crossing the dateline (longitude 180/-180) or not. If TRUE, frames are expanded towards the side of the dateline that is smaller in space. Applies only if the CRS of m is not projected (geographical, lon/lat). If FALSE (default), frames are clipped at the minimum and maximum longitudes and tracks cannot cross.

... — Additional arguments customizing the frame background:

- alpha, numeric, background transparency (0-1).
- maxpixels, maximum number of pixels to be plotted per frame. Defaults to 500000. Reduce to decrease detail and increase rendering speeds.
- maxColorValue, numeric, only relevant for RGB backgrounds (i.e. if r_type = "RGB" or if a default base map is used). Maximum colour value (e.g. 255). Defaults to maximum raster value.
- interpolate, logical, whether to spatially smooth the map raster (default is FALSE).

verbose — logical, if TRUE, messages and progress information are displayed on the console (default).

Details

If argument path_colours is not defined (set to NA), path colours can be defined by adding a character column named colour to m, containing a colour code or name per row (e.g. "red". This way, for example, column colour for all rows belonging to individual A can be set to "green", while column colour for all rows belonging to individual B can be set to "red". Colours could also be arranged to change through time or by behavioural segments, geographic locations, age, environmental or health parameters etc. If a column name colour in m is missing, colours will be selected using path_colours or automatically. Call colours() to see all available colours in R.

Basemap colour scales can be changed/added using add_colourscale or by using ggplot2 commands (see examples). For continuous scales, use r_type = "gradient". For discrete scales, use r_type = "discrete".

If argument equidistant is set, the map extent is calculated (thus enlarged into one axis direction) to represent equal distances on the x and y axis.

Value

A frames object of class moveVis.

Note

The use of some map services, e.g. "osm_stadia", "osm_thunderforest" or "mapbox", require registration to obtain an API token/key which can be supplied to map_token. See get_maptypes for details.

The Coordinate Reference System defined by argument crs is treated as target projection. Both m, basemaps accessed through a map service, or custom imagery provided through argument r will be reprojected into crs. The default CRS is Web Mercator (EPSG 3857), which guarantees undistorted map labels in web basemap imagery.

Author(s)

Jakob Schwalb-Willmann

See Also

frames_graph join_frames animate_frames

Examples

```
library(moveVis)
library(move2)
library(terra)

# Example using multi-temporal raster data as basemap
data("move_data")

# align movement
m <- align_move(move_data, res = units::set_units(4, "min"))

# get available map types
get_maptypes()

# with osm topographic base map
## Not run:
frames <- frames_spatial(
  m, map_service = "osm", map_type = "topographic",
  alpha = 0.5
)
# take a look at one of the frames, e.g. the 100th
frames[[100]]

frames <- frames %>%
  add_northarrow(position = "bottomleft") %>%
  add_scalebar(colour = "black", position = "bottomright") %>%
  add_progress() %>%
  add_timestamps(type = "label")
```

```
frames[[100]]

# animate frames as GIF
out_file <- tempfile(fileext = ".gif")
animate_frames(frames, out_file = out_file)
browseURL(out_file) # view animation

# use a larger margin around extent
frames <- frames_spatial(
  m, map_service = "osm", map_type = "topographic", alpha = 0.5,
  margin_factor = 1.8
)
frames[[100]] # take a look

# use a extent object as your AOI
ext <- sf::st_bbox(move_data)
ext[["xmin"]] <- ext[["xmin"]] - (ext[["xmin"]]*0.03)
ext[["xmax"]] <- ext[["xmax"]] + (ext[["xmax"]]*0.03)

frames <- frames_spatial(
  m, map_service = "osm", map_type = "topographic", alpha = 0.5,
  ext = ext
)
frames[[100]]

# alter path appearance (make it longer and bigger)
frames <- frames_spatial(
  m, map_service = "osm", map_type = "topographic", alpha = 0.5,
  path_size = 4, tail_length = 29
)
frames[[100]]

# adjust path colours manually
frames <- frames_spatial(
  m, map_service = "osm", map_type = "topographic", alpha = 0.5,
  path_colours = c("black", "blue", "purple")
)
frames[[100]]

m$colour <- plyr::mapvalues(
  as.character(mt_track_id(m)),
  unique(mt_track_id(m)), c("orange", "purple", "darkgreen")
)

frames <- frames_spatial(
  m, map_service = "osm", map_type = "topographic", alpha = 0.5
)
frames[[100]]


## End(Not run)
# create frames from custom (multi-temporal) basemaps
r <- readRDS(example_data(file = "raster_NDVI.rds"))
```

```
# timestamps of each raster are stored in the SpatRasterDataset:
time(r)

# create frames
frames <- frames_spatial(
  m, r = r, r_type = "gradient",
  fade_raster = TRUE
)

# customize
frames <- frames %>%
  add_colourscale(
    type = "gradient", colours = c("orange", "white", "darkgreen"),
    legend_title = "NDVI") %>%
  add_northarrow(position = "bottomleft") %>%
  add_scalebar(colour = "white", position = "bottomright") %>%
  add_progress() %>%
  add_timestamps(type = "label")

# render a single frame
frames[[100]]

# check available animation file formats
suggest_formats()

## Not run:
# animate frames as GIF
out_file <- tempfile(fileext = ".gif")
animate_frames(frames, out_file = out_file)
browseURL(out_file) # view animation

# animate frames as mov
out_file <- tempfile(fileext = ".mov")
animate_frames(frames, out_file = out_file)
browseURL(out_file) # view animation

## End(Not run)
```

`get_frametimes` *Get frame times from frames*

Description

This function extracts the timestamps associated with each frame from a `moveVis` object created using `frames_spatial` or `frames_graph` and returns them as a vector.

Usage

```
get_frametimes(frames)
```

Arguments

frames	an object of class moveVis created with frames_spatial.

Value

A POSIXct vector of timestamps representing the time associated with each frame.

See Also

frames_spatial frames_graph

Examples

```
library(moveVis)
library(move2)
library(terra)

data("move_data", package = "moveVis")
r <- readRDS(example_data(file = "raster_NDVI.rds"))

# align movement
m <- align_move(move_data, res = units::set_units(4, "min"))

# create frames
frames <- frames_spatial(m, r, r_type = "gradient", fade_raster = TRUE)

# get frametimes
get_frametimes(frames)
```

---

join_frames *Join multiple frames side-by-side*

Description

This function joins two or more moveVis frames of equal lengths side-by-side into a single plot per frame using wrap_plots. This is useful if you want to side-by-side combine spatial frames returned by frames_spatial with graph frames returned by frames_graph.

Usage

```
join_frames(
  frames_list,
  guides = "collect",
  design = NULL,
  render_all_legends = FALSE,
  ...,
  frames_lists = NULL,
  verbose = T
)
```

Arguments

frames_list	list, a list of two or more moveVis frame objects that you want to combine into onw. Must be of equal lengths. Frames are being passed to wrap_plots and combined frame-by-frame.
guides	character, controls how to treat the scales/legends of both frames. See wrap_plots for details. Defaults to 'collect'.
design	character, controls how frames are arranged. See wrap_plots for details. By default, frames are joined sidy-by-side horizontally.
render_all_legends	logical, whether legends should be preserved (TRUE) or only the legend of the first frames object should be rendered (FALSE, default).
...	Further arguments, specifying the appearance of the joined ggplot2 objects, passed to wrap_plots. See wrap_plots for further options.
frames_lists	deprecated
verbose	logical, if TRUE, messages and progress information are displayed on the console (default).

Value

A frames object of class moveVis.

See Also

frames_spatial frames_graph animate_frames

Examples

```
library(moveVis)
library(move2)
library(terra)

data("move_data", package = "moveVis")
r <- readRDS(example_data(file = "raster_NDVI.rds"))

# align movement
m <- align_move(move_data, res = units::set_units(4, "min"))

# create spatial frames
frames_sp <- frames_spatial(m, r, r_type = "gradient", fade_raster = TRUE) %>%
  add_colourscale(
    type = "gradient", colours = c("orange", "white", "darkgreen"),
    legend_title = "NDVI"
  )

# create graph frames
frames_flow <- frames_graph(m, r, path_legend = FALSE, graph_type = "flow")
frames_hist <- frames_graph(m, r, path_legend = FALSE, graph_type = "hist")

# check lengths (must be equal)
```

```
sapply(list(frames_sp, frames_flow, frames_hist), length)

# define a patchwork design (see ?wrap_plots)
design <- "
AAB
AAC
"

# create joined frames
frames_joined <- join_frames(
  list(frames_sp, frames_flow, frames_hist), design = design
)
frames_joined[[100]]

# adjust width and height to suite your joined frames
## Not run:
out_file <- tempfile(fileext = ".gif")
animate_frames(
  frames_joined, out_file = out_file, fps = 25, width = 900, height = 500,
  res = 90, display = TRUE, overwrite = TRUE
)

## End(Not run)
```

`length.moveVis` *Length of moveVis frames*

Description

Method to get length of `moveVis` frames, i.e. number of frames.

Usage

```
## S3 method for class 'moveVis'
length(x)
```

Arguments

`x`	an object of class `moveVis`.

Value

Numeric

move_data *Example simulated movement tracks*

Description

This dataset contains a move2 object, representing coordinates and acquisition times of three simulated movement tracks, covering a location nearby Lake of Constance, Germany. Individual names are made up for demonstration purposes.

Usage

```
data(move_data)
```

Format

move2 object, as used by the move2 package.

Details

This object is used by some moveVis examples and unit tests.

Note

All data contained should only be used for testing moveVis and are not suitable to be used for analysis or interpretation.

print.moveVis *Print moveVis frames*

Description

Method for printing moveVis frames. Prints show basic information about the object, including number of frames, extent and more.

Usage

```
## S3 method for class 'moveVis'
print(x, ...)
```

Arguments

x	an object of class moveVis.
...	further arguments passed to or from other methods.

Value

Invisible, used for its side effect.

raster_data *Example raster data*

Description

This function provides two example raster datasets as SpatRasterDataset, covering the Lake of Constance area, as well as invented dates and times that simulate acquisition times.

Usage

```
example_data(file = c("raster_NDVI.rds", "raster_classification.rds"))
```

Arguments

file	character, file name of the example dataset.

Details

The returned SpatRasterDataset objects are used as example data by some moveVis code examples and unit tests.

The two available datasets are:

- raster_NDVI.rds representing simulated NDVI images (continuous values)
- raster_classification.rds, representing an NDVI-derived classification (discrete values). NDVI values from -1 to 0 translate into class 1, from 0 to 0.2 into class 2, from 0.2 to 0.4 into class 3, from 0.4 to 0.8 into class 4 and from 0.8 to 1 into class 5.

Acquisition times can be retrieved using terra::time(readRDS(example_data()))

Value

A SpatRasterDataset dataset of either continuous or discrete values, including acquisition times as POSIXct objects.

Note

All data contained should only be used for testing moveVis and are not suitable to be used for analysis or interpretation.

Source

Simulated based on MODIS (MOD13Q1 NDVI)

rev.moveVis *Reverse moveVis frames*

Description

Method for reversing the order of frames in a `moveVis` frames object.

Usage

```
## S3 method for class 'moveVis'
rev(x)
```

Arguments

`x`	an object of class `moveVis`.

Value

A `moveVis` frames object.

settings *moveVis settings*

Description

These functions control session-wide settings that can increase processing speeds.

Usage

```
use_multicore(n_cores = NULL, verbose = TRUE)

use_disk(
  frames_to_disk = TRUE,
  dir_frames = paste0(tempdir(), "/moveVis"),
  n_memory_frames = NULL,
  verbose = TRUE
)
```

Arguments

`n_cores`	numeric, optional, number of cores to be used. If not defined, the number of cores will be detected automatically (`n-1` cores will be used with `n` being the number of cores available).
`verbose`	logical, if `TRUE`, messages and progress information are displayed on the console (default).

`frames_to_disk` logical, whether to use disk space for creating frames or not. If `TRUE`, frames will be written to `dir_frames`, clearing memory.

`dir_frames` character, directory where to save frame during frames creating.

`n_memory_frames` numeric, maximum number of frames allowed to be hold in memory. This number defines after how many frames memory should be cleared by writing frames in memory to disk.

Details

`use_multicore` enables multi-core usage of `moveVis` by setting the maximum number of cores to be used. This can strongly increase the speed of creating frames.

`use_disk` enables the usage of disk space for creating frames. This can prevent memory overload when creating frames for very large animations.

For most tasks, `moveVis` is able to use multiple cores to increase computational times through parallelization. By default, multi-core usage is disabled. This function saves the number of cores that `moveVis` should use to the global option `"moveVis.n_cores"` that can be printed using `getOption("moveVis.n_cores")`.

How much memory is needed to create frames depends on the frame resolution (number of pixels) and the number of frames. Depending on how much memory is available it can make sense to allow disk usage and set a maximum number of frames to be hold in memory that won't fill up the available memory completely.

`moveVis` uses the `parallel` package for parallelization.

Value

None. These functions are used for their side effects.

Examples

```
library(moveVis)

# enable multi-core usage automatically
use_multicore()

# define number of cores manually
use_multicore(n_cores = 2)

# allow disk use with default directory
# and maximum of 50 frames in memory
use_disk(frames_to_disk = TRUE, n_memory_frames = 50)
```

suggest_formats *Suggest known file formats*

Description

This function returns a selection of suggested file formats that can be used with `out_file` of `animate_frames` on your system.

Usage

```
suggest_formats(
  suggested = c("gif", "mov", "mp4", "flv", "avi", "mpeg", "3gp", "ogg")
)
```

Arguments

`suggested`	character, a vector of suggested file formats which are checked to be known by the available renderers on the running system. By default, these are `c("gif", "mov", "mp4", "flv", "avi", "mpeg", "3gp", "ogg")`.

Value

A subset of `suggested`, containing only those file formats which are known by the renderers on the running system.

See Also

`animate_frames`

Examples

```
library(moveVis)

# find out which formats are available
suggest_formats()

# check for a particular format not listed in "suggested" that you want to use, e.g. m4v
suggest_formats("m4v")
# if "m4v" is returned, you can use this format with animate_frames
```

tail.moveVis *Return first or last frames of an moveVis frames object*

Description

Method for returning n last or first frames of a moveVis frames objects.

Usage

```
## S3 method for class 'moveVis'
tail(x, n = 6L, ...)

## S3 method for class 'moveVis'
head(x, n = 6L, ...)
```

Arguments

x	an object of class moveVis.
n	an integer of length up to length(x).
...	further arguments passed to or from other methods.

Value

A moveVis frames object.

view_spatial *View movements on an interactive map*

Description

view_spatial is a simple wrapper that displays movement tracks on an interactive mapview or leaflet map.

Usage

```
view_spatial(
  m,
  render_as = "mapview",
  time_labels = TRUE,
  stroke = TRUE,
  path_colours = NA,
  path_legend = TRUE,
  path_legend_title = "Names",
  verbose = TRUE
)
```

Arguments

m	move2 object. Can contain a column named colour to control path colours (see details).
render_as	character, either 'mapview' to return a mapview map or 'leaflet' to return a leaflet map.
time_labels	logical, whether to display timestamps for each track fix when hovering it with the mouse cursor.
stroke	logical, whether to draw stroke around circles.
path_colours	character, a vector of colours. Must be of same length as number of individual tracks in m and refers to the order of tracks in m. If undefined (NA) and m contains a column named colour, colours provided within m are used (see details). Othwersie, colours are selected from a standard rainbow palette per individual track.
path_legend	logical, wether to add a path legend from m or not. Legend tracks and colours will be ordered by the tracks' temporal apperances, not by their order in m.
path_legend_title	character, path legend title. Default is "Names".
verbose	logical, if TRUE, messages and progress information are displayed on the console (default).

Details

If argument path_colours is not defined (set to NA), path colours can be defined by adding a character column named colour to m, containing a colour code or name per row (e.g. "red". This way, for example, column colour for all rows belonging to individual A can be set to "green", while column colour for all rows belonging to individual B can be set to "red". Colours could also be arranged to change through time or by behavioral segments, geographic locations, age, environmental or health parameters etc. If a column name colour in m is missing, colours will be selected automatically. Call colours() to see all available colours in R.

Value

An interactive mapview or leaflet map.

Author(s)

Jakob Schwalb-Willmann

See Also

frames_spatial

Examples

```
## Not run:
library(moveVis)
library(move2)
```

```
data("move_data", package = "moveVis")

view_spatial(move_data)

# return a leaflet map (leaflet must be installed)
view_spatial(move_data, render_as = "leaflet")

# turn off time labels and legend
view_spatial(move_data, time_labels = FALSE, path_legend = FALSE)

## End(Not run)
```

`whitestork_data` *White Stork LifeTrack tracks*

Description

This dataset contains a `data.frame` object, representing coordinates and acquisition times of 15 White Storks, migrating from Lake of Constance, SW Germany, to Africa.

Usage

```
data(whitestork_data)
```

Format

- `df` is a `data.frame` object
- `m` is a `moveStack` object

An object of class `move2` (inherits from `sf`, `data.frame`) with 155173 rows and 3 columns.

Details

These objects are used by some `moveVis` examples and have been included for demonstrational purposes.

The dataset represents a subset of the LifeTrack White Stork dataset by Cheng et al. (2019) and Fiedler et al. (2019), available under the Creative Commons license "CC0 1.0 Universal Public Domain Dedication" on Movebank (doi:10.5441/001/1.ck04mn78/1).

References

Cheng Y, Fiedler W, Wikelski M, Flack A (2019) "Closer-to-home" strategy benefits juvenile survival in a long-distance migratory bird. Ecology and Evolution. doi:10.1002/ece3.5395

Fiedler W, Flack A, Schäfle W, Keeves B, Quetting M, Eid B, Schmid H, Wikelski M (2019) Data from: Study "LifeTrack White Stork SW Germany" (2013-2019). Movebank Data Repository. doi:10.5441/001/1.ck04mn78

Index

∗ **datasets**
    move_data, 39
    whitestork_data, 46
[.moveVis, *4*, 5
[[.moveVis, *4*, 5

add_colourscale, *3*, 6, *32*
add_gg, *3*, *7*, 8, *29*
add_labels, *3*, 12
add_northarrow, *3*, 13
add_progress, *3*, 15
add_scalebar, *3*, 16
add_text, *3*, 18, *20*
add_timestamps, *3*, 20
align_move, *3*, 21, *27*, *30*
animate_frames, *3*, *7*, *9*, *13, 14*, *16, 17*, *19, 20*, 23, *26*, *28, 29*, *31*, *33*, *37*, *43*
animate_move *(*deprecated*)*, 26
animate_raster *(*deprecated*)*, 26
animate_stats *(*deprecated*)*, 26

c, *4*
c.moveVis, 25

deprecated, 26
df *(*whitestork_data*)*, 46
df2move *(*deprecated*)*, 26

example_data, *4*
example_data *(*raster_data*)*, 40
expr, *9*

frames_graph, *3*, *7–9*, *13, 14*, *16, 17*, *19, 20*, *22–24*, 26, *26*, *28*, *33*, *35–37*
frames_spatial, *3*, *6–9*, *12–24*, *26*, *28*, 29, *35–37*, *45*

get_formats *(*deprecated*)*, 26
get_frametimes, *3*, 35
get_libraries *(*deprecated*)*, 26
get_maptypes, *3*, *31*, *33*

head, *4*
head.moveVis *(*tail.moveVis*)*, 44

join_frames, *3*, *23, 24*, *26*, *28*, *33*, 36

length, *4*
length.moveVis, 38

m *(*whitestork_data*)*, 46
move_data, *4*, 39
moveVis, *3*
moveVis *(*moveVis-package*)*, 2
moveVis-package, 2

print, *4*
print.moveVis, 39

raster_data, 40
render_frame, *3, 4*, *29*
render_frame *(*[[.moveVis*)*, 5
rev, *4*
rev.moveVis, 41

settings, 41
strftime, *20*
subset_move *(*deprecated*)*, 26
suggest_formats, *3*, *24*, 43

tail, *4*
tail.moveVis, 44
time, *27*, *30*

use_disk, *4*
use_disk *(*settings*)*, 41
use_multicore, *4*
use_multicore *(*settings*)*, 41

view_spatial, *4*, 44

waiver, *12*
whitestork_data, *4*, 46
wrap_plots, *3*, *36, 37*