Package 'gganimate'

June 21, 2025

Type Package

Title A Grammar of Animated Graphics

Version 1.0.10

Maintainer Thomas Lin Pedersen <thomasp85@gmail.com>

Description The grammar of graphics as implemented in the 'ggplot2' package has been successful in providing a powerful API for creating static visualisation. In order to extend the API for animated graphics this package provides a completely new set of grammar, fully compatible with 'ggplot2' for specifying transitions and animations in a flexible and extensible way.

License MIT + file LICENSE

URL https://gganimate.com, https://github.com/thomasp85/gganimate

BugReports https://github.com/thomasp85/gganimate/issues

Depends ggplot2 (>= 3.5.2)

Imports cli, glue, grDevices, grid, lifecycle, progress, rlang, scales, stringi, transformr (>= 0.1.5), tweenr (>= 2.0.3), utils, vctrs

Suggests av, base64enc, covr, gifski (>= 1.4.3), htmltools, knitr, magick, ragg, rmarkdown, sf, svglite, testthat

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.2

Collate 'aaa.R' 'anim_save.R' 'animate.R' 'animation_store.R' 'ease-aes.R' 'gganim.R' 'gganimate-ggproto.R' 'gganimate-package.r' 'ggplot2_reimpl.R' 'group_column.R' 'import-standalone-obj-type.R' 'import-standalone-types-check.R' 'layer_type.R' 'match_shapes.R' 'plot-build.R' 'post_process.R' 'renderers.R' 'scene.R' 'shadow-.R' 'shadow-mark.R' 'shadow-null.R' 'shadow-trail.R' 'shadow-wake.R' 'transformr.R' 'transition-.R' 'transition-components.R' 'transition-events.R'

'transition-filter.R' 'transition-manual.R'
'transition-layers.R' 'transition-null.R' 'transition-states.R'
'transition-time.R' 'transition_reveal.R' 'transmute-.R'
'transmute-enter.R' 'transmute-exit.R' 'transmuters.R'
'tween_before_stat.R' 'view-.R' 'view-follow.R' 'view-static.R'
'view-step.R' 'view-step-manual.R' 'view-zoom.R'
'view-zoom-manual.R' 'zzz.R'

NeedsCompilation no

Author Thomas Lin Pedersen [aut, cre] (ORCID:
<https://orcid.org/0000-0002-5147-4711>),
David Robinson [aut],
Posit, PBC [cph, fnd]

Repository CRAN

Date/Publication 2025-06-21 13:40:02 UTC

Contents

`animate` *Render a gganim object*

Description

This function takes a gganim object and renders it into an animation. The nature of the animation is dependent on the renderer, but defaults to using `gifski` to render it to a gif. The length and framerate is decided on render time and can be any two combination of `nframes`, `fps`, and `duration`. Rendering is happening in discrete time units. This means that any event in the animation is rounded of to the nearest frame (e.g. entering will always take a whole number of frames). This means that rounding artifacts are possible when only rendering few frames. To avoid this you can increase the `detail` argument. `detail` will get multiplied to `nframes` and the resulting number of frames will get calculated, but only `nframes` evenly spaced frames are rendered.

Usage

```
animate(plot, ...)

## S3 method for class 'gganim'
animate(
  plot,
  nframes,
  fps,
  duration,
  detail,
  renderer,
  device,
  ref_frame,
  start_pause,
  end_pause,
  rewind,
  ...
)

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

knit_print.gganim(x, options, ...)
```

Arguments

`plot`, `x`	A `gganim` object
`...`	Arguments passed on to the device. For available device arguments, see `grDevices::png()` or `grDevices::svg()`
`nframes`	The number of frames to render (default `100`)
`fps`	The framerate of the animation in frames/sec (default `10`)

duration	The length of the animation in seconds (unset by default)
detail	The number of additional frames to calculate, per frame (default 1)
renderer	The function used to render the generated frames into an animation. Gets a vector of paths to images along with the framerate. (by default it will use gifski_renderer() if gifski is installed. If not it will use magick_renderer() if magick is installed and then av_renderer() if av is installed. If all fails it will use the file_renderer())
device	The device to use for rendering the single frames. Possible values are 'png', 'ragg_png' (requires the ragg package), 'jpeg', 'tiff', 'bmp', 'svg', and 'svglite' (requires the svglite package). (default 'png')
ref_frame	The frame to use for fixing dimensions of the plot, e.g. the space available for axis text. Defaults to the first frame. Negative values counts backwards (-1 is the last frame) (default 1)
start_pause, end_pause	Number of times to repeat the first and last frame in the animation (default is 0 for both)
rewind	Should the animation roll back in the end (default FALSE)
options	chunk options for the currently executing chunk

Details

print.gganim() is an alias for animate() in the same way as print.ggplot() is an alias for plot.ggplot(). This ensures that gganimate behaves ggplot2-like and produces the animation when the object is printed. The plot() method is different and produces a single frame for inspection (by default frame 50 out of 100).

Animations can be saved to disk using anim_save() in much the same way ggsave() works for static plots.

Value

The return value of the renderer function

Defaults

It is possible to overwrite the defaults used by gganimate for the animation by setting them with options() (prefixed with gganimate.. As an example, if you would like to change the default nframes to 50 you would call options(gganimate.nframes = 50). In order to set default device arguments (those you would normally pass through with ...) you should use the gganimate.dev_args options and provide a list of arguments e.g. options(gganimate.dev_args = list(width = 800, height = 600)) Defaults set this way can still be overridden by giving arguments directly to animate().

knitr Support:
It is possible to specify the arguments to animate() in the chunk options when using gganimate with knitr. Arguments specified in this way will have precedence over defaults, but not over arguments specified directly in animate(). The arguments should be provided as a list to the gganimate chunk option, e.g. {r, gganimate = list(nframes = 50, fps = 20)}. A few build-in knitr options have relevance for animation and will be used unless given specifically in the gganimate list option. The native knitr options supported are:

- dev: will set device
- dev.args: will set additional arguments to the device (...)
- fig.width, fig.height, fig.asp, fig.dim: will set width and height of the device.

Label variables

All plots have a certain set of variables available for string literal interpolation within plot labels. These are:

- **frame** gives you the frame index for the current frame
- **nframes** gives you the total number of frames in the animation
- **progress** gives you the progress of the animation at the current frame (equal to frame/nframes)
- **data** gives you the layer data for the current frame (as a list of data frames)

Further, the transition and view in use can also make variables available. Consult the documentation for these for more detail.

Examples

```
anim <- ggplot(mtcars, aes(mpg, disp)) +
  geom_point(aes(color = gear)) +
  transition_states(gear, transition_length = 2, state_length = 1) +
  enter_fade() +
  exit_fade()

## Not run:
# Explicitly animate using default (same as just printing the animation)
animate(anim)

# Change duration and framerate
animate(anim, fps = 20, duration = 15)

# Make the animation pause at the end and then rewind
animate(anim, nframes = 100, end_pause = 10, rewind = TRUE)

# Use a different renderer
animate(anim, renderer = file_renderer('~/animation/'))[1:6]

# Specify device dimensions and/or resolution
animate(anim, height = 2, width = 3, units = "in", res = 150)

## End(Not run)
```

anim_save *Save an animation to a file*

Description

This function is analogous to ggplot2::ggsave() in that it by default takes the last created animation and saves it to the specific location. As gganimate supports arbitrary renderers, and thus return types, the returned object must implement a save_animation method to be able to be used with anim_save(). This is provided natively for gif_image and magick-image objects.

Usage

```
anim_save(filename, animation = last_animation(), path = NULL, ...)
```

Arguments

filename	File name to create on disk.
animation	The animation object to save, as returned by animate(). Defaults to the last rendered animation using last_animation()
path	Path of the directory to save plot to: path and filename are combined to create the fully qualified file name. Defaults to the working directory.
...	arguments passed on to animate() if animation is a gganim object

ease_aes *Control easing of aesthetics*

Description

Easing defines how a value change to another during tweening. Will it progress linearly, or maybe start slowly and then build up momentum. In gganimate, each aesthetic or computed variable can be tweened with individual easing functions using the ease_aes() function. All easing functions implemented in tweenr are available, see tweenr::display_ease for an overview. Setting an ease for x and/or y will also affect the other related positional aesthetics (e.g. xmin, yend, etc).

Usage

```
ease_aes(default = "linear", ...)
```

Arguments

default	The default easing function to use (defaults to 'linear')
...	Override easing for specific aesthetics

Easing functions

- **quadratic** Models a power-of-2 function
- **cubic** Models a power-of-3 function
- **quartic** Models a power-of-4 function
- **quintic** Models a power-of-5 function
- **sine** Models a sine function
- **circular** Models a pi/2 circle arc
- **exponential** Models an exponential function
- **elastic** Models an elastic release of energy
- **back** Models a pullback and relase
- **bounce** Models the bouncing of a ball

Modifiers

- **-in** The easing function is applied as-is
- **-out** The easing function is applied in reverse
- **-in-out** The first half of the transition it is applied as-is, while in the last half it is reversed

Examples

```
anim <- ggplot(mtcars, aes(mpg, disp)) +
  transition_states(gear, transition_length = 2, state_length = 1) +
  enter_fade() +
  exit_fade()

## Not run:
# Default uses linear easing
animate(anim)

## End(Not run)

# Change all to 'cubic-in-out' for a smoother appearance
anim1 <- anim +
  ease_aes('cubic-in-out')
## Not run:
animate(anim1)

## End(Not run)

# Only change easing of y variables
anim2 <- anim +
  ease_aes(y = 'bounce-in')
## Not run:
animate(anim2)

## End(Not run)
```

enter_exit *Define how entering and exiting data behaves*

Description

The purpose of `enter_*()` and `exit_*()` is to control what happens with data that does not persist during a tween. In general the non-persistent data is transformed to an *invisible* version that can be tweened to, e.g. by setting the opacity to 0 or be moving the element off-screen. It is possible to define your own transformations, or rely on some of the build in *effects*.

Usage

```
enter_manual(default = NULL, ..., name = "manual")

enter_appear(early = FALSE, ...)

enter_fade(..., alpha = 0)

enter_grow(..., size = 0)

enter_recolour(..., colour = "white", fill = colour)

enter_recolor(..., color = "white", fill = color)

enter_fly(..., x_loc = NA, y_loc = NA)

enter_drift(..., x_mod = 0, y_mod = 0)

enter_reset()

exit_manual(default = NULL, ..., name = "manual")

exit_disappear(early = FALSE, ...)

exit_fade(..., alpha = 0)

exit_shrink(..., size = 0)

exit_recolour(..., colour = "white", fill = colour)

exit_recolor(..., color = "white", fill = color)

exit_fly(..., x_loc = NA, y_loc = NA)

exit_drift(..., x_mod = 0, y_mod = 0)

exit_reset()
```

Arguments

default	A default transformation to use
...	Additional specific transformations either named by the geom (e.g. bar, or by its position in the layer stack, e.g. "2")
name	A name for the manual modification (only used when printing the object)
early	Should the data appear in the beginning of the transition or in the end
alpha	The start/end transparency.
size	The proportional start/end size. 0 means complete shrinking while 1 means no shrinking
colour, color, fill	The start/end colour and fill the elements should (dis)appear into
x_loc, y_loc	Start and end positions of the graphic elements
x_mod, y_mod	Modification to add to the entering or exiting data

User-defined transformations

All enter/exit functions allows the user to add additional transformation functions targeting specific layers. If the functions are named, then the name is understood to reference the class of geoms it applies to. If the functions are unnamed or numbered they will apply to the layer with a matching index in the stack. Named and indexed transformations cannot be mixed.

All modifications except enter_manual()/exit_manual() sets a range of modifications already, but further can be added with the .... For the manual versions a default transformation can be set which will apply to all layers that does not match any of the other given transformations. Often a single default transformation is enough and no specific transformations are needed.

Transformation can be given as any expression that can be converted with rlang::as_function(). This means that purrr style lambda functions are allowed in addition to anonymous functions etc. Transformation functions must accept a data.frame and return a data.frame of the same dimensions. The function will be called with the entering/exiting layer data, except for the case of polygon- and path-like layers in which case the function recieves the entering/exiting polygon/path data one by one. A special option is to set a transformation as NULL instead of a function. In that case the entering and exiting data will simply appear/disappear when it is no longer part of a frame.

Modification composition

Enter and exit modifications are composable so that multiple different ones can be added to an animation and will be applied in turn. You can also combine multiples and save them as a new enter or exit modification using c().

Due to the composable nature of enter and exit modifications it is not possible to overwrite a prior modification by adding a new. If it is needed to start from scratch then the sentinels enter_reset() and exit_reset() are provided which clears all prior modifications.

Modification types

A range of modification types are provided by gganimate and using enter_manual()/exit_manual() or modification composition it is possible to create your own.

appear/disappear will simply make elements appear/disappear at either the start or end of the transition. The default if nothing else is added.

fade will simply set the alpha value to zero making the elements fade in/out during the transition.

grow/shrink will set the elements to zero size making them gradually grow into / shrink out of existence. Zero size depends on the type of layer, e.g. polygons/paths will have all their points set to the mean, while points will have size/stroke set to zero.

recolour/recolor will change the colour and/or fill of the elements making them gradually change from the defined colour and into their try colour. Be aware that unless the colour and fill are set to the same as the background colour of the plot this modification needs to be combined with others to ensure that elements does not abruptly appear.

fly will set a specific x and y position where all elements will enter from/ exit to, irrespectible of their real position.

drift will modify the real position of the entering and exiting elements by a specified amount, e.g. setting `x_mod = -5` will let all elements enter from/exit to the left with a terminal position 5 points to the left of the real position.

Examples

```
# Default is appear/disappear
anim <- ggplot(mtcars, aes(factor(gear), mpg)) +
  geom_boxplot() +
  transition_states(gear, 2, 1)

# Fade-in, fly-out
anim1 <- anim +
  enter_fade() +
  exit_fly(x_loc = 7, y_loc = 40)

# Enter and exit accumulates
anim2 <- anim +
  enter_fade() + enter_grow() +
  exit_fly(x_loc = 7, y_loc = 40) + exit_recolour(fill = 'forestgreen')
```

---

frame_vars *Access metadata about the frames in an animation*

Description

This function extracts the metadata generated about each frame during rendering. It corresponds to the information available to the labels for string literal interpretation and is thus dependent on the transition and view used for the animation.

Usage

```
frame_vars(animation = last_animation())
```

Arguments

animation	The animation to extract metadata from. Defaults to the last rendered animation

Value

A data.frame with a row per frame in the animation and metadata as columns

---

renderers *Renderers provided by gganimate*

Description

The purpose of the renderer function is to take a list of image files and assemble them into an animation. gganimate provide a range of renderers but it is also possible to provide your own, if the supplied ones are lacking in any way. A renderer is given as argument to animate()/print() and receives the paths to the individual frames once they have been created.

Usage

```
gifski_renderer(file = NULL, loop = TRUE, width = NULL, height = NULL)

file_renderer(dir = ".", prefix = "gganim_plot", overwrite = FALSE)

av_renderer(file = NULL, vfilter = "null", codec = NULL, audio = NULL)

ffmpeg_renderer(
  format = "auto",
  ffmpeg = NULL,
  options = list(pix_fmt = "yuv420p")
)

magick_renderer(loop = TRUE)

sprite_renderer()
```

Arguments

file	The animation file
loop	Logical. Should the produced gif loop
width, height	Dimensions of the animation in pixels. If NULL will take the dimensions from the frame, otherwise it will rescale it.
dir	The directory to copy the frames to
prefix	The filename prefix to use for the image files
overwrite	Logical. If TRUE, existing files will be overwritten.

vfilter	A string defining an ffmpeg filter graph. This is the same parameter as the -vf argument in the ffmpeg command line utility.
codec	The name of the video codec. The default is libx264 for most formats, which usually the best choice. See the av documentation for more information.
audio	An optional file with sounds to add to the video
format	The video format to encode the animation into
ffmpeg	The location of the ffmpeg executable. If NULL it will be assumed to be on the search path
options	Either a character vector of command line options for ffmpeg or a named list of option-value pairs that will be converted to command line options automatically

Details

The gifski_renderer() is used unless otherwise specified in animate() or in options('gganimate.renderer'). This renderer requires both the gifski and png packages to be installed.

Other possible renderers are:

- magick_renderer() which requires the magick package and produce a gif. If gifski is not installed, the rendering will be much slower than using the gifski_renderer() and can potentially result in system problems when many frames need to be rendered (if gifski is installed magick will use it under the hood)
- av_renderer() which requies the av package and uses ffmpeg to encode the animation into a video file.
- ffmpeg_renderer() which requires that ffmpeg has been installed on your computer. As with av_renderer() it will use ffmpeg to encode the animation into a video
- sprite_renderer() which requires magick and will render the animation into a spritesheet
- file_renderer() which has no dependencies and simply returns the animation as a list of image files (one for each frame)

It is possible to create your own renderer function providing that it matches the required signature (frames and fps argument). The return value of your provided function will be the return value ultimately given by animate()

Value

The provided renderers are factory functions that returns a new function that take frames and fps as arguments, the former being a character vector with file paths to the images holding the separate frames, in the order they should appear, and the latter being the framerate to use for the animation in frames-per-second.

The return type of the different returned renderers are:

- gifski_renderer: Returns a gif_image object
- magick_renderer: Returns a magick-image object
- av_renderer: Returns a video_file object
- ffmpeg_renderer: Returns a video_file object
- file_renderer: Returns a vector of file paths

Examples

```
anim <- ggplot(mtcars, aes(mpg, disp)) +
  transition_states(gear, transition_length = 2, state_length = 1) +
  enter_fade() +
  exit_fade()

## Not run:
# Renderers are specified in the `animate()` function
animate(anim, renderer = sprite_renderer())

## End(Not run)
```

---

shadow_mark *Show original data as background marks*

Description

This shadow lets you show the raw data behind the current frame. Both past and/or future raw data can be shown and styled as you want.

Usage

```
shadow_mark(past = TRUE, future = FALSE, ..., exclude_layer = NULL)
```

Arguments

past	Should raw data from earlier in the animation be shown
future	Should raw data from later in the animation be shown
...	changes to the shadow data, e.g. alpha = alpha/2 or colour = 'grey'
exclude_layer	Indexes of layers that should be excluded.

See Also

Other shadows: shadow_null(), shadow_trail(), shadow_wake()

Examples

```
# Use any of the aesthetics to add a style to either the past or the future raw data.
# Adding a grouping variable in a transition call prior to calling `shadow_mark()` will
# allow transitioning through different states in time.

p1 <- ggplot(airquality, aes(Day, Temp)) +
  geom_line(color = 'red', size = 1) +
  transition_time(Month) +
  shadow_mark(colour = 'black', size = 0.75)

# animate(p1)
```

```
# Add a future = TRUE argument to show data later in the animation.

p2 <- ggplot(airquality, aes(Day, Temp)) +
  geom_line(color = 'red', size = 1) +
  transition_time(Month) +
  shadow_mark(color = 'black', size = 0.75, past = FALSE, future = TRUE)

# animate(p2)
```

---

shadow_null *A non-existent shadow*

Description

This is the default shadow that simply doesn't show anything other than the data for the current frame.

Usage

```
shadow_null()
```

See Also

Other shadows: shadow_mark(), shadow_trail(), shadow_wake()

---

shadow_trail *A trail of evenly spaced old frames*

Description

This shadow will trace the movement in your animation by keeping every *n*th frame and will thus produce a breadcrumb-like trail. Note that the shadow frames will not be equidistant in space but in time (that is, if a point moves slowly the *crumbs* will be closer to each other). It is possible to modify the look of the shadow by changing the different graphic parameters in the data

Usage

```
shadow_trail(distance = 0.05, max_frames = Inf, ..., exclude_layer = NULL)
```

Arguments

distance	The temporal distance between the frames to show, as a fraction of the full animation length
max_frames	The maximum number of shadow frames to show
...	changes to the shadow data, e.g. alpha = alpha/2 or colour = 'grey'
exclude_layer	Indexes of layers that should be excluded.

See Also

Other shadows: shadow_mark(), shadow_null(), shadow_wake()

Examples

```
anim <- ggplot(airquality, aes(Day, Temp, colour = factor(Month))) +
  geom_point() +
  transition_time(Day)

# Change distance between points
anim1 <- anim +
  shadow_trail(0.02)

# Style shadow differently
anim2 <- anim +
  shadow_trail(alpha = 0.3, shape = 2)

# Restrict the shadow to 10 frames
anim3 <- anim +
  shadow_trail(max_frames = 10)
```

---

shadow_wake *Show preceding frames with gradual falloff*

Description

This shadow is meant to draw a small wake after data by showing the latest frames up to the current. You can choose to gradually diminish the size and/or opacity of the shadow. The length of the wake is not given in absolute frames as that would make the animation susceptible to changes in the framerate. Instead it is given as a proportion of the total length of the animation.

Usage

```
shadow_wake(
  wake_length,
  size = TRUE,
  alpha = TRUE,
  colour = NULL,
  fill = NULL,
  falloff = "cubic-in",
  wrap = TRUE,
  exclude_layer = NULL,
  exclude_phase = c("enter", "exit")
)
```

Arguments

wake_length	A number between 0 and 1 giving the length of the wake, in relation to the total number of frames.
size	Numeric indicating the size the wake should end on. If NULL then size is not modified. Can also be a boolean with TRUE beeing equal 0 and FALSE beeing equal to NULL
alpha	as size but for alpha modification of the wake
colour, fill	colour or fill the wake should end on. If NULL they are not modified.
falloff	An easing function that control how size and/or alpha should change.
wrap	Should the shadow wrap around, so that the first frame will get shadows from the end of the animation.
exclude_layer	Indexes of layers that should be excluded.
exclude_phase	Element phases that should not get a shadow. Possible values are 'enter', 'exit', 'static', 'transition', and 'raw'. If NULL all phases will be included. Defaults to 'enter' and 'exit'

See Also

Other shadows: shadow_mark(), shadow_null(), shadow_trail()

Examples

```
anim <- ggplot(iris, aes(Petal.Length, Sepal.Length)) +
  geom_point() +
  labs(title = "{closest_state}") +
  transition_states(Species, transition_length = 4, state_length = 1)

# `shadow_wake` can be combined with e.g. `transition_states` to show
# motion of geoms as they are in transition with respect to the selected state.
anim1 <- anim +
  shadow_wake(wake_length = 0.05)

# Different qualities can be manipulated by setting a value for it that it
# should taper off to
anim2 <- anim +
  shadow_wake(0.1, size = 10, alpha = FALSE, colour = 'grey92')

# Use `detail` in the `animate()` call to increase the number of calculated
# frames and thus make the wake smoother
## Not run:
animate(anim2, detail = 5)

## End(Not run)
```

split_animation *Split an animation into chunks*

Description

Sometimes it is of interest to split an animation out in smaller chunks so they can be orchestrated, e.g. in a presentation. This function lets you provide a 'factor' to split by in the same way as `base::split()` though this one will be evaluated in the context of the animations `frame_vars()` data, so you can split directly on frame metadata.

Usage

```
split_animation(animation = last_animation(), by)
```

Arguments

`animation`	The animation to extract metadata from. Defaults to the last rendered animation
`by`	An unquoted expression to be evaluated in the context of the frame metadata. The result must be of equal length to the number of frames in the animation and define a grouping

Value

Depending on the output type of the renderer used to produce the animation. Often a list with elements referencing the chunks of the animation. that can then be saved individually.

transition_components *Transition individual components through their own lifecycle*

Description

This transition allows individual visual components to define their own "life-cycle". This means that the final animation will not have any common "state" and "transition" phase as any component can be moving or static at any point in time.

Usage

```
transition_components(
  time,
  range = NULL,
  enter_length = NULL,
  exit_length = NULL
)
```

Arguments

time	The unquoted name of the column holding the time for each state of the components
range	The range the animation should span. Defaults to the range of time plus enter and exit length
enter_length, exit_length	How long time should be spend on enter and exit transitions. Defaults to 0

Label variables

transition_components makes the following variables available for string literal interpretation, in addition to the general ones provided by animate():

- **frame_time** gives the time that the current frame corresponds to

Object permanence

transition_components uses the group aesthetic of each layer to identify which rows in the input data correspond to the same graphic element and will therefore define stages in time that the element will animate through. The group aesthetic, if not set, will be calculated from the interaction of all discrete aesthetics in the layer (excluding label), so it is often better to set it explicitly when animating, to make sure your data is interpreted in the right way. If the group aesthetic is not set, and no discrete aesthetics exists then all rows will have the same group.

Computed Variables

It is possible to use variables calculated by the statistic to define the transition. Simply inclose the variable in stat() in the same way as when using computed variables in aesthetics.

See Also

Other transitions: transition_events(), transition_filter(), transition_layers(), transition_manual(), transition_null(), transition_reveal(), transition_states(), transition_time()

Examples

```
data <- data.frame(
  x = runif(10),
  y = runif(10),
  size = sample(1:3, 10, TRUE),
  time = c(1, 4, 6, 7, 9, 6, 7, 8, 9, 10),
  id = rep(1:2, each = 5)
)

anim <- ggplot(data, aes(x, y, group = id, size = size)) +
  geom_point() +
  transition_components(time)

# By default the time range is set to the range of the time variable (plus
# any enter and exit length), but this can be overwritten
```

```
anim2 <- ggplot(data, aes(x, y, group = id, size = size)) +
  geom_point() +
  transition_components(time, range = c(4, 8))

# If you are using any enter/exit functions you need to give them some time
anim3 <- ggplot(data, aes(x, y, group = id, size = size)) +
  geom_point() +
  transition_components(time, enter_length = 2, exit_length = 2) +
  enter_grow() +
  exit_fade()
```

`transition_events` *Transition individual events in and out*

Description

This transition treats each visual element as an event in time and allows you to control the duration and enter/exit length individually for each event.

Usage

```
transition_events(
  start,
  end = NULL,
  range = NULL,
  enter_length = NULL,
  exit_length = NULL
)
```

Arguments

`start`, `end`	The unquoted expression giving the start and end time of each event. If `end` is `NULL` the event will be treated as having no duration.
`range`	The range the animation should span. Defaults to the range of the events from they enter to they have exited.
`enter_length`, `exit_length`	The unquoted expression giving the length to be used for enter and exit for each event.

Label variables

`transition_components` makes the following variables available for string literal interpretation, in addition to the general ones provided by `animate()`:

- **frame_time** gives the time that the current frame corresponds to

Object permanence

transition_events does not link rows across data to the same graphic element, so elements will be defined uniquely by each row and its specific start, end, enter and exit.

Computed Variables

It is possible to use variables calculated by the statistic to define the transition. Simply inclose the variable in stat() in the same way as when using computed variables in aesthetics.

See Also

Other transitions: transition_components(), transition_filter(), transition_layers(), transition_manual(), transition_null(), transition_reveal(), transition_states(), transition_time()

Examples

```
data <- data.frame(
  x = 1:10,
  y = runif(10),
  begin = runif(10, 1, 100),
  length = runif(10, 5, 20),
  enter = runif(10, 5, 10),
  exit = runif(10, 5, 10)
)

anim <- ggplot(data, aes(x, y)) +
  geom_col() +
  transition_events(start = begin,
                    end = begin + length,
                    enter_length = enter,
                    exit_length = exit) +
 enter_grow() +
 exit_drift(x_mod = 11) +
 exit_fade()
```

---

transition_filter *Transition between different filters*

Description

This transition allows you to transition between a range of filtering conditions. The conditions are expressed as logical statements and rows in the data will be retained if the statement evaluates to TRUE. It is possible to keep filtered data on display by setting keep = TRUE which will let data be retained as the result of the exit function. Note that if data is kept the enter function will have no effect.

Usage

```
transition_filter(
  transition_length = 1,
  filter_length = 1,
  ...,
  wrap = TRUE,
  keep = FALSE
)
```

Arguments

transition_length	The relative length of the transition. Will be recycled to match the number of states in the data
filter_length	The relative length of the pause at the states. Will be recycled to match the number of states in the data
...	A number of expressions to be evaluated in the context of the layer data, returning a logical vector. If the expressions are named, the name will be available as a frame variable.
wrap	Should the animation *wrap-around*? If TRUE the last filter will be transitioned into the first.
keep	Should rows that evaluates to FALSE be kept in the data as it looks after exit has been applied

Label variables

transition_filter makes the following variables available for string literal interpretation, in addition to the general ones provided by animate():

- **transitioning** is a boolean indicating whether the frame is part of the transitioning phase
- **previous_filter** The name of the last filter the animation was at
- **closest_filter** The name of the filter closest to this frame
- **next_filter** The name of the next filter the animation will be part of
- **previous_expression** The expression of the last filter the animation was at
- **closest_expression** The expression of the filter closest to this frame
- **next_expression** The expression of the next filter the animation will be part of

Object permanence

transition_filter does not link rows across data to the same graphic element, so elements will be defined uniquely by each row. If keep = TRUE the rows not matching the conditions of a filter is not removed from the plot after the exit animation, and a possible subsequent enter will begin from the state they were left in, rather than enter anew from the state defined by the enter function.

Computed Variables

It is possible to use variables calculated by the statistic to define the transition. Simply inclose the variable in `stat()` in the same way as when using computed variables in aesthetics.

See Also

Other transitions: `transition_components()`, `transition_events()`, `transition_layers()`, `transition_manual()`, `transition_null()`, `transition_reveal()`, `transition_states()`, `transition_time()`

Examples

```
anim <- ggplot(iris, aes(Petal.Width, Petal.Length, colour = Species)) +
  geom_point() +
  transition_filter(
    transition_length = 2,
    filter_length = 1,
    Setosa = Species == 'setosa',
    Long = Petal.Length > 4,
    Wide = Petal.Width > 2
  ) +
  ggtitle(
    'Filter: {closest_filter}',
    subtitle = '{closest_expression}'
  ) +
  enter_fade() +
  exit_fly(y_loc = 0)

# Setting `keep = TRUE` allows you to keep the culled data on display. Only
# exit functions will be used in that case (as elements enters from the
# result of the exit function)
anim2 <- ggplot(iris, aes(Petal.Width, Petal.Length, colour = Species)) +
  geom_point() +
  transition_filter(
    transition_length = 2,
    filter_length = 1,
    Setosa = Species == 'setosa',
    Long = Petal.Length > 4,
    Wide = Petal.Width > 2,
    keep = TRUE
  ) +
  ggtitle(
    'Filter: {closest_filter}',
    subtitle = '{closest_expression}'
  ) +
  exit_recolour(colour = 'grey') +
  exit_shrink(size = 0.5)
```

`transition_layers` *Build up a plot, layer by layer*

Description

This transition gradually adds layers to the plot in the order they have been defined. By default prior layers are kept for the remainder of the animation, but they can also be set to be removed as the next layer enters.

Usage

```
transition_layers(
  layer_length = 1,
  transition_length = 1,
  keep_layers = TRUE,
  from_blank = TRUE,
  layer_order = NULL,
  layer_names = NULL
)
```

Arguments

`layer_length`	The proportional time to pause at each layer before a new one enters
`transition_length`	The proportional time to use for the entrance of a new layer
`keep_layers`	Either an integer indicating for how many following layers the layers should stay on screen or a logical. In the case of the later, `TRUE` will mean keep the layer for the remainder of the animation (equivalent to setting it to `Inf`) and `FALSE` will mean to transition the layer out as the next layer enters.
`from_blank`	Should the first layer transition in or be present on the onset of the animation
`layer_order`	An alternative order the layers should appear in (default to using the stacking order). All other arguments that references the layers index in some way refers to this order.
`layer_names`	A character vector of names for each layers, to be used when interpreting label literals

Label variables

`transition_layers` makes the following variables available for string literal interpretation, in addition to the general ones provided by `animate()`:

- **transitioning** is a boolean indicating whether the frame is part of the transitioning phase
- **previous_layer** The name of the last layer the animation was showing
- **closest_layer** The name of the layer the animation is closest to showing
- **next_layer** The name of the next layer the animation will show
- **nlayers** The total number of layers

Object permanence

transition_layer does not link rows across data to the same graphic element, so elements will be defined uniquely by each row and the enter and exit of the layer it belongs to.

See Also

Other transitions: transition_components(), transition_events(), transition_filter(), transition_manual(), transition_null(), transition_reveal(), transition_states(), transition_time()

Examples

```
# Default is to use layer order and keep layers for duration of animation
anim <- ggplot(mtcars, aes(mpg, disp)) +
  geom_point() +
  geom_smooth(colour = 'grey', se = FALSE) +
  geom_smooth(aes(colour = factor(gear))) +
  transition_layers(layer_length = 1, transition_length = 2) +
  enter_fade() + enter_grow()

# Start with the first layer already present
anim1 <- ggplot(mtcars, aes(mpg, disp)) +
  geom_point() +
  geom_smooth(colour = 'grey', se = FALSE) +
  geom_smooth(aes(colour = factor(gear))) +
  transition_layers(layer_length = 1, transition_length = 2,
                    from_blank = FALSE) +
  enter_fade() + enter_grow()

# Change the order of the layers
anim2 <- ggplot(mtcars, aes(mpg, disp)) +
  geom_point() +
  geom_smooth(colour = 'grey', se = FALSE) +
  geom_smooth(aes(colour = factor(gear))) +
  transition_layers(layer_length = 1, transition_length = 2,
                    from_blank = FALSE, layer_order = c(3, 1, 2)) +
  enter_fade() + enter_grow()

# Keep layer 1 for the whole animation, but remove the 2nd layer as the 3rd
# enters
anim3 <- ggplot(mtcars, aes(mpg, disp)) +
  geom_point() +
  geom_smooth(colour = 'grey', se = FALSE) +
  geom_smooth(aes(colour = factor(gear))) +
  transition_layers(layer_length = 1, transition_length = 2,
                    from_blank = FALSE, keep_layers = c(Inf, 0, 0)) +
  enter_fade() + enter_grow() +
  exit_fade() + exit_shrink()
```

`transition_manual` *Create an animation by specifying the frame membership directly*

Description

This transition allows you to map a variable in your data to a specific frame in the animation. No tweening of data will be made and the number of frames in the animation will be decided by the number of levels in the frame variable.

Usage

```
transition_manual(frames, ..., cumulative = FALSE)
```

Arguments

`frames`	The unquoted name of the column holding the frame membership.
`...`	Additional variables
`cumulative`	Keep data from previous frames as part of the current frame data

Label variables

`transition_states` makes the following variables available for string literal interpretation, in addition to the general ones provided by `animate()`:

- **previous_frame** The name of the last frame the animation was at
- **current_frame** The name of the current frame
- **next_frame** The name of the next frame in the animation

Object permanence

`transition_manual` does not link rows across data to the same graphic element. Every frame is a discrete state and no animation between the states is done.

Computed Variables

It is possible to use variables calculated by the statistic to define the transition. Simply inclose the variable in `stat()` in the same way as when using computed variables in aesthetics.

See Also

Other transitions: `transition_components()`, `transition_events()`, `transition_filter()`, `transition_layers()`, `transition_null()`, `transition_reveal()`, `transition_states()`, `transition_time()`

Examples

```
anim <- ggplot(mtcars, aes(factor(gear), mpg)) +
  geom_boxplot() +
  transition_manual(gear)

# Using `cumulative = TRUE` to keep data from older frames
anim2 <- ggplot(mtcars, aes(factor(gear), mpg)) +
  geom_boxplot() +
  transition_manual(gear, cumulative = TRUE)

# Use `factor()` to set the order of the frames
anim3 <- ggplot(mtcars, aes(factor(gear), mpg)) +
  geom_boxplot() +
  transition_manual(factor(gear, levels = c('4', '3', '5')))
```

---

transition_null *Keep all data constant across the animation*

Description

Keep all data constant across the animation

Usage

```
transition_null()
```

See Also

Other transitions: transition_components(), transition_events(), transition_filter(), transition_layers(), transition_manual(), transition_reveal(), transition_states(), transition_time()

---

transition_reveal *Reveal data along a given dimension*

Description

This transition allows you to let data gradually appear, based on a given time dimension. In contrast to e.g. transition_time() transition_reveal() calculates intermediary values at exact positions instead of coercing raw values into the closest frame. It further keeps old data for path and polygon type layers so that they are gradually build up instead of being a set of disconnected segments as will happen when using transition_time() and shadow_mark() together.

Usage

```
transition_reveal(along, range = NULL, keep_last = TRUE)
```

Arguments

along	An unquoted expression giving the dimension to tween along. For a gradually revealing time series this should be set to the same as the x aesthetic.
range	The time range to animate. If NULL it will be set to the range of along
keep_last	For non-path/polygon layers should the last row be kept for subsequent frames.

Label variables

transition_reveal makes the following variables available for string literal interpretation, in addition to the general ones provided by animate():

- **frame_along** gives the position on the along-dimension that the current frame corresponds to

Object permanence

transition_reveal uses the group aesthetic of each layer to identify which rows in the input data correspond to the same graphic element and will therefore define a whole to be revealed over the animation. The group aesthetic, if not set, will be calculated from the interaction of all discrete aesthetics in the layer (excluding label), so it is often better to set it explicitly when animating, to make sure your data is interpreted in the right way. If the group aesthetic is not set, and no discrete aesthetics exists then all rows will have the same group.

Computed Variables

It is possible to use variables calculated by the statistic to define the transition. Simply inclose the variable in stat() in the same way as when using computed variables in aesthetics.

See Also

Other transitions: transition_components(), transition_events(), transition_filter(), transition_layers(), transition_manual(), transition_null(), transition_states(), transition_time()

Examples

```
anim <- ggplot(airquality, aes(Day, Temp, group = Month)) +
  geom_line() +
  transition_reveal(Day)

# Non-paths will only show the current position, not the history
anim1 <- ggplot(airquality, aes(Day, Temp, group = Month)) +
  geom_line() +
  geom_point(colour = 'red', size = 3) +
  transition_reveal(Day)

# Points can be kept by giving them a unique group and set `keep = TRUE` (the
# default)
anim2 <- ggplot(airquality, aes(Day, Temp, group = Month)) +
  geom_line() +
  geom_point(aes(group = seq_along(Day))) +
  geom_point(colour = 'red', size = 3) +
```

```
  transition_reveal(Day)

# Since ggplot2 3.4 geom_ribbon and geom_area has used stat_align
# This stat is incompatible with transition_reveal when applied before
# stats are calculated
anim3 <- ggplot(airquality, aes(Day, Temp, group = Month)) +
  geom_area() +
  transition_reveal(Day)

# This can be fixed by either reverting to use stat_identity
anim4 <- ggplot(airquality, aes(Day, Temp, group = Month)) +
  geom_area(stat = "identity") +
  transition_reveal(Day)

# Or by applying the transition after the stat
anim5 <- ggplot(airquality, aes(Day, Temp, group = Month)) +
  geom_area() +
  transition_reveal(after_stat(x))
```

`transition_states` *Transition between several distinct stages of the data*

Description

This transition splits your data into multiple states based on the levels in a given column, much like `ggplot2::facet_wrap()` splits up the data in multiple panels. It then tweens between the defined states and pauses at each state. Layers with data without the specified column will be kept constant during the animation (again, mimicking `facet_wrap`).

Usage

```
transition_states(states, transition_length = 1, state_length = 1, wrap = TRUE)
```

Arguments

`states`	The unquoted name of the column holding the state levels in the data.
`transition_length`	The relative length of the transition. Will be recycled to match the number of states in the data
`state_length`	The relative length of the pause at the states. Will be recycled to match the number of states in the data
`wrap`	Should the animation *wrap-around*? If `TRUE` the last state will be transitioned into the first.

Label variables

`transition_states` makes the following variables available for string literal interpretation, in addition to the general ones provided by `animate()`:

- **transitioning** is a boolean indicating whether the frame is part of the transitioning phase
- **previous_state** The name of the last state the animation was at
- **closest_state** The name of the state closest to this frame
- **next_state** The name of the next state the animation will be part of

Object permanence

`transition_states` uses the group aesthetic of each layer to identify which rows in the input data correspond to the same graphic element and will therefore define which elements will turn into each other between states. The group aesthetic, if not set, will be calculated from the interaction of all discrete aesthetics in the layer (excluding `label`), so it is often better to set it explicitly when animating, to make sure your data is interpreted in the right way. If the group aesthetic is not set, and no discrete aesthetics exists then all rows will have the same group. If the group aesthetic is not unique in each state, then rows will be matched first by group and then by index. Unmatched rows will appear/disappear, potentially using an enter or exit function.

Computed Variables

It is possible to use variables calculated by the statistic to define the transition. Simply inclose the variable in `stat()` in the same way as when using computed variables in aesthetics.

See Also

Other transitions: `transition_components()`, `transition_events()`, `transition_filter()`, `transition_layers()`, `transition_manual()`, `transition_null()`, `transition_reveal()`, `transition_time()`

Examples

```
anim <- ggplot(iris, aes(Sepal.Width, Petal.Width)) +
  geom_point() +
  labs(title = "{closest_state}") +
  transition_states(Species, transition_length = 3, state_length = 1)

# Use a unique group to avoid matching graphic elements
iris$group <- seq_len(nrow(iris))
anim1 <- ggplot(iris, aes(Sepal.Width, Petal.Width, group = group)) +
  geom_point() +
  labs(title = "{closest_state}") +
  transition_states(Species, transition_length = 3, state_length = 1) +
  enter_fade() +
  exit_fade()

# Set `wrap = FALSE` to avoid transitioning the last state to the first
anim2 <- ggplot(iris, aes(Sepal.Width, Petal.Width)) +
  geom_point() +
  labs(title = "{closest_state}") +
```

```
transition_states(Species, transition_length = 3, state_length = 1,
                  wrap = FALSE)
```

`transition_time` *Transition through distinct states in time*

Description

This is a variant of `transition_states()` that is intended for data where the states are representing specific point in time. The transition length between the states will be set to correspond to the actual time difference between them.

Usage

```
transition_time(time, range = NULL)
```

Arguments

`time`	An unquoted expression giving the time, and thus state membership, of each observation.
`range`	The time range to animate. If `NULL` it will be set to the range of `time`

Label variables

`transition_time` makes the following variables available for string literal interpretation, in addition to the general ones provided by `animate()`:

- **frame_time** gives the time that the current frame corresponds to

Object permanence

`transition_time` uses the group aesthetic of each layer to identify which rows in the input data correspond to the same graphic element and will therefore define which elements will turn into each other between time points. The group aesthetic, if not set, will be calculated from the interaction of all discrete aesthetics in the layer (excluding `label`), so it is often better to set it explicitly when animating, to make sure your data is interpreted in the right way. If the group aesthetic is not set, and no discrete aesthetics exists then all rows will have the same group. If the group aesthetic is not unique in each state, then rows will be matched first by group and then by index. Unmatched rows will appear/disappear, potentially using an enter or exit function.

Computed Variables

It is possible to use variables calculated by the statistic to define the transition. Simply inclose the variable in `stat()` in the same way as when using computed variables in aesthetics.

See Also

Other transitions: transition_components(), transition_events(), transition_filter(), transition_layers(), transition_manual(), transition_null(), transition_reveal(), transition_states()

Examples

```
anim <- ggplot(airquality, aes(Day, Temp)) +
  geom_point(aes(colour = factor(Month))) +
  transition_time(Day)

# Removing a time point will prolong the tweening between neighbouring time
# points so the time dimension stays linear
airquality_missing <- airquality[airquality$Day <= 10 | airquality$Day >= 20, ]
anim1 <- ggplot(airquality_missing, aes(Day, Temp)) +
  geom_point(aes(colour = factor(Month))) +
  transition_time(Day)

# Range can be constrained if needed
anim2 <- ggplot(airquality, aes(Day, Temp)) +
  geom_point(aes(colour = factor(Month))) +
  transition_time(Day, range = c(10L, 20L))

# The group aesthetic is used to connect elements
# No grouping
anim3 <- ggplot(airquality, aes(Day, Temp)) +
  geom_line() +
  transition_time(Month)

# Group by month
anim4 <- ggplot(airquality, aes(Day, Temp)) +
  geom_line(aes(group = Month)) +
  transition_time(Month) +
  enter_fade() +
  exit_fade()
```

---

view_follow *Let the view follow the data in each frame*

Description

This view will set the panels to include the data present in the frame.

Usage

```
view_follow(
  fixed_x = FALSE,
  fixed_y = FALSE,
  exclude_layer = NULL,
  aspect_ratio = 1
)
```

Arguments

fixed_x, fixed_y	Either a logical indicating if the dimension should not be modified by the view, or a numeric vector giving the lower and upper bounds of the dimension. For the latter, an NA value will be substituted for whatever is calculated by the view (e.g. fixed_x = c(0, NA)) will fix the minimum x value to 0 and let the view calculate the upper bound.
exclude_layer	Integer vector of layer indices that should be ignored when calculating the view
aspect_ratio	If the coord is fixed, ensure that the view matches the given aspect ratio. Will override anything given in fixed_x/fixed_y

See Also

Other views: view_static(), view_step(), view_zoom()

Examples

```
anim <- ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_point() +
  labs(title = "{closest_state}") +
  transition_states(Species, transition_length = 4, state_length = 1) +
  view_follow()

# Fixing a dimension can be done in general
anim1 <- ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_point() +
  labs(title = "{closest_state}") +
  transition_states(Species, transition_length = 4, state_length = 1) +
  view_follow(fixed_x = TRUE)

# ...or just for one side of the range
anim1 <- ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_point() +
  labs(title = "{closest_state}") +
  transition_states(Species, transition_length = 4, state_length = 1) +
  view_follow(fixed_x = c(4, NA), fixed_y = c(2, NA))
```

---

view_static *Keep a fixed view that include all of the data*

Description

This view keeps positional scales fixed across all frames

Usage

```
view_static()
```

See Also

Other views: view_follow(), view_step(), view_zoom()

---

view_step *Follow the data in steps*

Description

This view is a bit like view_follow() but will not match the data in each frame. Instead it will switch between being static and zoom to the range of the data. It is a great pairing with transition_states() as it can move the view while the data is static and then be static while the data moves. The standard version will look at the data present in the calculated frames and set the ranges based on that, while the _manual version will allow you to define your own ranges.

Usage

```
view_step(
  pause_length = 1,
  step_length = 1,
  nsteps = NULL,
  look_ahead = pause_length,
  delay = 0,
  include = TRUE,
  ease = "cubic-in-out",
  wrap = TRUE,
  pause_first = FALSE,
  fixed_x = FALSE,
  fixed_y = FALSE,
  exclude_layer = NULL,
  aspect_ratio = 1
)

view_step_manual(
  pause_length = 1,
  step_length = 1,
  xmin,
  xmax,
  ymin,
  ymax,
  delay = 0,
  ease = "cubic-in-out",
  wrap = TRUE,
  pause_first = FALSE,
  fixed_x = FALSE,
  fixed_y = FALSE,
  exclude_layer = NULL,
  aspect_ratio = 1
)
```

Arguments

pause_length	The relative length the view will be kept static. Will be recycled to match the number of steps
step_length	The relative length the view will use to transition to the new position. Will be recycled to match the number of steps
nsteps	The number of steps. If NULL it will be set to the max length of pause_length or step_length
look_ahead	A relative length to look ahead in the animation to get the new zoom area. Allow the view to zoom to where the data will be
delay	A relative length to switch the view back and forth relative to the actual frames. E.g. if delay is calculated to 5 frames, frame 6 will get the view intended for frame 1.
include	Should the steps include both the start and end frame range
ease	The easing function used for the step
wrap	As in transition_states(). Should the view wrap around and zoom back to the first state.
pause_first	Should the view start with a pause. The default is to start with a step so that it is aligned to the static period in transition_states()
fixed_x, fixed_y	Either a logical indicating if the dimension should not be modified by the view, or a numeric vector giving the lower and upper bounds of the dimension. For the latter, an NA value will be substituted for whatever is calculated by the view (e.g. fixed_x = c(0, NA)) will fix the minimum x value to 0 and let the view calculate the upper bound.
exclude_layer	Integer vector of layer indices that should be ignored when calculating the view
aspect_ratio	If the coord is fixed, ensure that the view matches the given aspect ratio. Will override anything given in fixed_x/fixed_y
xmin, xmax, ymin, ymax	Vectors of even length defining the boundaries of the different views to go through

See Also

Other views: view_follow(), view_static(), view_zoom()

Examples

```
anim <- ggplot(iris, aes(Petal.Length, Petal.Width)) +
  geom_point() +
  transition_states(Species, transition_length = 2, state_length = 1) +
  view_step(pause_length = 2, step_length = 1, nsteps = 3)

# Default is to include the data from the two states you're stepping between
# but this can be turned off
anim <- ggplot(iris, aes(Petal.Length, Petal.Width)) +
  geom_point() +
```

```
  transition_states(Species, transition_length = 2, state_length = 1) +
  view_step(pause_length = 2, step_length = 1, nsteps = 3, include = FALSE)

# Default is to work off-beat of transition_states so that view changes while
# data is static. Setting pause_first=TRUE changes this
anim <- ggplot(iris, aes(Petal.Length, Petal.Width)) +
  geom_point() +
  transition_states(Species, transition_length = 2, state_length = 1) +
  view_step(pause_length = 1, step_length = 2, nsteps = 3, pause_first = TRUE)

# If the transition doesn't wrap, then the view shouldn't either
anim <- ggplot(iris, aes(Petal.Length, Petal.Width)) +
  geom_point() +
  transition_states(Species, transition_length = 2, state_length = 1, wrap = FALSE) +
  view_step(pause_length = 2, step_length = 1, nsteps = 3, wrap = FALSE)
```

`view_zoom` *Pan and zoom smoothly between different states*

Description

This view is in many ways equivalent to `view_step()` and `view_step_manual()` but instead of simply tweening the bounding box of each view it implement the smooth zoom and pan technique developed by Reach & North (2018). It gradually zooms out and then in during the pan to allow a smooth transition of the view. As with `view_step()` the standard version will look at the data present in the calculated frames and set the ranges based on that, while the `_manual` version will allow you to define your own ranges to zoom between.

Usage

```
view_zoom(
  pause_length = 1,
  step_length = 1,
  nsteps = NULL,
  look_ahead = 0,
  delay = 0,
  include = FALSE,
  pan_zoom = 0,
  ease = "sine-in-out",
  wrap = TRUE,
  pause_first = TRUE,
  fixed_x = FALSE,
  fixed_y = FALSE,
  exclude_layer = NULL,
  aspect_ratio = 1
)
```

```
view_zoom_manual(
  pause_length = 1,
  step_length = 1,
  xmin,
  xmax,
  ymin,
  ymax,
  delay = 0,
  pan_zoom = 0,
  ease = "sine-in-out",
  wrap = TRUE,
  pause_first = TRUE,
  fixed_x = FALSE,
  fixed_y = FALSE,
  exclude_layer = NULL,
  aspect_ratio = 1
)
```

Arguments

pause_length	The relative length the view will be kept static. Will be recycled to match the number of steps
step_length	The relative length the view will use to transition to the new position. Will be recycled to match the number of steps
nsteps	The number of steps. If NULL it will be set to the max length of pause_length or step_length
look_ahead	A relative length to look ahead in the animation to get the new zoom area. Allow the view to zoom to where the data will be
delay	A relative length to switch the view back and forth relative to the actual frames. E.g. if delay is calculated to 5 frames, frame 6 will get the view intended for frame 1.
include	Should the steps include both the start and end frame range
pan_zoom	The tradeoff between pan- and zoom-induced movement. Negative values will value zoom over pan and positive values will value pan over zoom
ease	The easing function used for the step
wrap	As in transition_states(). Should the view wrap around and zoom back to the first state.
pause_first	Should the view start with a pause. The default is to start with a step so that it is aligned to the static period in transition_states()
fixed_x, fixed_y	Either a logical indicating if the dimension should not be modified by the view, or a numeric vector giving the lower and upper bounds of the dimension. For the latter, an NA value will be substituted for whatever is calculated by the view (e.g. fixed_x = c(0, NA)) will fix the minimum x value to 0 and let the view calculate the upper bound.
exclude_layer	Integer vector of layer indices that should be ignored when calculating the view

`aspect_ratio` If the coord is fixed, ensure that the view matches the given aspect ratio. Will override anything given in `fixed_x`/`fixed_y`

`xmin`, `xmax`, `ymin`, `ymax`
Vectors of even length defining the boundaries of the different views to go through

References

Reach, A., North, C. (2018) *Smooth, Efficient, and Interruptible Zooming and Panning*. IEEE Transactions on Visualization and Computer Graphics DOI:10.1109/TVCG.2018.2800013

See Also

Other views: `view_follow()`, `view_static()`, `view_step()`

Examples

```
anim <- ggplot(iris, aes(Petal.Length, Petal.Width, colour = Species)) +
  geom_point() +
  transition_states(Species, transition_length = 2, state_length = 1) +
  shadow_mark(past = TRUE, future = TRUE, colour = 'grey') +
  view_zoom(pause_length = 1, step_length = 2, nsteps = 3)

# Use pan_zoom to change the relationship between pan- and zoom movement
# Mainly zooming
anim1 <- ggplot(iris, aes(Petal.Length, Petal.Width, colour = Species)) +
  geom_point() +
  transition_states(Species, transition_length = 2, state_length = 1) +
  shadow_mark(past = TRUE, future = TRUE, colour = 'grey') +
  view_zoom(pause_length = 1, step_length = 2, nsteps = 3, pan_zoom = -3)

# Mainly panning
anim2 <- ggplot(iris, aes(Petal.Length, Petal.Width, colour = Species)) +
  geom_point() +
  transition_states(Species, transition_length = 2, state_length = 1) +
  shadow_mark(past = TRUE, future = TRUE, colour = 'grey') +
  view_zoom(pause_length = 1, step_length = 2, nsteps = 3, pan_zoom = -3)
```

Index