Package 'ggpointless'

March 9, 2026

Title Extra Geometries and Stats for 'ggplot2'

Version 0.2.0

Description A collection of layers for 'ggplot2'.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

URL https://flrd.github.io/ggpointless/,
https://github.com/flrd/ggpointless

BugReports https://github.com/flrd/ggpointless/issues

Depends ggplot2 (>= 4.0.0), R (>= 4.2.0)

Suggests knitr, covr, testthat (>= 3.0.0), rmarkdown, ragg, scales, vdiffr (>= 1.0.0), spelling, withr

Config/testthat/edition 3

VignetteBuilder knitr

Collate 'aaa.R' 'geom-area-fade.R' 'geom-catenary.R' 'geom-chaikin.R' 'geom-fourier.R' 'geom-lexis.R' 'geom-point-glow.R' 'geom-pointless.R' 'ggpointless-package.R' 'stat-catenary.R' 'stat-fourier.R' 'stat-lexis.R' 'stat-pointless.R' 'stat-chaikin.R'

Imports cli, grid, lifecycle, rlang, stats

Language en-GB

NeedsCompilation no

Author Markus Döring [aut, cre, cph]

Maintainer Markus Döring <m4rkus.doering@gmail.com>

Repository CRAN

Date/Publication 2026-03-09 11:50:02 UTC

Contents

`geom_area_fade` *Area Plots with Fading Linear Gradient*

Description

This geom behaves like `ggplot2::geom_area()` but uses `grid::linearGradient()` to create area plots. The gradient is always anchored at `y = 0`: maximum transparency there, fading to opaque at the data values. Opacity scales with the absolute distance from zero, so equal `|y|` values always receive the same alpha — full opacity is reached only at the extreme with the largest absolute value. This works for positive values, negative values, and groups that cross zero (where a three-stop gradient is used).

When `fill` is mapped to a variable (e.g. `aes(fill = z)`), the geom combines the horizontal colour gradient produced by ggplot2 with the vertical alpha fade, creating a two-dimensional gradient effect. This requires a device that supports Porter-Duff compositing (e.g. `ragg::agg_png()`, `grDevices::svg()`). On unsupported devices the geom falls back to a single-colour vertical fade and emits an informational message.

Usage

```
geom_area_fade(
  mapping = NULL,
  data = NULL,
  stat = "align",
  position = "stack",
  ...,
  alpha_fade_to = 0,
  alpha_scope = "global",
  orientation = NULL,
  outline.type = "upper",
  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.

`alpha_fade_to` A single finite number between 0 and 1. The alpha value at `y = 0` (the baseline). Defaults to `0` (fully transparent).

`alpha_scope` How to scale alpha across groups. `"global"` (default) computes the maximum absolute y value across **all** groups in the panel so that equal `|y|` always maps to equal alpha. `"group"` computes the maximum per group, giving each group the full alpha range independently — useful with `position = "identity"` when groups have very different amplitudes.

`orientation` The orientation of the layer. The default (`NA`) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting `orientation` to either `"x"` or `"y"`. See the *Orientation* section for more detail.

`outline.type` Which edges of the area to draw an outline on. One of `"upper"` (default), `"lower"`, `"both"` (`"upper"` and `"lower"`), `"full"` (closed polygon outline), or `"none"`. When no `colour` is specified explicitly the outline inherits the `fill` colour.

`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 `ggplot2::layer()` object that can be added to a `ggplot2::ggplot()`.

Orientation

This geom treats each axis differently and, thus, can thus have two orientations. Often the orientation is easy to deduce from a combination of the given mappings and the types of positional scales in use. Thus, ggplot2 will by default try to guess which orientation the layer should have. Under

rare circumstances, the orientation is ambiguous and guessing may fail. In that case the orientation can be specified directly using the `orientation` parameter, which can be either `"x"` or `"y"`. The value gives the axis that the geom should run along, `"x"` being the default orientation you would expect for the geom.

Aesthetics

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

- x
- y
- alpha → NA
- colour → via theme()
- fill → via theme()
- group → inferred
- linetype → via theme()
- linewidth → via theme()

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

References

Murrell, P. (2021). "Luminance Masks in R Graphics." Technical Report 2021-04, Department of Statistics, The University of Auckland. Version 1. https://www.stat.auckland.ac.nz/~paul/Reports/GraphicsEngine/masks/masks.html

Murrell, P. (2022). "Vectorised Pattern Fills in R Graphics." Technical Report 2022-01, Department of Statistics, The University of Auckland. Version 1. https://www.stat.auckland.ac.nz/~paul/Reports/GraphicsEngine/vecpat/vecpat.html

Murrell, P., Pedersen, T. L., and Skintzos, P. (2023). "Porter-Duff Compositing Operators in R Graphics." Department of Statistics, The University of Auckland. Version 1. https://www.stat.auckland.ac.nz/~paul/Reports/GraphicsEngine/compositing/compositing.html

Murrell, P. (2023). "Groups, Compositing Operators, and Affine Transformations in R Graphics." Technical Report 2021-02, Department of Statistics, The University of Auckland. Version 3. https://www.stat.auckland.ac.nz/~paul/Reports/GraphicsEngine/groups/groups.html

See Also

`ggplot2::geom_area()` for fully opaque area charts, the ggfx package for real magic.

Examples

```
library(ggplot2)
df1 <- data.frame(
  g = c("a", "a", "a", "b", "b", "b"),
  x = c(1, 3, 5, 2, 4, 6),
  y = c(2, 5, 1, 3, 6, 7)
)
```

```
a <- ggplot(df1, aes(x, y, fill = g)) +
  theme_minimal()

# default behaviour: opaque at data line, transparent at y = 0
# the outline colour remains unaffected
a + geom_area_fade()

# change overall opacity
a + geom_area_fade(alpha = .25)

# keep some opacity at the baseline
a + geom_area_fade(alpha_fade_to = .25)

# suppress the default upper outline
a + geom_area_fade(outline.type = "none")

# closed outline (all four edges)
a + geom_area_fade(outline.type = "full")

# horizontal orientation
a + geom_area_fade(aes(y, x), orientation = "y")

# disable stat alignment (useful when x values are already aligned)
a + geom_area_fade(stat = "identity")

# draw upper and lower outlines (no left/right edges)
a + geom_area_fade(outline.type = "both", stat = "identity")

# Use the "alpha_scope" argument to scale the alpha
# value of the gradients separately for each group
df2 <- data.frame(
  g = c("a", "a", "a", "b", "b", "b"),
  x = c(1, 3, 5, 2, 4, 6),
  y = c(1, 2, 1, 9, 10, 8)
)
b <- ggplot(df2, aes(x, y, fill = g)) +
  theme_minimal()

# alpha_scope = "group": each group uses the alpha range independently
b + geom_area_fade(
  alpha_scope = "group",
  position = "identity"
  )

# compare with the default where small groups appear washed out
# next to dominant groups, especially when position = "identity"
b + geom_area_fade(
  alpha_scope = "global", # default
  position = "identity"
  )

# geom_area_fade works with negative values too:
```

```
# the gradient fades towards y = 0 from both sides
d <- ggplot(df2, aes(x, y - mean(y))) +
  theme_minimal()
d + geom_area_fade()

# overwrite both fill and colour
d + geom_area_fade(
  fill = "#0833F5",
  colour = "#d77e7b",
  outline.type = "lower"
  )

# a 2D-gradient is produced when fill is mapped to a variable
# this may not work on all graphic devices, see vignette for details
d + geom_area_fade(
  aes(fill = y),
  colour = "#333333",
  outline.type = "both"
  )
```

---

geom_catenary *Catenary Curves and Arches*

Description

geom_catenary() draws a catenary curve (hanging chain) between successive points. geom_arch() draws an inverted catenary curve and is hence intended for people living on the southern hemisphere.

The shape follows the catenary equation: $y = a\ \cosh\left(\frac{x-h}{a}\right) + v$.

Usage

```
geom_catenary(
  mapping = NULL,
  data = NULL,
  stat = "catenary",
  position = "identity",
  ...,
  chain_length = NULL,
  sag = NULL,
  chainLength = lifecycle::deprecated(),
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_arch(
```

```
  mapping = NULL,
  data = NULL,
  stat = "arch",
  position = "identity",
  ...,
  arch_length = NULL,
  arch_height = NULL,
  arrow = NULL,
  arrow.fill = NULL,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_catenary(
  mapping = NULL,
  data = NULL,
  geom = "catenary",
  position = "identity",
  ...,
  chain_length = NULL,
  sag = NULL,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_arch(
  mapping = NULL,
  data = NULL,
  geom = "line",
  position = "identity",
  ...,
  arch_length = NULL,
  arch_height = 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)`).

`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.

`chain_length` Numeric vector of physical chain lengths. Recycled to the number of segments. If `NULL` and `sag` is also `NULL`, defaults to twice the Euclidean distance per segment. Can be mixed with `sag` by placing `NA` in the appropriate positions.

`sag` Numeric vector giving the vertical drop of the curve below the **lowest endpoint** of each segment. Takes precedence over `chain_length` when both are supplied for the same segment.

chainLength	**[Deprecated]** Use chain_length instead.
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().
arch_length	Numeric vector of arch lengths. Recycled to the number of segments. If NULL and arch_height is also NULL, defaults to twice the Euclidean distance per segment. Can be mixed with arch_height by placing NA in the appropriate positions.
arch_height	Numeric vector giving the vertical rise of the arch above the **highest endpoint** of each segment. Takes precedence over arch_length when both are supplied for the same segment.
arrow	Arrow specification, as created by grid::arrow().
arrow.fill	fill colour to use for the arrow head (if closed). NULL means use colour aesthetic.
lineend	Line end style (round, butt, square).
linejoin	Line join style (round, mitre, bevel).
linemitre	Line mitre limit (number greater than 1).
geom, stat	Override the default connection between geom_catenary() and stat_catenary(), or between geom_arch() and stat_arch().

Value

A ggplot2::layer() object that can be added to a ggplot2::ggplot().

Aesthetics

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

- x
- y
- alpha → NA
- colour → via theme()
- group → inferred
- linetype → via theme()
- linewidth → via theme()

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

See Also

geom_fourier() for fitting smooth curves to data via Fourier series, geom_chaikin() for smoothing paths via corner cutting. The catenary equation is described at https://en.wikipedia.org/wiki/Catenary.

Examples

```
library(ggplot2)

df <- data.frame(x = seq_len(4), y = c(1, 1, 0, 2))

# basic usage
p <- ggplot(df, aes(x, y)) + ylim(-3, NA) + geom_point(size = 3)
p + geom_catenary()

# Catenary with sag = 2, considered from lowest point of each segment
# recycled, if only a one value is provided
p + geom_catenary(sag = 2)
p + geom_catenary(sag = c(2, 1, 1))

# if sag and chain_length are provided for same segment(s), sag wins
p + geom_catenary(sag = c(2, 1, NA), chain_length = 10)

# Arch with height = 2, considered from highest point of each segment
p + geom_arch(arch_height = c(2, 1, 1))

# Rice house, see https://en.wikipedia.org/wiki/Rice_House,_Eltham
rice_house <- data.frame(x = c(0, 1.5, 2.5, 3.5, 5), y = c(0, 1, 1, 1, 0))
ggplot(rice_house, aes(x, y)) +
  geom_arch(arch_height = .15, lwd = 2) +
  geom_segment(aes(xend = x, yend = 0)) +
  geom_hline(yintercept = 0, colour = "forestgreen", linewidth = 3) +
  coord_equal()
```

---

geom_chaikin *Apply Chaikin's corner cutting algorithm to smooth a path*

Description

Chaikin's corner-cutting algorithm can be used to smooth sharp corners of a path.

Usage

```
geom_chaikin(
  mapping = NULL,
  data = NULL,
  stat = "chaikin",
  position = "identity",
  ...,
```

```
  mode = "open",
  iterations = 5,
  ratio = 0.25,
  arrow = NULL,
  arrow.fill = NULL,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_chaikin(
  mapping = NULL,
  data = NULL,
  geom = "path",
  position = "identity",
  ...,
  mode = "open",
  iterations = 5,
  ratio = 0.25,
  closed = lifecycle::deprecated(),
  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)).
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.

`mode` Character. Should the geom draw a closed polygon or an open path? Must be one of `"open"` (default) or `"closed"`.

`iterations` Integer. Number of iterations to apply between `1` and `10`. When `iterations = 0` the original data is unchanged so essentially this is the same as calling `ggplot2::geom_path()`; however this might be useful when you want to toggle smoothing on/off programmatically without removing the layer.

`ratio` Numeric. Cutting ratio must be a number between `0` and `1`. If `ratio > 0.5`, then it will be flipped to `1 - ratio`.

`arrow` Arrow specification, as created by `grid::arrow()`.

`arrow.fill` fill colour to use for the arrow head (if closed). `NULL` means use `colour` aesthetic.

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

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

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

`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().
geom, stat	Use to override the default connection between geom_chaikin() and stat_chaikin().
closed	**[Deprecated]** Use mode instead.

Details

Chaikin's corner cutting algorithm iteratively turns a jagged path into a smooth path.

The recursion formula starts from two vertices A and B, which represent a single corner of your path. From this, the algorithm derives two new points: one at the specified ratio when going from point A to point B, and one when going from B to A in the opposite direction. By default, a ratio of 0.25 results in two points: the first at 25% of point A and the other at 75% of point A (or 25% of point B). Those new points form a smoother path. Then the algorithm applies the same rule to each pair of new points. The rule is applied iterations times. The maximum number of iterations is 10, default is 5.

Value

A ggplot2::layer() object that can be added to a ggplot2::ggplot().

Aesthetics

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

- x
- y
- alpha → NA
- colour → via theme()
- group → inferred
- linetype → via theme()
- linewidth → via theme()

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

References

Chaikin, G. An algorithm for high speed curve generation. Computer Graphics and Image Processing 3 (1974), 346–349

See Also

The smoothr package offers tools to smooth and tidy spatial features

Examples

```
set.seed(42)
dat <- data.frame(
  x = seq.int(10),
  y = sample(15:30, 10)
)

p1 <- ggplot(dat, aes(x, y)) +
  geom_line(linetype = "12")

p1 +
  geom_chaikin()

p1 +
  geom_chaikin(iterations = 1)

triangle <- data.frame(x = c(0, 0, 1), y = c(0, 1, 1))
p2 <- ggplot(triangle, aes(x, y)) +
  geom_path(linetype = "12") +
  coord_equal()

# ratio lets you control the cutting amount
p2 + geom_chaikin(ratio = .1)
p2 + geom_chaikin(ratio = .5)

# mode controls whether the result is an open or closed shape
p2 + geom_chaikin(mode = "open")   # default
p2 + geom_chaikin(mode = "closed")
```

---

geom_fourier *Fourier Series Smoothing*

Description

`geom_fourier()` and `stat_fourier()` fit a truncated Fourier (discrete Fourier transform, DFT) series to the supplied x/y observations and render the reconstructed smooth curve. The data are first aggregated at duplicate x positions, interpolated to a uniform grid, optionally de-trended, transformed via `stats::fft()`, and then reconstructed from the requested number of harmonics.

Usage

```
geom_fourier(
  mapping = NULL,
  data = NULL,
```

```
  stat = "fourier",
  position = "identity",
  ...,
  n_harmonics = NULL,
  detrend = NULL,
  arrow = NULL,
  arrow.fill = NULL,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_fourier(
  mapping = NULL,
  data = NULL,
  geom = "line",
  position = "identity",
  ...,
  n_harmonics = NULL,
  detrend = 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)).
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.

`n_harmonics` Integer or NULL. Number of Fourier harmonics to retain. NULL (default) uses all harmonics up to the Nyquist limit, giving an interpolating fit. Smaller values produce smoother curves.

`detrend` Character string or `NULL`. De-trending method applied before the FFT; one of `"lm"`, `"loess"`, or `NULL` (default). See the *Detrending* section for details.

`arrow` Arrow specification, as created by `grid::arrow()`.

`arrow.fill` fill colour to use for the arrow head (if closed). `NULL` means use `colour` aesthetic.

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

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

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

`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()`.

`geom`, `stat` Override the default connection between `geom_fourier()` and `stat_fourier()`.

Value

A `ggplot2::layer()` object that can be added to a `ggplot2::ggplot()`.

Period convention

The DFT treats the input as one period of an infinitely repeating signal. The correct period for N uniformly-spaced samples with spacing Δx is $P = N \cdot \Delta x$, not $x_{max} - x_{min}$. Using the latter (a closed interval) implicitly maps the last sample to $t = 1$, which coincides with $t = 0$ of the next period, causing a boundary discontinuity and Gibbs-phenomenon ringing whenever the first and last y values differ. This implementation uses the half-open period.

Detrending

Before the FFT is applied the data can be de-trended so that slow, non-periodic trends do not dominate the low-frequency coefficients:

`NULL` **(default)** No de-trending; the raw signal is transformed.

`"lm"` Subtract a global ordinary-least-squares linear fit.

`"loess"` Subtract a LOESS smooth. Falls back to `"lm"` with a message if the group is too small for LOESS (fewer than 4 observations).

The trend is added back before the final curve is returned, so the output is always on the original y-scale.

Nyquist limit

The maximum number of harmonics recoverable from N observations is $\lfloor N/2 \rfloor$. Requesting more triggers a message and the limit is used instead.

Irregular spacing

The input data is linearly interpolated onto a uniform grid before the FFT. If the original x-spacing is highly irregular (e.g. monthly time series data), the interpolation may introduce artefacts in sparse regions. A message is emitted when the coefficient of variation of the x-spacing exceeds `0.5`.

Aesthetics

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

- `x`
- `y`
- `alpha` → `NA`

- colour → via theme()
- group → inferred
- linetype → via theme()
- linewidth → via theme()

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

See Also

`stats::fft()` for the underlying Fast Fourier Transform, `lm()` and `loess()` for the optional de-trending fits, `geom_catenary()` and `geom_chaikin()` for other curve-fitting geoms.

Examples

```
library(ggplot2)

n <- 50
df1 <- data.frame(
  x = seq(0, 1, length.out = n),
  y = sin(seq(0, 2 * pi, length.out = n)) + rnorm(n, sd = 0.2)
)

# Basic usage - Interpolating fit (all harmonics)
p <- ggplot(df1, aes(x, y)) +
  geom_point(alpha = 0.5)
p + geom_fourier()

# Use 1 harmonic only
p + geom_fourier(n_harmonics = 1)

# De-trending a linearly drifting signal
set.seed(2)
x <- seq(0, 4 * pi, length.out = n)
df2 <- data.frame(
  x = x,
  y = sin(x) + x * 0.3 + rnorm(n, sd = 0.15)
)

ggplot(df2, aes(x, y))  +
geom_point(alpha = 0.35) +
  geom_fourier(aes(colour = "detrend = NULL"), n_harmonics = 3) +
  geom_fourier(aes(colour = "detrend = \"lm\""), n_harmonics = 3,
               detrend = "lm")

# Multiple groups
set.seed(3)
x <- seq(0, 2 * pi, length.out = n/2)
df3 <- rbind(
  data.frame(x = x,
             y = sin(x) + rnorm(n/2, sd = 0.2),
             grp = "sine"),
```

```
  data.frame(x = x,
             y = cos(x) + rnorm(n/2, sd = 0.2),
             grp = "cosine")
)

ggplot(df3, aes(x, y, colour = grp)) +
  geom_point(alpha = 0.5) +
  geom_fourier()

# when the data is not uniformly-spaced, the Fourier
# curve will not pass through every data point
df4 <- data.frame(
  x = c(1:10, 19:20),
  y = sin(seq_len(12))
)

ggplot(df4, aes(x, y)) +
  geom_fourier()
```

`geom_lexis` *Lexis diagrams*

Description

This geom can be used to plot 45° lifelines for a cohort. Lexis diagrams are named after Wilhelm Lexis and used by demographers for more than a century.

Usage

```
geom_lexis(
  mapping = NULL,
  data = NULL,
  stat = "lexis",
  position = "identity",
  ...,
  point_show = TRUE,
  point_colour = NULL,
  gap_filler = TRUE,
  lineend = "round",
  linejoin = "round",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

stat_lexis(
  mapping = NULL,
```

```
  data = NULL,
  geom = "lexis",
  position = "identity",
  ...,
  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.

`point_show` logical. Should a point be shown at the end of each segment? `TRUE` by default.

`point_colour` colour of the endpoint point. If `NULL` (default), the group colour is used.

`gap_filler` logical. Should horizontal gap-filler segments be drawn? `TRUE` by default.

`lineend` line end style (round, butt, square)

`linejoin` line join style (round, mitre, bevel)

`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()`.

`geom` The geometric object to use to display the data for this layer. When using a `stat_*()` function to construct a layer, the `geom` argument can be used to override the default coupling between stats and geoms. The `geom` argument accepts the following:

- A `Geom` ggproto subclass, for example `GeomPoint`.
- A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `geom_point()`, give the geom as `"point"`.
- For more information and other ways to specify the geom, see the layer geom documentation.

Details

This geom draws 45° lines from the start to the end of a 'lifetime'. It is a combination of a segment, and a point. Besides y and yend coordinates this geom creates one additional variable called type in the layer data. You might want to map to an aesthetic with ggplot2::after_stat(), see Examples section and vignette("ggpointless") for more details.

Rows in your data with either missing x or xend values will be removed because your segments must start and end somewhere.

Value

A ggplot2::layer() object that can be added to a ggplot2::ggplot().

Aesthetics

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

- x
- y
- xend
- yend
- alpha → NA
- colour → "black"
- fill → NA
- group → inferred
- linetype → "solid"
- linewidth → 0.5
- shape → 19
- size → 1.5
- stroke → 0.5

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

Examples

```
df1 <- data.frame(
  key = c("A", "B", "B", "C", "D", "E"),
  start = c(0, 1, 6, 5, 6, 9),
  end = c(5, 4, 10, 9, 8, 11)
)
p <- ggplot(df1, aes(x = start, xend = end, color = key))
p +
  geom_lexis()
p +
  geom_lexis(gap_filler = FALSE)
p +
  geom_lexis(aes(linetype = after_stat(type)),
    point_show = FALSE
```

```
  )

# change point appearance
p + geom_lexis(
  point_colour = "black",
  size = 3,
  shape = 21,
  fill = "white",
  stroke = 1
)

# missing values will be removed
df2 <- data.frame(
  key = c("A", "B", "B", "C", "D"),
  start = c(0, 1, 7, 5, 6),
  end = c(5, 4, 13, 9, NA)
)
ggplot(df2, aes(x = start, xend = end, color = key)) +
  geom_lexis()

# Ideally, `x` values should be increasing, unlike
# in the next example
df3 <- data.frame(x = Sys.Date() - 0:2, xend = Sys.Date() + 1:3)
ggplot(df3, aes(x = x, xend = xend)) +
  geom_lexis()
```

`geom_pointless` *Emphasize some observations with points*

Description

This is a wrapper around `ggplot2::geom_point()` with the one additional argument: `location`. This geom aims to emphasis some observations but is not particularly useful on its own - hence its name - but hopefully in conjunction with `geom_line()` and friends, see examples.

Usage

```
geom_pointless(
  mapping = NULL,
  data = NULL,
  stat = "pointless",
  position = "identity",
  ...,
  location = "last",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

```
stat_pointless(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  location = "last",
  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.

`location` Position(s) to highlight: `"minimum"`, `"maximum"`, `"first"`, `"last"` (default), or `"all"`.

`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()`.

`geom` The geometric object to use to display the data for this layer. When using a `stat_*()` function to construct a layer, the `geom` argument can be used to override the default coupling between stats and geoms. The `geom` argument accepts the following:

- A `Geom` ggproto subclass, for example `GeomPoint`.
- A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `geom_point()`, give the geom as `"point"`.
- For more information and other ways to specify the geom, see the layer geom documentation.

Value

A ggplot2::layer() object that can be added to a ggplot2::ggplot().

Details

The location argument allows you to specify which observations should be highlighted. If location is "last", the default, a single point will be plotted at the last non-missing observation. The locations are determined in the order in which they appear in the data – like ggplot2::geom_path() does compared to ggplot2::geom_line().

Points may be plotted on top of one another; if location is set to "all", then the order in which points are plotted from top to bottom is: "first" > "last" > "minimum" > "maximum". Otherwise, the order is determined as specified in the location argument, which also then applies to the order legend key labels, see vignette("ggpointless") for more details.

Aesthetics

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

- x
- y
- alpha → NA
- colour → via theme()
- fill → via theme()
- group → inferred
- shape → via theme()
- size → via theme()
- stroke → via theme()

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

See Also

ggplot2::geom_point()

Examples

```
x <- seq(-pi, pi, length.out = 150)
y <- outer(x, 1:5, FUN = \(x, y) sin(x * y))

df1 <- data.frame(
  x = x,
  y = rowSums(y)
)

# not terribly useful on its own ...
p <- ggplot(df1, aes(x = x, y = y))
p + geom_pointless()
```

```
p + geom_pointless(location = "all")

# ... but in conjunction with geom_line(), hopefully
p <- p + geom_line()
p + geom_pointless(location = "all")
p + geom_pointless(location = c("first", "last"))
p + geom_pointless(location = c("minimum", "maximum"))

# The layer computes one additional variable, 'location',
# that you can map e.g. to colour
p + geom_pointless(
  aes(colour = after_stat(location)),
  location = "all",
  size = 3
)

# Example with missing first and last observations
set.seed(42)
df2 <- data.frame(x = 1:10, y = c(NA, sample(1:8), NA))
ggplot(df2, aes(x, y)) +
  geom_line() +
  geom_pointless(location = c("first", "last"))

# Change the order in which points are drawn when they overlap
df3 <- data.frame(x = 1:2, y = 1:2)

p <- ggplot(df3, aes(x = x, y = y)) +
  geom_path() +
  coord_equal()

# same as location = 'all'
p + geom_pointless(aes(colour = after_stat(location)),
  location = c("first", "last", "minimum", "maximum")
) +
  labs(subtitle = "same as location = 'all'")

# reversed custom order
p + geom_pointless(aes(colour = after_stat(location)),
  location = c("maximum", "minimum", "last", "first")
) +
  labs(subtitle = "custom order")

# same as location = 'all' again
p + geom_pointless(aes(colour = after_stat(location)),
  location = c("maximum", "minimum", "last", "first", "all")
) +
  labs(subtitle = "same as location = 'all' again")

# Use stat_pointless() with a geom other than "point"
set.seed(42)
df4 <- data.frame(x = 1:10, y = sample(1:10))
ggplot(df4, aes(x, y)) +
  geom_line() +
```

```
  geom_pointless(location = c("maximum", "minimum"), size = 3) +
  stat_pointless(
    aes(label = after_stat(y)),
    location = c("maximum", "minimum"),
    geom = "text",
    hjust = -1
  )

# Example using facets
# https://stackoverflow.com/q/29375169
p <- ggplot(economics_long, aes(x = date, y = value)) +
  geom_line() +
  facet_wrap(vars(variable), ncol = 1, scales = "free_y")

p + geom_pointless(
  aes(colour = after_stat(location)),
  location = c("minimum", "maximum"),
  size = 2
  )
```

`geom_point_glow` *Points that Glow*

Description

geom_point_glow is a version of (`geom_point()`) that adds a glow (radial gradient) behind each point.

Usage

```
geom_point_glow(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  glow_alpha = 0.5,
  glow_colour = NA,
  glow_size = NA,
  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.

`glow_alpha` Transparency of the glow between 0 (fully transparent) and 1 (fully opaque). Defaults to `0.5`.

`glow_colour` colour of the glow. If `NA` (default), it inherits the colour of the point itself.

`glow_size` Numerical value for the glow radius. If `NA` (default), it is calculated as three times the point size.

`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 `ggplot2::layer()` object that can be added to a `ggplot2::ggplot()`.

Aesthetics

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

- x
- y
- alpha → NA
- colour → via theme()
- fill → via theme()
- group → inferred
- shape → via theme()
- size → via theme()
- stroke → via theme()

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

References

Murrell, P. (2022). "Vectorised Pattern Fills in R Graphics." Technical Report 2022-01, Department of Statistics, The University of Auckland. Version 1. https://www.stat.auckland.ac.nz/~paul/Reports/GraphicsEngine/vecpat/vecpat.html

See Also

ggplot2::geom_point(), grid::radialGradient()

Examples

```
library(ggplot2)

# Basic usage
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point_glow()

# Customizing glow parameters (fixed for all points)
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point_glow(glow_colour = "#333", glow_alpha = 0.25, glow_size = 5) +
  theme_minimal()

# use the Geom with another Stat
ggplot(head(economics), aes(date, uempmed)) +
  geom_line() +
  stat_pointless(
    geom = "PointGlow",
    glow_colour = "tomato",
    glow_size = 10,
    location = c("first", "last")
)
```

Index