Package 'ggarrow'

November 2, 2026

Title Arrows for 'ggplot2'

Version 0.2.1

Description A 'ggplot2' extension that adds specialised arrow geometry layers. It offers more arrow options than the standard 'grid' arrows that are built-in many line-based geom layers.

License MIT + file LICENSE

Encoding UTF-8

Depends ggplot2 (>= 3.5.0)

Imports cli, grid, lifecycle, polyclip, rlang (>= 1.1.0), S7, scales, utils, vctrs

Suggests covr, knitr, rmarkdown, testthat (>= 3.0.0), vdiffr

Config/testthat/edition 3

VignetteBuilder knitr

URL https://teunbrand.github.io/ggarrow/, https://github.com/teunbrand/ggarrow

BugReports https://github.com/teunbrand/ggarrow/issues

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Teun van den Brand [aut, cre] (ORCID: <https://orcid.org/0000-0002-9335-7468>)

Maintainer Teun van den Brand <tahvdbrand@gmail.com>

Repository CRAN

Date/Publication 2026-11-02 22:00:02 UTC

Contents

annotate_arrow 2
arrow_ornaments 4
continuous_arrow_scales 5

discrete_arrow_scales 8
distortion_functions 11
draw_key_arrow 12
element_arrow 12
GeomArrow 15
geom_arrow 15
geom_arrow_chain 20
geom_arrow_curve 24
geom_arrow_segment 29
grob_arrow 33
grob_arrow_curve 36
scale_resect 38
whirlpool 42

Index **43**

`annotate_arrow` *Arrow annotation layer*

Description

This function mirrors `annotate()` with the following changes. First, the `geom` argument is pre-populated with `"arrow"`. Second, several parameters from ggarrow are special-cased, because no warning needs to be issued when they don't have length 1.

Usage

```
annotate_arrow(
  geom = "arrow",
  x = NULL,
  y = NULL,
  xmin = NULL,
  xmax = NULL,
  ymin = NULL,
  ymax = NULL,
  xend = NULL,
  yend = NULL,
  ...,
  na.rm = FALSE
)
```

Arguments

`geom`	name of geom to use for annotation
`x`, `y`, `xmin`, `ymin`, `xmax`, `ymax`, `xend`, `yend`	Positioning aesthetics. At least one of these must be specified.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

Value

A `<Layer>` ggproto object that can be added to a plot.

See Also

Other arrow geoms: `geom_arrow()`, `geom_arrow_chain()`, `geom_arrow_curve()`, `geom_arrow_segment()`

Examples

```
# Annotate an arrow
ggplot() +
  annotate_arrow(
    x = c(0, 1), y = c(0, 1),
    arrow_head = arrow_head_line(),
    arrow_fins = arrow_fins_line(),
    length_head = unit(5, "mm"),
    length_fins = unit(5, "mm")
  )

# Still works with other geoms as well
ggplot(mtcars, aes(x = wt, y = mpg)) + geom_point() +
  annotate_arrow("text", x = 4, y = 25, label = "Some text")
```

arrow_ornaments *Arrow ornament functions*

Description

There are two types of arrow ornament functions: functions for arrow heads, and functions for arrow fins. The heads and fins can be used interchangeably, but the name makes it clearer what is suitable.

Usage

```
arrow_head_wings(offset = 20, inset = 30)

arrow_fins_feather(indent = 0.3, outdent = indent, height = 0.5)

arrow_head_line(angle = 30, lineend = "butt")

arrow_fins_line(angle = 30, lineend = "butt")

arrow_cup(lineend = "round", angle = NULL)

arrow_head_minimal(angle = 45)

arrow_fins_minimal(angle = 45)

arrow_head_halfwing(offset = 20, inset = 30, direction = 1L)

arrow_head_halfline(angle = 30, lineend = "butt", direction = 1L)
```

Arguments

offset, angle	A numeric(1) giving an angle in degrees for the angle between the line and tip.
inset	A numeric(1) giving an angle in degrees for the angle inside the tip of the arrowhead.
indent, outdent	A numeric(1) giving the fraction of the feather feather length to offset the notch and the end respectively.
height	A numeric(1) ratio between the length of the feathers and the height of the feathers.
lineend	A character(1), one of "butt", "square", "round" or "parallel". For arrow_cup(), only "butt" and "round" are allowed.
direction	A numeric(1) whose sign determines the side of the shaft that the arrowhead is placed on.

Details

The convention for these functions is that the arrow shaft is fused to the ornament at the (0,0) point and the ornaments ends at the (1,0) point.

Value

A <matrix[n, 2]> with x and y columns describing a polygon. It has a notch_angle attribute that is used fusing the fins/head to the shaft of the arrow. They can be given to an arrow plotting function.

Functions

- arrow_head_wings(): Places two triangles at either side of the line. Let ABC be a triangle, where A is at the end of the line, B is on the line and C is the arrow wingtip. Then offset is the angle at corner A and inset is the angle at corner C.
- arrow_fins_feather(): Places trapezoids at either side of the line. Let ABCD be a quadrilateral shape, where A is at the end of the line, B is on the line, and CD is parallel to AB, but offset from the line. Then, indent is the distance along the line between A and D and outdent is the distance along the line between B and C.
- arrow_head_line(): A line as an arrow head.
- arrow_fins_line(): A line as an arrow fin.
- arrow_cup(): A curved line some fixed distance away from the point to be resected, resembling a 'cup' shape.
- arrow_head_minimal(): This is a 'fake' arrow head who in practice doesn't draw anything, but sets the notch_angle attribute such that the arrow shaft is whittled into a triangular point.
- arrow_fins_minimal(): This is a 'fake' arrow head who in practise doesn't draw anything, but sets the notch_angle attribute such that a triangle is taken out of the arrow shaft.
- arrow_head_halfwing(): This a 'half' version of arrow_head_wings().
- arrow_head_halfline(): This a 'half' version of arrow_head_line().

Examples

```
# Plotting winged head
plot(c(-0.5, 1), c(-0.6, 0.6), type = "n")
polygon(arrow_head_wings(), col = "gray")

# Plotting feather fins
plot(c(0, 1), c(-0.25, 0.25), type = "n")
polygon(arrow_fins_feather(), col = "gray")
```

---

continuous_arrow_scales

Continuous arrow scales

Description

These scales can map continuous input to an argument of an arrow generator. The arrow head, arrow fins and middle arrows have separate scales and by default use different generators.

Usage

```
scale_arrow_head_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  generator = arrow_head_wings,
  map_arg = "offset",
  other_args = list(),
  range = c(10, 80),
  transform = "identity",
  guide = "legend"
)

scale_arrow_fins_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  generator = arrow_fins_feather,
  map_arg = "indent",
  other_args = list(),
  range = c(0, 1),
  transform = "identity",
  guide = "legend"
)

scale_arrow_mid_continuous(
  name = waiver(),
  breaks = waiver(),
  labels = waiver(),
  limits = NULL,
  generator = arrow_head_wings,
  map_arg = "offset",
  other_args = list(),
  range = c(10, 80),
  transform = "identity",
  guide = "legend"
)
```

Arguments

name	The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.
breaks	One of: • `NULL` for no breaks • `waiver()` for the default breaks computed by the transformation object

- A numeric vector of positions
- A function that takes the limits as input and returns breaks as output (e.g., a function returned by `scales::extended_breaks()`). Note that for position scales, limits are provided after scale expansion. Also accepts rlang lambda function notation.

`labels` One of the options below. Please note that when `labels` is a vector, it is highly recommended to also set the `breaks` argument as a vector to protect against unintended mismatches.

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

`limits` One of:

- `NULL` to use the default scale range
- A numeric vector of length two providing limits of the scale. Use `NA` to refer to the existing minimum or maximum
- A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang lambda function notation. Note that setting limits on positional scales will **remove** data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see `coord_cartesian()`).

`generator` A `<function>` that can create an arrow ornament, such as ornamentation functions.

`map_arg` An argument of the `generator` function to map input to.

`other_args` Additional, fixed, arguments to pass to the `generator`.

`range` The range that `generator`'s `map_arg` may take

`transform` For continuous scales, the name of a transformation object or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "date", "exp", "hms", "identity", "log", "log10", "log1p", "log2", "logit", "modulus", "probability", "probit", "pseudo_log", "reciprocal", "reverse", "sqrt" and "time".

A transformation object bundles together a transform, its inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called `transform_<name>`. If transformations require arguments, you can call them from the scales package, e.g. `scales::transform_boxcox(p = 2)`. You can create your own transformation with `scales::new_transform()`.

`guide` A function used to create a guide or its name. See `guides()` for more information.

Value

A `<Scale>` that can be added to a plot.

Examples

```
base <- ggplot(whirlpool(5), aes(x, y, colour = group)) +
  coord_fixed()

p <- base +
  geom_arrow(
    aes(arrow_head = as.integer(group)),
    length_head = 10
  )

# A typical scale
p + scale_arrow_head_continuous()

# Change other arguments passed to the generator
p + scale_arrow_head_continuous(other_args = list(inset = 90))

# Using another argument of the generator
p + scale_arrow_head_continuous(name = "inset",  map_arg = "inset")

# Using a different generator
p + scale_arrow_head_continuous(
  generator = arrow_head_line,
  map_arg = "angle",
  range = c(20, 80)
)

# The same goes for other arrow aesthetics, but the `generator()` might
# differ.
base +
  geom_arrow(
    aes(arrow_fins = as.integer(group), arrow_mid = as.integer(group)),
    length_fins = 10, arrow_head = NULL
  ) +
  scale_arrow_fins_continuous(map_arg = "height", range = c(0.1, 1)) +
  scale_arrow_mid_continuous(map_arg = "inset")
```

---

discrete_arrow_scales *Discrete arrow scales*

Description

These scales can map discrete input to various sorts of arrow shapes. The arrow head, arrow fins and middle arrows have separate scales.

Usage

```
scale_arrow_head_discrete(values = NULL, aesthetics = "arrow_head", ...)

scale_arrow_fins_discrete(values = NULL, aesthetics = "arrow_fins", ...)

scale_arrow_mid_discrete(values = NULL, aesthetics = "arrow_mid", ...)
```

Arguments

`values` One of the following:

- A <character> vector of arrow function names, without the `arrow_`-prefix, such as `"head_wings"` or `"fins_line"`.
- An unnested <list>, possibly mixed <list>, containing any of the following elements:
  - A single <character> as described above.
  - A <function> that when called without any arguments produces a 2-column <matrix> that can be used as an arrow.
  - A 2-column <matrix> giving a polygon to use as an arrow.
- `NULL`, which defaults to a built-in palette with a maximum of 3 arrows.

`aesthetics` The names of the aesthetics that this scale works with

`...` Arguments passed on to `ggplot2::discrete_scale`

`scale_name` **[Deprecated]** The name of the scale that should be used for error messages associated with this scale.

`palette` A palette function that when called with a single integer argument (the number of levels in the scale) returns the values that they should take (e.g., `scales::pal_hue()`).

`name` The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

`breaks` One of:

- `NULL` for no breaks
- `waiver()` for the default breaks (the scale limits)
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output. Also accepts rlang lambda function notation.

`labels` One of the options below. Please note that when `labels` is a vector, it is highly recommended to also set the `breaks` argument as a vector to protect against unintended mismatches.

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

`limits` One of:

- `NULL` to use the default scale values
- A character vector that defines possible values of the scale and their order
- A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang lambda function notation.

`expand` For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function `expansion()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

`na.translate` Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify `na.translate = FALSE`.

`na.value` If `na.translate = TRUE`, what aesthetic value should the missing values be displayed as? Does not apply to position scales where `NA` is always placed at the far right.

`drop` Should unused factor levels be omitted from the scale? The default, `TRUE`, uses the levels that appear in the data; `FALSE` includes the levels in the factor. Please note that to display every level in a legend, the layer should use `show.legend = TRUE`.

`guide` A function used to create a guide or its name. See `guides()` for more information.

`position` For position scales, The position of the axis. `left` or `right` for y axes, `top` or `bottom` for x axes.

`fallback.palette` Function to use when `palette = NULL` and the palette is not represented in the theme.

`call` The `call` used to construct the scale for reporting messages.

`super` The super class to use for the constructed scale

Value

A `<Scale>` that can be added to a plot.

Examples

```
# A standard arrow plot
p <- ggplot(whirlpool(5), aes(x, y, colour = group)) +
  geom_arrow(length_head = 10, length_fins = 10, arrow_head = NULL)

# A character vector naming arrow shapes as arrow head scale
p + aes(arrow_head = group) +
  scale_arrow_head_discrete(values = c(
      "head_wings", "head_line", "head_minimal", "fins_line", "fins_feather"
  ))

# A mixed list with arrows as arrow fins scale
p + aes(arrow_fins = group) +
  scale_arrow_fins_discrete(values = list(
    "head_wings",              # Using a character
    arrow_head_wings(20, 100), # Using an arrow function
    NULL,                      # No arrow
    matrix(c(1, 0, 0, 0, 0.5, -0.5), ncol = 2), # A matrix
    "fins_feather"
  ))
```

distortion_functions *Line distortion functions*

Description

These are functions that build a pattern that can be used as the `distort` argument in arrow geoms. They produce one 'oscillation' that will be repeated to distort a line.

Usage

```
distort_sinewave(length = 4, width = 2, n = 30L)

distort_sawtooth(length = 4, width = 2)

distort_squarewave(length = 4, width = 2)
```

Arguments

`length`	A positive `numeric(1)` setting the length along the path in millimetres that one repetition of the distortion occupies. Actual length may end up slightly wider to cover a remainder length.
`width`	A `numeric(1)` setting the width (amplitude) of the distortion in millimetres. Negative values can be used to flip the distortion.
`n`	An `integer(1)` setting the number of vertices to use for the path.

Details

It is possible to write your own `distort_*()` function. The assumption is that an oscillation starts at the `(0, 0)` coordinate and ends at the `(length, 0)` coordinate.

The exact length of an oscillation will be fitted based on the arc-length of the path it will be placed on. If the arc-length will not accommodate an exact integer number of oscillations, the oscillations will be stretched to cover the whole arc-length.

Value

A `<matrix[n, 2]>` with `x` and `y` columns describing a path. It may have a `size` attribute.

Functions

- `distort_sinewave()`: Follows a sine waveform.
- `distort_sawtooth()`: Follows a triangular waveform.
- `distort_squarewave()`: Follows a square waveform.

Examples

```
plot(distort_sinewave(), type = 'l')
```

draw_key_arrow *Legend key glyph for arrows*

Description

Like any legend key glyphs, this key can be used to display arrows in a legend.

Usage

```
draw_key_arrow(data, params, size)
```

Arguments

data	A single row data frame containing the scaled aesthetics to display in this key
params	A list of additional parameters supplied to the geom.
size	Width and height of key in mm.

Value

An <arrow_path> grob

Examples

```
ggplot(mpg, aes(displ, colour = factor(cyl))) +
  geom_density(key_glyph = draw_key_arrow)
```

element_arrow *Arrow theme element*

Description

Using the theme system, draws arrows in places where `element_line()` are valid theme elements. Note that the default use of `element_arrow()` does *not* actually draw an arrow unless one of the `arrow_` arguments is set.

Usage

```
element_arrow(
  colour = NULL,
  linewidth = NULL,
  linewidth_head = NULL,
  linewidth_fins = NULL,
  stroke_colour = NULL,
  stroke_width = NULL,
  arrow_head = NULL,
```

```
  arrow_fins = NULL,
  arrow_mid = NULL,
  length = NULL,
  length_head = NULL,
  length_fins = NULL,
  length_mid = NULL,
  resect = NULL,
  resect_head = NULL,
  resect_fins = NULL,
  justify = NULL,
  force_arrow = NULL,
  mid_place = NULL,
  lineend = NULL,
  linejoin = NULL,
  linemitre = NULL,
  distort = NULL,
  inherit.blank = FALSE,
  ...
)
```

Arguments

`colour` The colour of the arrow.

`linewidth`, `linewidth_head`, `linewidth_fins`
: The width of the arrow shaft in millimetres. `linewidth` is the default width, whereas `linewidth_head` and `linewidth_fins` can set non-uniform width at the end and start of the line respectively.

`stroke_colour` The colour of the arrow outline.

`stroke_width` The width of the arrow outlien.

`arrow_head`, `arrow_fins`, `arrow_mid`
: Arrow ornament shapes for the arrow head, arrow fins and middle arrows respectively. Can be one of the following:
  - `NULL` for not drawing the ornament.
  - A `<character>` of length 1 naming an ornament constructor without the `"arrow_"`-prefix, like `"head_wings"` or `"fins_feather"`.
  - A 2-column matrix, such as those built by the ornament constructors.

`length`, `length_head`, `length_fins`, `length_mid`
: Determines the size of the arrow ornaments. `length` sets the default length, whereas `length_head`, `length_fins` and `length_mid` set the lengths of the arrow head, arrow fins or middle arrows respectively. Can be one of the following:
  - A `<numeric>` to set the ornament size relative to the `linewidth{_\*}` settings.
  - A `<unit>` to control the ornament size in an absolute manner. Behaviour of relative units such as `"npc"` or `"null"` is undefined.

`resect`, `resect_head`, `resect_fins`
: A `numeric(1)` denoting millimetres or `<unit>` to set an offset from the start and end points of the line such that the arrow is shortened. `resect` sets the default

offset, whereas `resect_head` and `resect_fins` sets these offsets for the end- and start-point respectively.

`justify` A `numeric(1)` between [0-1] to control where the arrow ornaments should be drawn relative to the (resected) path's endpoints. A value of `0` (default) sets the ornament's tips at the path's endpoint, whereas a value of `1` sets the ornament's base at the path's endpoint.

`force_arrow` A `logical(1)` which if `TRUE`, will draw arrow ornaments even when the path's length is shorter than the arrow heads and fins. If `FALSE`, such ornaments will be dropped.

`mid_place` Sets the location of middle (interior) ornaments when `arrow_mid` has been provided. Can be one of the following:

- A `<numeric>` vector with values between [0-1] to set middle ornaments at relative positions along the arc-length of the (resected) path.
- A `<unit>` to fill a path with ornaments with th provided unit as spacing between one ornament to the next.

`lineend` A `character(1)` setting the style of the line ends without ornaments. Can be `"round"`, `"butt"` or `"square"`.

`linejoin` A `character(1)` setting the style of path corners. Can be `"round"`, `"mitre"` or `"bevel"`.

`linemitre` A `numeric(1)` greater than 1 setting the path's mitre limits.

`distort` **[Experimental]**

A way of specifying a line distortion. A line distortion is not great at dealing with paths with pronounced angles. One of the following:

- `NULL` to not distort any paths.
- A `<matrix[n, 2]>` of x/y coordinates giving one 'oscillation' of a distortion in millimetres. The distortion functions create such matrices.
- A string, naming one of the distortion functions without the `distort_` prefix.

`inherit.blank` A `logical(1)` indicating if this element should inherit the existence of an `<element_blank>` among its parents. If `TRUE`, the existence of a blank element among its parents will cause this element to be blank as well. If `FALSE`, any blank parent element will be ignored when calculating final element state.

`...` Not currently in use.

Value

An `<element_arrow>` object that can replace `<element_line>` objects in `theme()`.

Examples

```
# Setting a bunch of arrows all over the theme
ggplot(whirlpool(5), aes(x, y, group = group)) +
  geom_path() +
  theme(
    # Proper arrow with variable width for x-axis line
```

```
  axis.line.x = element_arrow(
    arrow_head = "head_wings", linewidth_head = 2, linewidth_fins = 0
  ),
  # Just a variable width line for the y-axis line
  axis.line.y = element_arrow(linewidth_head = 0, linewidth_fins = 5,
                              lineend = "round"),
  # Arrows for the y-axis ticks
  axis.ticks.y = element_arrow(arrow_fins = arrow_head_line(angle = 45)),
  # Variable width lines for the x-axis ticks
  axis.ticks.x = element_arrow(linewidth_head = 3, linewidth_fins = 0),
  axis.ticks.length = unit(0.5, 'cm'),
  # Arrows for major panel grid
  panel.grid.major = element_arrow(
    arrow_head = "head_wings", arrow_fins = "fins_feather", length = 10
  ),
  # Shortened lines for the minor panel grid
  panel.grid.minor = element_arrow(resect = 20)
)
```

GeomArrow *ggarrow extensions to ggplot2*

Description

ggarrow relieas on the extension mechanism of ggplot2 through ggproto class objects, that allow for cross-package inheritance of geoms. These objects can be ignored by users for the purpose of making plots, since interacting with these objects is preferred through various `geom_*()` functions.

---

geom_arrow *Arrows*

Description

This arrow geom can be used to draw lines adorned with arrow heads or fins. It is useful as an annotation layer to point to or away from other things on the plot. An arrow typically consists of three parts: the arrowhead, the shaft and fins. This geom places arrow heads at the end of a line and fins at the beginning of a line.

Usage

```
geom_arrow(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  arrow_head = arrow_head_wings(),
```

```
  arrow_fins = NULL,
  arrow_mid = NULL,
  length = 4,
  length_head = NULL,
  length_fins = NULL,
  length_mid = NULL,
  justify = 0,
  force_arrow = FALSE,
  mid_place = 0.5,
  resect = 0,
  resect_head = NULL,
  resect_fins = NULL,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  sep = 0,
  distort = NULL,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used to override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.

position
: A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

  - The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.
  - A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter".
  - For more information and other ways to specify the position, see the layer position documentation.

...
: Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

  - Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
  - When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.
  - Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.
  - The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

arrow_head, arrow_fins, arrow_mid
: A function call to one of the arrow ornament functions that can determine the shape of the arrow head, fins or middle (interior) arrows.

length, length_head, length_fins, length_mid
: Determines the size of the arrow ornaments. length sets the default length, whereas length_head, length_fins and length_mid set the lengths of the arrow head, arrow fins or middle arrows respectively. Can be one of the following:

  - A <numeric> to set the ornament size relative to the linewidth{_*} settings.
  - A <unit> to control the ornament size in an absolute manner. Behaviour of relative units such as "npc" or "null" is undefined.

justify
: A numeric(1) between [0-1] to control where the arrows should be drawn relative to the path's endpoints. A value of 0 sets the arrow's tips at the path's end, whereas a value of 1 sets the arrow's base at the path's end.

force_arrow
: A logical(1) which, if TRUE an arrow will be drawn even when the length of the arrow is shorter than the arrow heads and fins. If FALSE, will drop such arrows.

mid_place
: Sets the location of middle (interior) arrows, when applicable. Can be one of the following:

  A numeric **vector** with values between [0-1] to set middle arrows at relative positions along the arc-length of a path.

  A <unit> to fill a path with arrows with the provided unit as distance between one arrow to the next.

resect, resect_head, resect_fins
: A numeric(1) denoting millimetres or <unit> to shorten the arrow. resect_head shortens the arrow from the arrow head side, whereas resect_fins shortens the arrow from the fins side. Both inherit from resect.

lineend
: Line end style (round, butt, square).

linejoin
: Line join style (round, mitre, bevel).

linemitre
: Line mitre limit (number greater than 1).

sep
: A numeric(1) setting offset spacing in millimetres between arrow paths that are identical or identical as inverses. The default, 0, will draw paths without offsets. Alternatively, a <unit>.

distort
: **[Experimental]**

  A way of specifying a line distortion. A line distortion is not great at dealing with paths with pronounced angles. One of the following:

  - NULL to not distort any paths.
  - A <matrix[n, 2]> of x/y coordinates giving one 'oscillation' of a distortion in millimetres. The distortion functions create such matrices.
  - A string, naming one of the distortion functions without the distort_ prefix.

na.rm
: If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

show.legend
: logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes
: If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

Value

A <Layer> ggproto object that can be added to a plot.

Aesthetics

geom_arrow() understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- alpha → NA
- arrow_fins → NULL
- arrow_head → NULL
- arrow_mid → NULL
- colour → via theme()
- group → inferred
- linetype → via theme()
- linewidth → via theme()
- resect_fins → NULL
- resect_head → NULL
- stroke_colour → via theme()
- stroke_width → via theme()

Learn more about setting these aesthetics in vignette("ggplot2-specs").

See Also

Other arrow geoms: annotate_arrow(), geom_arrow_chain(), geom_arrow_curve(), geom_arrow_segment()

Examples

```
# Setting up a plot
p <- ggplot(whirlpool(), aes(x, y, colour = group)) +
  coord_equal()

# A standard arrow
p + geom_arrow()

# Arrows can have varying linewidths
p + geom_arrow(aes(linewidth = arc))

# You can use `length_head` to decouple arrow-head size from linewidth
p + geom_arrow(aes(linewidth = arc), length_head = unit(10, "mm"))

# The arrow head shape can be controlled with the `arrow_head` argument
p + geom_arrow(arrow_head = arrow_head_line(), length_head = unit(10, "mm"))

# This works similarly for the arrow fins
p + geom_arrow(
  arrow_fins = arrow_fins_feather(),
  length_fins = unit(7, "mm")
)
```

`geom_arrow_chain` *Arrow chains*

Description

An arrow chains connects a set of coordinates with a sequence of arrows. The `geom_arrow_chain()` function can be useful to connect observations in a directed manner.

Usage

```
geom_arrow_chain(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  arrow_head = arrow_head_wings(),
  arrow_fins = NULL,
  arrow_mid = NULL,
  length = 4,
  length_head = NULL,
  length_fins = NULL,
  length_mid = NULL,
  justify = 0,
  force_arrow = FALSE,
  mid_place = 0.5,
  resect = 1,
  resect_head = NULL,
  resect_fins = NULL,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  sep = 0,
  distort = NULL,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).

`stat` The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following:

- A `Stat` ggproto subclass, for example `StatCount`.
- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`.
- For more information and other ways to specify the stat, see the layer stat documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.

- The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

arrow_head, arrow_fins, arrow_mid
: A function call to one of the arrow ornament functions that can determine the shape of the arrow head, fins or middle (interior) arrows.

length, length_head, length_fins, length_mid
: Determines the size of the arrow ornaments. length sets the default length, whereas length_head, length_fins and length_mid set the lengths of the arrow head, arrow fins or middle arrows respectively. Can be one of the following:

  - A <numeric> to set the ornament size relative to the linewidth{_*} settings.
  - A <unit> to control the ornament size in an absolute manner. Behaviour of relative units such as "npc" or "null" is undefined.

justify
: A numeric(1) between [0-1] to control where the arrows should be drawn relative to the path's endpoints. A value of 0 sets the arrow's tips at the path's end, whereas a value of 1 sets the arrow's base at the path's end.

force_arrow
: A logical(1) which, if TRUE an arrow will be drawn even when the length of the arrow is shorter than the arrow heads and fins. If FALSE, will drop such arrows.

mid_place
: Sets the location of middle (interior) arrows, when applicable. Can be one of the following:

  A numeric **vector** with values between [0-1] to set middle arrows at relative positions along the arc-length of a path.

  A <unit> to fill a path with arrows with the provided unit as distance between one arrow to the next.

resect, resect_head, resect_fins
: A numeric(1) denoting millimetres or <unit> to shorten the arrow. resect_head shortens the arrow from the arrow head side, whereas resect_fins shortens the arrow from the fins side. Both inherit from resect.

lineend
: Line end style (round, butt, square).

linejoin
: Line join style (round, mitre, bevel).

linemitre
: Line mitre limit (number greater than 1).

sep
: A numeric(1) setting offset spacing in millimetres between arrow paths that are identical or identical as inverses. The default, 0, will draw paths without offsets. Alternatively, a <unit>.

distort
: **[Experimental]**

  A way of specifying a line distortion. A line distortion is not great at dealing with paths with pronounced angles. One of the following:

  - NULL to not distort any paths.
  - A <matrix[n, 2]> of x/y coordinates giving one 'oscillation' of a distortion in millimetres. The distortion functions create such matrices.
  - A string, naming one of the distortion functions without the distort_ prefix.

na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

Value

A <Layer> ggproto object that can be added to a plot.

Aesthetics

geom_arrow_segment() understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- xend ***or*** yend
- alpha → NA
- arrow_fins → NULL
- arrow_head → NULL
- arrow_mid → NULL
- colour → via theme()
- group → inferred
- linetype → via theme()
- linewidth → via theme()
- linewidth_fins → NULL
- linewidth_head → NULL
- resect_fins → NULL
- resect_head → NULL
- stroke_colour → via theme()
- stroke_width → via theme()

The linewidth_fins and linewidth_head inherit from linewidth. They can be used to separately control the start- and end-width.

Learn more about setting these aesthetics in vignette("ggplot2-specs").

See Also

Other arrow geoms: annotate_arrow(), geom_arrow(), geom_arrow_curve(), geom_arrow_segment()

Examples

```
# Setup dummy data
t <- seq(0, 2 * pi, length.out = 11)
l <- rep(c(1, 0.4), length.out = 11)

df <- data.frame(
  x = cos(t) * l,
  y = sin(t) * l,
  size = l + 0.4
)

p <- ggplot(df, aes(x, y, size = size)) +
  geom_point(colour = 2) +
  coord_equal()

# An arrow chains adapts to the `size` aesthetic to go nicely with points
p + geom_arrow_chain()

# Without arrowhead, it is similar to a `type = 'b'` base R plot
p + geom_arrow_chain(arrow_head = NULL)

# To widen the gap, one can increase the `resect` parameter
p + geom_arrow_chain(resect = 5)

# To ignore the points, set `resect` and `size` to 0
p + geom_arrow_chain(size = 0, resect = 0)

# Linewidths will be interpolated across arrows
p + geom_arrow_chain(aes(linewidth = seq_along(x)))

# Alternatively, we can set them separately for starts and ends
p + geom_arrow_chain(linewidth_fins = 0, linewidth_head = 3)
```

---

geom_arrow_curve *Curves with arrows*

Description

This arrow geom can be used to draw curves from one point to oneanother with arrow heads or fins.

Usage

```
geom_arrow_curve(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  curvature = 0.5,
```

```
  angle = 90,
  ncp = 5L,
  arrow_head = arrow_head_wings(),
  arrow_fins = NULL,
  arrow_mid = NULL,
  length = 4,
  length_head = NULL,
  length_fins = NULL,
  length_mid = NULL,
  justify = 0,
  force_arrow = FALSE,
  mid_place = 0.5,
  resect = 0,
  resect_head = NULL,
  resect_fins = NULL,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  sep = 0,
  distort = NULL,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used to override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count".

- For more information and other ways to specify the stat, see the layer stat documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`curvature` A numeric value giving the amount of curvature. Negative values produce left-hand curves, positive values produce right-hand curves, and zero produces a straight line.

`angle` A numeric value between 0 and 180, giving an amount to skew the control points of the curve. Values less than 90 skew the curve towards the start point and values greater than 90 skew the curve towards the end point.

`ncp` The number of control points used to draw the curve. More control points creates a smoother curve.

`arrow_head`, `arrow_fins`, `arrow_mid`
A function call to one of the arrow ornament functions that can determine the shape of the arrow head, fins or middle (interior) arrows.

length, length_head, length_fins, length_mid
: Determines the size of the arrow ornaments. length sets the default length, whereas length_head, length_fins and length_mid set the lengths of the arrow head, arrow fins or middle arrows respectively. Can be one of the following:
  - A <numeric> to set the ornament size relative to the linewidth{_*} settings.
  - A <unit> to control the ornament size in an absolute manner. Behaviour of relative units such as "npc" or "null" is undefined.

justify
: A numeric(1) between [0-1] to control where the arrows should be drawn relative to the path's endpoints. A value of 0 sets the arrow's tips at the path's end, whereas a value of 1 sets the arrow's base at the path's end.

force_arrow
: A logical(1) which, if TRUE an arrow will be drawn even when the length of the arrow is shorter than the arrow heads and fins. If FALSE, will drop such arrows.

mid_place
: Sets the location of middle (interior) arrows, when applicable. Can be one of the following:

  A numeric **vector** with values between [0-1] to set middle arrows at relative positions along the arc-length of a path.

  A <unit> to fill a path with arrows with the provided unit as distance between one arrow to the next.

resect, resect_head, resect_fins
: A numeric(1) denoting millimetres or <unit> to shorten the arrow. resect_head shortens the arrow from the arrow head side, whereas resect_fins shortens the arrow from the fins side. Both inherit from resect.

lineend
: Line end style (round, butt, square).

linejoin
: Line join style (round, mitre, bevel).

linemitre
: Line mitre limit (number greater than 1).

sep
: A numeric(1) setting offset spacing in millimetres between arrow paths that are identical or identical as inverses. The default, 0, will draw paths without offsets. Alternatively, a <unit>.

distort
: **[Experimental]**

  A way of specifying a line distortion. A line distortion is not great at dealing with paths with pronounced angles. One of the following:
  - NULL to not distort any paths.
  - A <matrix[n, 2]> of x/y coordinates giving one 'oscillation' of a distortion in millimetres. The distortion functions create such matrices.
  - A string, naming one of the distortion functions without the distort_ prefix.

na.rm
: If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

show.legend
: logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

Value

A `<Layer>` ggproto object that can be added to a plot.

Aesthetics

`geom_arrow()` understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- alpha → NA
- arrow_fins → NULL
- arrow_head → NULL
- arrow_mid → NULL
- colour → via theme()
- group → inferred
- linetype → via theme()
- linewidth → via theme()
- resect_fins → NULL
- resect_head → NULL
- stroke_colour → via theme()
- stroke_width → via theme()

Learn more about setting these aesthetics in `vignette("ggplot2-specs")`.

See Also

Other arrow geoms: `annotate_arrow()`, `geom_arrow()`, `geom_arrow_chain()`, `geom_arrow_segment()`

Examples

```
curve_data <- data.frame(
  x1 = c(2.62, 1.835),
  x2 = c(3.57, 5.250),
  y1 = c(21.0, 33.9),
  y2 = c(15.0, 10.4),
  group = c("A", "B")
)

ggplot(mtcars, aes(wt, mpg)) +
  geom_point() +
  geom_arrow_curve(
    aes(x = x1, y = y1, xend = x2, yend = y2,
        colour = group, arrow_head = group),
```

```
  data = curve_data,
  curvature = -0.2, length_head = 10
)
```

`geom_arrow_segment` *Arrow segments*

Description

`geom_arrow_segment()` draws a straight arrow between points (x, y) and (xend, yend). In contrast to `geom_segment()`, the `xend` and `yend` aesthetics default to `x` and `y` respectively, so only one of `xend` and `yend` is required.

Usage

```
geom_arrow_segment(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  arrow_head = arrow_head_wings(),
  arrow_fins = NULL,
  arrow_mid = NULL,
  length = 4,
  length_head = NULL,
  length_fins = NULL,
  length_mid = NULL,
  justify = 0,
  force_arrow = FALSE,
  mid_place = 0.5,
  resect = 0,
  resect_head = NULL,
  resect_fins = NULL,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  sep = 0,
  distort = NULL,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used to override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the geom part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`arrow_head`, `arrow_fins`, `arrow_mid`
: A function call to one of the arrow ornament functions that can determine the shape of the arrow head, fins or middle (interior) arrows.

`length`, `length_head`, `length_fins`, `length_mid`
: Determines the size of the arrow ornaments. `length` sets the default length, whereas `length_head`, `length_fins` and `length_mid` set the lengths of the arrow head, arrow fins or middle arrows respectively. Can be one of the following:

  - A `<numeric>` to set the ornament size relative to the `linewidth{_\*}` settings.
  - A `<unit>` to control the ornament size in an absolute manner. Behaviour of relative units such as `"npc"` or `"null"` is undefined.

`justify`
: A `numeric(1)` between [0-1] to control where the arrows should be drawn relative to the path's endpoints. A value of `0` sets the arrow's tips at the path's end, whereas a value of `1` sets the arrow's base at the path's end.

`force_arrow`
: A `logical(1)` which, if `TRUE` an arrow will be drawn even when the length of the arrow is shorter than the arrow heads and fins. If `FALSE`, will drop such arrows.

`mid_place`
: Sets the location of middle (interior) arrows, when applicable. Can be one of the following:

  A `numeric` **vector** with values between [0-1] to set middle arrows at relative positions along the arc-length of a path.

  A `<unit>` to fill a path with arrows with the provided unit as distance between one arrow to the next.

`resect`, `resect_head`, `resect_fins`
: A `numeric(1)` denoting millimetres or `<unit>` to shorten the arrow. `resect_head` shortens the arrow from the arrow head side, whereas `resect_fins` shortens the arrow from the fins side. Both inherit from `resect`.

`lineend`
: Line end style (round, butt, square).

`linejoin`
: Line join style (round, mitre, bevel).

`linemitre`
: Line mitre limit (number greater than 1).

`sep`
: A `numeric(1)` setting offset spacing in millimetres between arrow paths that are identical or identical as inverses. The default, 0, will draw paths without offsets. Alternatively, a `<unit>`.

distort **[Experimental]**

A way of specifying a line distortion. A line distortion is not great at dealing with paths with pronounced angles. One of the following:

- NULL to not distort any paths.
- A <matrix[n, 2]> of x/y coordinates giving one 'oscillation' of a distortion in millimetres. The distortion functions create such matrices.
- A string, naming one of the distortion functions without the distort_ prefix.

na.rm If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

show.legend logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

Value

A <Layer> ggproto object that can be added to a plot.

Aesthetics

geom_arrow_segment() understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- xend ***or*** yend
- alpha → NA
- arrow_fins → NULL
- arrow_head → NULL
- arrow_mid → NULL
- colour → via theme()
- group → inferred
- linetype → via theme()
- linewidth → via theme()
- linewidth_fins → NULL
- linewidth_head → NULL
- resect_fins → NULL
- resect_head → NULL
- stroke_colour → via theme()
- stroke_width → via theme()

The linewidth_fins and linewidth_head inherit from linewidth. They can be used to separately control the start- and end-width.

Learn more about setting these aesthetics in vignette("ggplot2-specs").

See Also

Other arrow geoms: annotate_arrow(), geom_arrow(), geom_arrow_chain(), geom_arrow_curve()

Examples

```
# Setup dummy data
set.seed(42)
df <- data.frame(
  x = LETTERS[1:6],
  y = 6:1 + rnorm(6)
)

# We can omit either `xend` or `yend` for this segment geom
p <- ggplot(df, aes(x, y = 0, yend = y, colour = x))
p + geom_arrow_segment()

# We can set the linewidth globally
p + geom_arrow_segment(aes(linewidth = y))

# Or separately for the head and fins
p + geom_arrow_segment(aes(linewidth_head = y, linewidth_fins = 0))

# We can also place arrows in the middle
p + geom_arrow_segment(
  arrow_mid = arrow_head_line(), mid_place = c(0.33, 0.66),
  arrow_head = NULL # Turn off head
)
```

---

grob_arrow *Arrow grob*

Description

Creates a graphical object that draws arrows. An arrow typically consists of three parts: the arrowhead, the shaft and fins. Relative to how an arrow is drawn from coordinates, these three parts describe the end, middle and beginning of an arrow line.

Usage

```
grob_arrow(
  x = unit(c(0, 1), "npc"),
  y = unit(c(0, 1), "npc"),
  id = NULL,
  id.lengths = NULL,
```

```
  arrow_head = arrow_head_wings(),
  arrow_fins = NULL,
  arrow_mid = NULL,
  length_head = unit(5, "mm"),
  length_fins = NULL,
  length_mid = NULL,
  justify = 0,
  shaft_width = unit(1, "mm"),
  mid_place = 0.5,
  resect = unit(0, "mm"),
  resect_fins = NULL,
  resect_head = NULL,
  force_arrow = FALSE,
  offset = NULL,
  distort = NULL,
  default.units = "mm",
  name = NULL,
  gp = gpar(),
  vp = NULL
)
```

Arguments

x	A numeric vector or unit object specifying x-values.
y	A numeric vector or unit object specifying y-values.
id	A numeric vector used to separate locations in x and y into multiple lines. All locations with the same id belong to the same line.
id.lengths	A numeric vector used to separate locations in x and y into multiple lines. Specifies consecutive blocks of locations which make up separate lines.
arrow_head, arrow_fins, arrow_mid	A <matrix[n, 2]>, such as those returned by arrow ornament functions, giving arrow shapes. The matrix can (should) have the notch_angle attribute that will be used to fuse the shaft to the arrow ornaments. If NULL, no ornament will be drawn.
length_head, length_fins, length_mid	A <unit> object controlling the size of the arrow ornaments.
justify	A numeric(1) between [0-1] to control where the arrows should be drawn relative to the path's endpoints. A value of 0 sets the arrow's tips at the path's end, whereas a value of 1 sets the arrow's base at the path's end.
shaft_width	A <unit> object controlling the width of the arrow's shaft.
mid_place	Sets the location of middle (interior) arrows, when applicable. Can be one of the following: **A** numeric **vector** with values between [0-1] to set middle arrows at relative positions along the arc-length of a path. **A** <unit> to fill a path with arrows with the provided unit as distance between one arrow to the next.

resect, resect_fins, resect_head
: A <unit> object that can be used to create an offset between the endings of the coordinates and where the arrow will be displayed visually. resect_fins and resect_head control this offset at the start and end of the arrow respectively and both default to resect.

force_arrow
: A logical(1) which, if TRUE an arrow will be drawn even when the length of the arrow is shorter than the arrow heads and fins. If FALSE, will drop such arrows.

offset
: A numeric() vector giving the offset in millimetres per group to displace paths.

distort
: **[Experimental]**

A way of specifying a line distortion. A line distortion is not great at dealing with paths with pronounced angles. One of the following:

- NULL to not distort any paths.
- A <matrix[n, 2]> of x/y coordinates giving one 'oscillation' of a distortion in millimetres. The distortion functions create such matrices.
- A string, naming one of the distortion functions without the distort_ prefix.

default.units
: A string indicating the default units to use if x or y are only given as numeric vectors.

name
: A character identifier.

gp
: An object of class "gpar", typically the output from a call to the function gpar. This is basically a list of graphical parameter settings.

vp
: A Grid viewport object (or NULL).

Value

A <arrow_path> graphical object.

Examples

```
requireNamespace("grid")

# Creating an arrow
arrow <- grob_arrow(
  x = unit(c(0.2, 0.5, 0.8, 0.2, 0.5, 0.8), "npc"),
  y = unit(c(0.2, 0.8, 0.2, 0.8, 0.2, 0.8), "npc"),
  id.lengths  = c(3, 3),
  arrow_head  = arrow_head_wings(),
  arrow_fins  = arrow_fins_feather(),
  length_fins = 8,
  shaft_width = 1,
  gp = grid::gpar(fill = c("dodgerblue", "tomato"), col = "black")
)

# Drawing the arrow
grid::grid.newpage(); grid::grid.draw(arrow)
```

grob_arrow_curve *Arrow curve grob.*

Description

Creates a graphical object that draws curved arrows.

Usage

```
grob_arrow_curve(
  x1,
  y1,
  x2,
  y2,
  default.units = "mm",
  curvature = 1,
  angle = 90,
  ncp = 1L,
  shape = 0.5,
  square = TRUE,
  squareShape = 1L,
  inflect = FALSE,
  open = TRUE,
  name = NULL,
  gp = gpar(),
  vp = NULL,
  ...,
  width_head = unit(1, "mm"),
  width_fins = unit(1, "mm")
)
```

Arguments

x1	A numeric vector or unit object specifying the x-location of the start point.
y1	A numeric vector or unit object specifying the y-location of the start point.
x2	A numeric vector or unit object specifying the x-location of the end point.
y2	A numeric vector or unit object specifying the y-location of the end point.
default.units	A string indicating the default units to use if x1, y1, x2 or y2 are only given as numeric values.
curvature	A numeric value giving the amount of curvature. Negative values produce left-hand curves, positive values produce right-hand curves, and zero produces a straight line.
angle	A numeric value between 0 and 180, giving an amount to skew the control points of the curve. Values less than 90 skew the curve towards the start point and values greater than 90 skew the curve towards the end point.

ncp — The number of control points used to draw the curve. More control points creates a smoother curve.

shape — A numeric vector of values between -1 and 1, which control the shape of the curve relative to its control points. See `grid.xspline` for more details.

square — A logical value that controls whether control points for the curve are created city-block fashion or obliquely. When `ncp` is 1 and `angle` is 90, this is typically `TRUE`, otherwise this should probably be set to `FALSE` (see Examples below).

squareShape — A `shape` value to control the behaviour of the curve relative to any additional control point that is inserted if `square` is `TRUE`.

inflect — A logical value specifying whether the curve should be cut in half and inverted (see Examples below).

open — A logical value indicating whether to close the curve (connect the start and end points).

name — A character identifier.

gp — An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.

vp — A Grid viewport object (or NULL).

... — Arguments passed on to `grob_arrow`

`arrow_head,arrow_fins,arrow_mid` A `<matrix[n, 2]>`, such as those returned by arrow ornament functions, giving arrow shapes. The matrix can (should) have the `notch_angle` attribute that will be used to fuse the shaft to the arrow ornaments. If `NULL`, no ornament will be drawn.

`length_head,length_fins,length_mid` A `<unit>` object controlling the size of the arrow ornaments.

`resect,resect_fins,resect_head` A `<unit>` object that can be used to create an offset between the endings of the coordinates and where the arrow will be displayed visually. `resect_fins` and `resect_head` control this offset at the start and end of the arrow respectively and both default to `resect`.

`force_arrow` A `logical(1)` which, if `TRUE` an arrow will be drawn even when the length of the arrow is shorter than the arrow heads and fins. If `FALSE`, will drop such arrows.

`justify` A `numeric(1)` between [0-1] to control where the arrows should be drawn relative to the path's endpoints. A value of `0` sets the arrow's tips at the path's end, whereas a value of `1` sets the arrow's base at the path's end.

`offset` A `numeric()` vector giving the offset in millimetres per group to displace paths.

`distort` **[Experimental]**
A way of specifying a line distortion. A line distortion is not great at dealing with paths with pronounced angles. One of the following:

- `NULL` to not distort any paths.
- A `<matrix[n, 2]>` of x/y coordinates giving one 'oscillation' of a distortion in millimetres. The distortion functions create such matrices.
- A string, naming one of the distortion functions without the `distort_` prefix.

`mid_place` Sets the location of middle (interior) arrows, when applicable. Can be one of the following:

A `numeric` **vector** with values between [0-1] to set middle arrows at relative positions along the arc-length of a path.

A `<unit>` to fill a path with arrows with the provided unit as distance between one arrow to the next.

`x` A numeric vector or unit object specifying x-values.

`y` A numeric vector or unit object specifying y-values.

`id` A numeric vector used to separate locations in `x` and `y` into multiple lines. All locations with the same `id` belong to the same line.

`id.lengths` A numeric vector used to separate locations in `x` and `y` into multiple lines. Specifies consecutive blocks of locations which make up separate lines.

`width_fins`, `width_head`

A `<unit>` object controlling the width of the arrow's shaft at the `(x1, y1)` and `(x2, y2)` location respectively.

Value

A `<curve_arrow>` graphical object.

Examples

```
requireNamespace("grid")

# Creating the curved arrow
grob <- grob_arrow_curve(
  x1 = unit(0.25, "npc"), y1 = unit(0.25, "npc"),
  x2 = unit(0.75, "npc"), y2 = unit(0.75, "npc"),
  angle = 90, curvature = 0.5, ncp = 5,
  arrow_head = arrow_head_line()
)

# Drawing the arrow
grid::grid.newpage(); grid::grid.draw(grob)
```

---

scale_resect *Scale for resection*

Description

Arrow geoms have a `resect` aesthetic that controls how much an arrow should be shortened. These scales can help to rescale the output range of resection.

Usage

```
scale_resect_continuous(
  ...,
  range = NULL,
  aesthetics = c("resect_head", "resect_fins"),
  guide = "none"
)

scale_resect_discrete(
  ...,
  values = NULL,
  aesthetics = c("resect_head", "resect_fins"),
  range = NULL,
  guide = "none"
)
```

Arguments

`...` Arguments passed on to ggplot2::continuous_scale, ggplot2::discrete_scale

`name` The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

`breaks` One of:

- `NULL` for no breaks
- `waiver()` for the default breaks computed by the transformation object
- A numeric vector of positions
- A function that takes the limits as input and returns breaks as output (e.g., a function returned by `scales::extended_breaks()`). Note that for position scales, limits are provided after scale expansion. Also accepts rlang lambda function notation.

`minor_breaks` One of:

- `NULL` for no minor breaks
- `waiver()` for the default breaks (none for discrete, one minor break between each major break for continuous)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation. When the function has two arguments, it will be given the limits and major break positions.

`n.breaks` An integer guiding the number of major breaks. The algorithm may choose a slightly different number to ensure nice break labels. Will only have an effect if `breaks = waiver()`. Use `NULL` to use the default number of breaks given by the transformation.

`labels` One of the options below. Please note that when `labels` is a vector, it is highly recommended to also set the `breaks` argument as a vector to protect against unintended mismatches.

- `NULL` for no labels

- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

`limits` One of:

- `NULL` to use the default scale range
- A numeric vector of length two providing limits of the scale. Use `NA` to refer to the existing minimum or maximum
- A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang lambda function notation. Note that setting limits on positional scales will **remove** data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see `coord_cartesian()`).

`rescaler` A function used to scale the input values to the range [0, 1]. This is always `scales::rescale()`, except for diverging and n colour gradients (i.e., `scale_colour_gradient2()`, `scale_colour_gradientn()`). The `rescaler` is ignored by position scales, which always use `scales::rescale()`. Also accepts rlang lambda function notation.

`oob` One of:

- Function that handles limits outside of the scale limits (out of bounds). Also accepts rlang lambda function notation.
- The default (`scales::censor()`) replaces out of bounds values with `NA`.
- `scales::squish()` for squishing out of bounds values into range.
- `scales::squish_infinite()` for squishing infinite values into range.

`expand` For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function `expansion()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

`na.value` Missing values will be replaced with this value.

`transform` For continuous scales, the name of a transformation object or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "date", "exp", "hms", "identity", "log", "log10", "log1p", "log2", "logit", "modulus", "probability", "probit", "pseudo_log", "reciprocal", "reverse", "sqrt" and "time".

A transformation object bundles together a transform, its inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales package, and are called `transform_<name>`. If transformations require arguments, you can call them from the scales package, e.g. `scales::transform_boxcox(p = 2)`. You can create your own transformation with `scales::new_transform()`.

`trans` **[Deprecated]** Deprecated in favour of `transform`.

position For position scales, The position of the axis. left or right for y axes, top or bottom for x axes.

fallback.palette Function to use when palette = NULL and the palette is not represented in the theme.

call The call used to construct the scale for reporting messages.

na.translate Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify na.translate = FALSE.

drop Should unused factor levels be omitted from the scale? The default, TRUE, uses the levels that appear in the data; FALSE includes the levels in the factor. Please note that to display every level in a legend, the layer should use show.legend = TRUE.

range A numeric vector of length 2 indicating the minimum and maximum size of the resection after transformation in millimetres. range is mutually exclusive with the values argument in discrete scales.

aesthetics The names of the aesthetics that this scale works with.

guide A function used to create a guide or its name. See guides() for more information.

values (Discrete scale only) A numeric vector to map data values to. The values will be matched in order with the limits of the scale, or with breaks if provided. If this is a named vector, the values will be matched based on the names instead. Data values that don't match will be given na.value. values is mutually exclusive with the range

Details

Conceptually, these scales depart slightly from ggplot2 conventions. The scale_resect_continuous() function returns an identity scale when range = NULL (default) and a typical continuous scale when the range argument is set. The scale_resect_discrete() acts as a manual scale when values is set and as an ordinal scale when range is set.

Value

A <Scale> that can be added to a plot.

Examples

```
# A plot with points indicating path ends
p <- ggplot(whirlpool(5), aes(x, y, colour = group)) +
  geom_point(data = ~ subset(.x, arc == ave(arc, group, FUN = max)))

# Resect scale as an identity scale
p + geom_arrow(aes(resect_head = as.integer(group))) +
  scale_resect_continuous()

# Resect scale as typical continuous scale
p + geom_arrow(aes(resect_head = as.integer(group))) +
  scale_resect_continuous(range = c(0, 10))
```

```
# Resect scale as manual scale
p + geom_arrow(aes(resect_head = group)) +
  scale_resect_discrete(values = c(10, 5, 0, 5, 10))

# Resect scale as ordinal scale
p + geom_arrow(aes(resect_head = group)) +
  scale_resect_discrete(range = c(0, 10))
```

`whirlpool` *Whirlpool data*

Description

This function returns some made-up data to test arrow functionality with.

Usage

```
whirlpool(n = 5L, detail = 100L)
```

Arguments

`n`	The number of streams in the whirlpool.
`detail`	The number of points per stream.

Value

A `data.frame`

Examples

```
whirlpool()
```

Index

∗ **arrow geoms**
    annotate_arrow, 2
    geom_arrow, 15
    geom_arrow_chain, 20
    geom_arrow_curve, 24
    geom_arrow_segment, 29
<unit>, *13, 14*, *17*, *22*, *27*, *31*, *34, 35*, *37, 38*

aes(), *16*, *20*, *25*, *30*
alpha, *19*, *23*, *28*, *32*
annotate(), *2*
annotate_arrow, 2
annotate_arrow(), *19*, *23*, *28*, *33*
annotation_borders(), *18*, *23*, *28*, *32*
arrow ornament, *17*, *22*, *26*, *31*, *34*, *37*
arrow_cup *(*arrow_ornaments*)*, 4
arrow_fins_feather *(*arrow_ornaments*)*, 4
arrow_fins_line *(*arrow_ornaments*)*, 4
arrow_fins_minimal *(*arrow_ornaments*)*, 4
arrow_head_halfline *(*arrow_ornaments*)*, 4
arrow_head_halfwing *(*arrow_ornaments*)*, 4
arrow_head_line *(*arrow_ornaments*)*, 4
arrow_head_minimal *(*arrow_ornaments*)*, 4
arrow_head_wings *(*arrow_ornaments*)*, 4
arrow_ornaments, 4

colour, *19*, *23*, *28*, *32*
continuous_arrow_scales, 5
coord_cartesian(), *7*, *40*

discrete_arrow_scales, 8
distort_sawtooth
        *(*distortion_functions*)*, 11
distort_sinewave
        *(*distortion_functions*)*, 11
distort_squarewave
        *(*distortion_functions*)*, 11
distortion, *14*, *18*, *22*, *27*, *32*, *35*, *37*
distortion_functions, 11
draw_key_arrow, 12

element_arrow, 12
element_line(), *12*
expansion(), *10*, *40*

fortify(), *16*, *21*, *25*, *30*

geom_arrow, 15
geom_arrow(), *3*, *23*, *28*, *33*
geom_arrow_chain, 20
geom_arrow_chain(), *3*, *19*, *28*, *33*
geom_arrow_curve, 24
geom_arrow_curve(), *3*, *19*, *23*, *33*
geom_arrow_segment, 29
geom_arrow_segment(), *3*, *19*, *23*, *28*
GeomArrow, 15
GeomArrowChain *(*GeomArrow*)*, 15
GeomArrowCurve *(*GeomArrow*)*, 15
GeomArrowSegment *(*GeomArrow*)*, 15
ggarrow_extensions *(*GeomArrow*)*, 15
ggplot(), *16*, *20*, *25*, *30*
ggplot2::continuous_scale, *39*
ggplot2::discrete_scale, *9*, *39*
ggproto, *15*
gpar, *35*, *37*
graphical object, *35*, *38*
grob_arrow, 33, *37*
grob_arrow_curve, 36
group, *19*, *23*, *28*, *32*
guides(), *7*, *10*, *41*

key glyphs, *3*, *17*, *22*, *26*, *31*

lambda, *7*, *9*, *39, 40*
layer position, *17*, *21*, *26*, *30*
layer stat, *16*, *21*, *26*, *30*
layer(), *3*, *17*, *21, 22*, *26*, *30, 31*
legend key glyphs, *12*
linetype, *19*, *23*, *28*, *32*
linewidth, *19*, *23*, *28*, *32*

ornament constructors, *13*

ornament shapes, *13*
ornamentation, *7*

scale_arrow_fins_continuous *(*continuous_arrow_scales*)*, 5
scale_arrow_fins_discrete *(*discrete_arrow_scales*)*, 8
scale_arrow_head_continuous *(*continuous_arrow_scales*)*, 5
scale_arrow_head_discrete *(*discrete_arrow_scales*)*, 8
scale_arrow_mid_continuous *(*continuous_arrow_scales*)*, 5
scale_arrow_mid_discrete *(*discrete_arrow_scales*)*, 8
scale_colour_gradient2(), *40*
scale_colour_gradientn(), *40*
scale_resect, 38
scale_resect_continuous *(*scale_resect*)*, 38
scale_resect_discrete *(*scale_resect*)*, 38
scales::censor(), *40*
scales::extended_breaks(), *7*, *39*
scales::new_transform(), *7*, *40*
scales::pal_hue(), *9*
scales::rescale(), *40*
scales::squish(), *40*
scales::squish_infinite(), *40*

theme, *12*
theme(), *14*
transformation object, *6*, *39*

unit, *18*, *22*, *27*, *31*, *34*, *38*

whirlpool, 42

x, *19*, *23*, *28*, *32*
xend, *23*, *32*

y, *19*, *23*, *28*, *32*
yend, *23*, *32*