Package 'ggpointless'

May 21, 2026

Title Extra Geometries and Stats for 'ggplot2'

Version 0.3.0

Description A collection of layers for 'ggplot2'. Provides geoms built on linear and radial gradients from the 'grid' package, giving areas, bars, paths, rectangles, and ridgelines a fading or glowing visual effect. Also includes mathematically driven layers — catenary curves, Chaikin's corner-cutting smoothing (Chaikin, 1974, <doi:10.1016/0146-664X(74)90028-8>), and Fourier-series reconstruction — plus Lexis diagrams, isotype bar charts.

License MIT + file LICENSE

Encoding UTF-8

Config/roxygen2/markdown TRUE

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 covr, testthat (>= 3.0.0), ragg, vdiffr (>= 1.0.0), spelling, withr

Config/testthat/edition 3

Collate 'aaa.R' 'alpha-scope.R' 'geom-path-fade.R' 'geom-segment-fade.R' 'geom-abline-fade.R' 'geom-area-fade.R' 'geom-catenary.R' 'geom-chaikin.R' 'geom-col-fade.R' 'geom-curve-fade.R' 'geom-density-fade.R' 'geom-fourier.R' 'geom-freqpoly-fade.R' 'geom-gridline.R' 'geom-histogram-fade.R' 'geom-lexis.R' 'geom-line-fade.R' 'geom-point-glow.R' 'geom-pointless.R' 'geom-rect-fade.R' 'geom-ridgeline-fade.R' 'geom-ridgeline-density-fade.R' 'geom-ridgeline-freqpoly-fade.R' 'stat-bin-ridges.R' 'geom-ridgeline-histogram-fade.R' 'geom-step-fade.R' 'geom-unit-bar.R' 'geom-unit-histogram.R' 'ggpointless-package.R' 'position-ridgeline.R' 'stat-catenary.R' 'stat-chaikin.R' 'stat-fourier.R' 'stat-lexis.R' 'stat-pointless.R'

Imports cli, farver, grid, scales, lifecycle, rlang, stats

Language en-US

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Markus Döring [aut, cre, cph]

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

Repository CRAN

Date/Publication 2026-05-21 17:20:20 UTC

Contents

`alpha_scope` *Alpha scope: how the fade gradient is normalised across rows*

Description

Several fade geoms accept an `alpha_scope` argument that controls how the alpha gradient is *normalised* — that is, which subset of the rendered shapes the most-opaque end of the gradient is calibrated to. The vocabulary overlaps but is not identical across the geom families, because what counts as a meaningful "reference" differs between bars, areas, and ridgelines. This page is the consolidated reference.

What `alpha_scope` does:

Each fade geom interpolates the rendered alpha between two extremes: `alpha_fade_to` (the faded end, default fully transparent) and the row's aesthetic `alpha` value (the opaque end, default fully opaque). The peak of that interpolation is positioned somewhere on the data, and `alpha_scope` chooses where:

- Per-row scopes anchor the peak at the row's own extreme (each shape gets the full alpha range to itself).
- Group-relative scopes anchor the peak at the maximum within a discrete subset (rows in the same subset share an alpha range; smaller rows appear proportionally fainter).
- Layer-wide scopes anchor the peak at the maximum across the whole layer (or all panels under faceting).

Vocabulary by geom family:

The accepted values and defaults are family-specific:

Area family (`geom_area_fade()`, `geom_density_fade()`, `geom_freqpoly_fade()`) Default `"global"`. Allowed: `"global"`, `"group"`. `"global"` scales every group to the layer-wide maximum `|y|`, so equal `|y|` always maps to equal alpha. `"group"` lets each group use the full alpha range independently.

Bar family (`geom_col_fade()`, `geom_bar_fade()`) Default `"bar"`. Allowed: `"bar"`, `"group"`, `"x"`, `"y"`, `"fill"`, `"colour"`, `"global"`. `"bar"` gives every bar its own range (every peak hits full opacity); `"x"` / `"y"` normalise within a position-axis cluster (useful for stacks and dodges); `"fill"` / `"colour"` normalise within a colour class; `"global"` normalises layer-wide; `"group"` falls back to ggplot2's `data$group`.

Histogram family (`geom_histogram_fade()`) Default `"bar"`. Allowed: `"bar"`, `"group"`, `"bin"`, `"fill"`, `"colour"`, `"global"`. The shared bar-family scopes carry over, but `"x"` / `"y"` are **not** accepted: they key on `round(data$x|y)` which is meaningless on a continuous binned axis. Use `"bin"` instead — it normalises within each bin (every cluster of dodged bars in one bin shares an alpha range), recovering the per-cluster intent that `"x"` / `"y"` give on `geom_col_fade()` / `geom_bar_fade()`.

Ridgeline family (`geom_ridgeline_fade()`, `geom_ridgeline_density_fade()`) Default `"group"`. Allowed: `"group"`, `"global"`. `"group"` lets each ridge use the full alpha range independently. `"global"` scales relative to the tallest ridge in the entire layer (incl. across facets), so shorter ridges fade in proportion.

Why the same name means different things:

`"global"` always means *the layer-wide maximum* — but the *maximum of what* depends on the geom: `|y|` for areas, `|y|` (or `|x|` under `orientation = "y"`) for bars and histograms, and the tallest ridge `height` for ridgelines. `"group"` always means *normalise per* `ggplot2` *group* (`data$group`, the interaction of all discrete aesthetics). For ridgelines this is effectively "per ridge" because each `data$group` corresponds to one ridge. The defaults above are chosen so that the most common usage of each family produces sensible output without explicit `alpha_scope`.

See Also

`geom_area_fade()`, `geom_col_fade()`, `geom_ridgeline_fade()` for the geom-side documentation that drives the actual rendering.

`geom_abline_fade` *Reference Lines with a Fading Gradient*

Description

These geoms draw reference lines – horizontal, vertical, or diagonal – with an alpha gradient along the line so that one or both ends fade to transparent.

Like their ggplot2 counterparts, these geoms can be used as annotations by passing `slope`/`intercept`, `yintercept`, or `xintercept` as arguments directly. In that case the line is constant across facets. To vary lines across facets, supply your own `data` and `mapping`.

Usage

```
geom_abline_fade(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  ...,
  slope,
  intercept,
  alpha_fade_to = 0,
  fade_direction = "start",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = FALSE
)

geom_hline_fade(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  yintercept,
  alpha_fade_to = 0,
  fade_direction = "start",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = FALSE
)

geom_vline_fade(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
```

```
  ...,
  xintercept,
  alpha_fade_to = 0,
  fade_direction = "start",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = FALSE
)
```

Arguments

mapping	Set of aesthetic mappings created by ggplot2::aes().
data	A data frame, or other object, to override the plot data.
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.
...	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 the fading end(s). Defaults to 0 (fully transparent).

`fade_direction` Which end(s) of the line fade. A character vector containing `"start"`, `"end"`, or both `c("start", "end")`. Defaults to `"start"`. For horizontal lines, `"start"` fades the visual left end; for vertical lines, `"start"` fades the visual bottom end; for diagonal lines, `"start"` fades toward the visual left edge of the panel. See *Gradient direction and reversed scales* for behaviour under `ggplot2::scale_x_reverse()` / `ggplot2::scale_y_reverse()`.

`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()`.

`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.

`xintercept`, `yintercept`, `slope`, `intercept`
Parameters that control the position of the line. If these are set, `data`, `mapping` and `show.legend` are overridden.

Details

Gradient direction and reversed scales:

The fade direction is defined in **visual** (panel) space, not in data space. `fade_direction = "start"` always makes the *start* of the line transparent – for horizontal lines this is the visual left edge; for vertical lines, the visual bottom edge – regardless of whether the x or y scale is reversed. So `fade_direction = "start"` means "transparent at the left/bottom panel edge" irrespective of axis direction.

If you *do* want the gradient to track the data direction – making the low-value end transparent even when it appears on the visual right – pass `fade_direction = "end"` to override the default:

```
# On a reversed x-axis, "end" fades toward larger x values (visual left)
p + scale_x_reverse() +
  geom_hline_fade(yintercept = 20, fade_direction = "end")
```

Non-linear coordinate systems (coord_polar / coord_radial):

Under non-linear coordinate systems the "line" is conceptually a curve in device space:

- `geom_hline_fade()` traces a **circle** at the given `yintercept` (constant radius).
- `geom_vline_fade()` traces a **ray** at the given `xintercept` (constant angle).
- `geom_abline_fade()` traces a **curve** that spirals or arcs based on the slope/intercept.

The fade is applied along the **path length** of that curve, so `fade_direction = "start"` fades the beginning of the traced path. For an `hline_fade()` circle the "start" is the first vertex of the traced circle (at the leftmost angle of the coord system, 12 o'clock by default), which may not match intuition from cartesian use – pass `fade_direction = "end"` to reverse.

Internally, the line is subdivided into a dense set of vertices in data space before the coord transform so the curve appears smooth. This is transparent to the user but explains why the grob carries more vertices than in the linear case.

Value

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

Aesthetics

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

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

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

References

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

`geom_segment_fade()` for fading line segments with explicit endpoints, `geom_path_fade()` and `geom_line_fade()` for connected paths; `ggplot2::geom_abline()`, `ggplot2::geom_hline()`, and `ggplot2::geom_vline()` for their ggplot2 originals.

Examples

```
library(ggplot2)

p <- ggplot(mtcars, aes(wt, mpg)) + geom_point()

# Horizontal reference line, fading from the left
p + geom_hline_fade(yintercept = 20, linewidth = 1.5)

# Vertical line fading at both ends
p + geom_vline_fade(xintercept = 3, linewidth = 1.5,
                    fade_direction = c("start", "end"))

# Diagonal line of best fit, fading from the left
coefs <- coef(lm(mpg ~ wt, data = mtcars))
p + geom_abline_fade(intercept = coefs[1], slope = coefs[2], linewidth = 1.5)
```

`geom_area_fade` *Areas, Densities, and Frequency Polygons with Fading Gradient*

Description

`geom_area_fade()` 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.

`geom_density_fade()` computes and draws a kernel density estimate – a smoothed version of the histogram – with the same vertical alpha gradient as `geom_area_fade()`. Under the hood this is GeomAreaFade paired with `ggplot2::stat_density()`, so all smoothing parameters (`bw`, `adjust`, `kernel`, `bounds`, ...) are forwarded to the stat.

`geom_freqpoly_fade()` draws a frequency polygon (like `ggplot2::geom_freqpoly()`) filled with the same linear gradient as `geom_area_fade()`.

Usage

```
geom_area_fade(
  mapping = NULL,
  data = NULL,
  stat = "align",
  position = "stack",
```

```
  ...,
  alpha_fade_to = 0,
  alpha_scope = "global",
  orientation = NA,
  outline.type = "upper",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_density_fade(
  mapping = NULL,
  data = NULL,
  stat = "density",
  position = "identity",
  ...,
  bw = "nrd0",
  adjust = 1,
  kernel = "gaussian",
  bounds = c(-Inf, Inf),
  alpha_fade_to = 0,
  alpha_scope = "global",
  orientation = NA,
  outline.type = "upper",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_freqpoly_fade(
  mapping = NULL,
  data = NULL,
  stat = "bin",
  position = "identity",
  ...,
  binwidth = NULL,
  bins = NULL,
  alpha_fade_to = 0,
  alpha_scope = "global",
  orientation = NA,
  pad = TRUE,
  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	Use to override the default connection between geom_freqpoly_fade() and stat_bin().
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().

bw — The smoothing bandwidth to be used. If numeric, the standard deviation of the smoothing kernel. If character, a rule to choose the bandwidth, as listed in stats::bw.nrd().

adjust — A multiplicate bandwidth adjustment. This makes it focused on giving the kernel bandwidth more or less smoothing.

kernel — Kernel. See stats::density() for more details.

bounds — Known lower and upper bounds for the variable. Default is c(-Inf, Inf).

binwidth — Width of each bin in data units. When supplied, takes precedence over bins. Forwarded to ggplot2::stat_bin().

bins — Number of bins. Overridden by binwidth. Defaults to 30. Forwarded to ggplot2::stat_bin().

pad — If TRUE, adds empty bins at either end of x. This ensures frequency polygons touch 0. Defaults to FALSE.

Value

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

Coordinate systems

`geom_area_fade()`, `geom_density_fade()`, and `geom_freqpoly_fade()` only support linear gradients. When used with `ggplot2::coord_polar()` or `ggplot2::coord_radial()`, they fall back to standard area rendering (equivalent to `ggplot2::geom_area()`), which means no gradient fill is added. A warning is emitted in this case.

alpha_scope = "global" under faceting

`alpha_scope = "global"` ties opacity to absolute height across the whole layer, so two ridges / areas / bars of equal height render at equal alpha regardless of which panel they're in. This is meaningful only when panels share a common y scale. Under `facet_wrap(scales = "free_y")` (or `facet_grid(rows = ..., scales = "free")`) each panel rescales y independently, so the visual height of a shape no longer reflects its data height; the alpha encoding then conflicts with what the eye reads from the panel size. For comparable alpha across free-y panels you have two options: stick to the default `scales = "fixed"`, or accept that under free scales `alpha_scope = "group"` is the more honest choice (each shape independently uses its own alpha range).

Legend key under coord_flip

The legend key glyph always shows the canonical (data-axis) fade direction – vertical for the default orientation, horizontal under `orientation = "y"`. Under `ggplot2::coord_flip()` the rendered geom rotates correctly but the legend key does *not*: ggplot2's legend builder is coord-independent by design (`draw_key` has no access to the coord). For a legend key that matches a horizontal layout, prefer `aes(y = ...)` with auto-detected `orientation = "y"` over `aes(x = ...) + coord_flip()`.

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()`, `ggplot2::geom_density()`, `ggplot2::geom_freqpoly()`.

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()

# With 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"
  )

# Negative values are supported 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"
  )

# Basic density curve: opaque at the peak, fully transparent at the baseline.
```

```
ggplot(diamonds, aes(carat)) +
  geom_density_fade()

# Map the values to y to flip the orientation
ggplot(diamonds, aes(y = carat)) +
  geom_density_fade()

# `alpha_fade_to` controls the alpha at the baseline.
# The default `0` is fully transparent; raise it to keep some
# opacity at the floor.
ggplot(diamonds, aes(carat)) +
  geom_density_fade(alpha_fade_to = 0.2)

# Multiple groups via `fill`. With the default `alpha_scope = "global"`
# the tallest peak in the layer reaches full opacity; shorter peaks fade
# in proportion. `xlim()` trims the long tails for clarity.
ggplot(diamonds, aes(depth, fill = cut)) +
  geom_density_fade() +
  xlim(55, 70)

# Switch to `alpha_scope = "group"` so every
# area hits full opacity independently
ggplot(diamonds, aes(depth, fill = cut)) +
  geom_density_fade(alpha_scope = "group") +
  xlim(55, 70)

# You can use position = "fill" to produce a conditional density estimate
ggplot(diamonds, aes(carat, after_stat(count), fill = cut)) +
  geom_density_fade(position = "fill")

# Basic frequency polygon with fading gradient
ggplot(faithful, aes(waiting)) +
  geom_freqpoly_fade(
    colour = "#3b528b",
    bins = 20
  ) +
  theme_minimal()

# Rather than stacking histograms, compare frequency polygons
ggplot(iris, aes(Sepal.Length, fill = Species, colour = Species)) +
  geom_freqpoly_fade(
    alpha = 0.8,
    position = "identity",
    bins = 20
  ) +
  scale_fill_viridis_d() +
  scale_colour_viridis_d() +
  theme_minimal()
```

---

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 = "arch",
  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` **[Superseded]** 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

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, a number in [0, 1]. The conventional Chaikin range is [0, 0.5]: values outside that range are flipped to 1 - ratio with a warning, because ratios above 0.5 are not geometrically meaningful in the usual corner-cutting sense. For **closed** paths the flipped result has the same vertex set as the input ratio (with a cyclic shift). For **open** paths it is a different curve – prefer increasing iterations if you want stronger smoothing.

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 — **[Superseded]** 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. M. (1974). An algorithm for high-speed curve generation. *Computer Graphics and Image Processing*, 3(4), 346–349. doi:10.1016/0146664X(74)900288

See Also

The smoothr package offers tools to smooth and tidy spatial features

Examples

```
library(ggplot2)
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_col_fade` *Bar Charts with Fading Gradient and Rounded Corners*

Description

`geom_col_fade()` and `geom_bar_fade()` draw bar charts like `ggplot2::geom_col()` / `ggplot2::geom_bar()` but with options to add an alpha gradient that fades from opaque at the peak of each bar to transparent at its baseline; and rounded corners are supported via `grid::roundrectGrob()`, controlled by the `radius` argument.

Usage

```
geom_col_fade(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "stack",
  ...,
  alpha_fade_to = 0,
  alpha_scope = "bar",
  orientation = NA,
  radius = grid::unit(0, "pt"),
  lineend = "butt",
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_bar_fade(
  mapping = NULL,
  data = NULL,
  stat = "count",
  position = "stack",
  ...,
  alpha_fade_to = 0,
  alpha_scope = "bar",
  orientation = NA,
```

```
  radius = grid::unit(0, "pt"),
  lineend = "butt",
  linejoin = "mitre",
  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, as a string. `geom_col_fade()` uses `"identity"` (no transform), `geom_bar_fade()` uses `"count"`, and `geom_histogram_fade()` uses `"bin"`.
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 the baseline of each bar. Defaults to `0` (fully transparent).

`alpha_scope` How to choose the per-bar reference height that the gradient normalises against. One of:

- `"bar"` (default): each bar uses its own height – every peak hits full opacity.
- `"global"`: every bar normalises to the tallest bar in the entire layer, **including across facet panels**. The single tallest bar reaches full opacity; all others fade in proportion.
- `"x"` / `"y"`: every bar normalises to the tallest bar at the same discrete x (or y) position. Useful for highlighting the leader within a stack or dodge cluster. `"x"` is for vertical bars; `"y"` is for horizontal bars (`coord_flip()` / `orientation = "y"`). Mismatched orientation aborts with a hint.
- `"group"`: every bar normalises to the tallest bar in its ggplot2 group (`data$group` – the interaction of all discrete aesthetics). Most useful with explicit `aes(group = ...)`; with the typical `aes(x = factor, fill = factor)` layout it degenerates to `"bar"` because every (x, fill) pair is its own group.
- `"fill"` / `"colour"`: every bar normalises to the tallest bar with the same fill (or colour). Bars sharing a fill share an alpha range across x and across facet panels.

`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.

`radius` Corner radius passed to `grid::roundrectGrob()`. A `grid::unit()` object (e.g. `unit(4, "pt")`); a bare number is interpreted as points. Defaults to `unit(0, "pt")` (matching `geom_bar()` / `geom_col()`).

`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()`.

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.

alpha_scope = "global" under faceting

`alpha_scope = "global"` ties opacity to absolute height across the whole layer, so two ridges / areas / bars of equal height render at equal alpha regardless of which panel they're in. This is meaningful only when panels share a common y scale. Under `facet_wrap(scales = "free_y")` (or `facet_grid(rows = ..., scales = "free")`) each panel rescales y independently, so the visual height of a shape no longer reflects its data height; the alpha encoding then conflicts with what the eye reads from the panel size. For comparable alpha across free-y panels you have two options: stick to the default `scales = "fixed"`, or accept that under free scales `alpha_scope = "group"` is the more honest choice (each shape independently uses its own alpha range).

Legend key under coord_flip

The legend key glyph always shows the canonical (data-axis) fade direction – vertical for the default orientation, horizontal under `orientation = "y"`. Under `ggplot2::coord_flip()` the rendered geom rotates correctly but the legend key does *not*: ggplot2's legend builder is coord-independent by design (`draw_key` has no access to the coord). For a legend key that matches a horizontal layout, prefer `aes(y = ...)` with auto-detected `orientation = "y"` over `aes(x = ...) + coord_flip()`.

Aesthetics

`geom_col_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()`

- `width` → `0.9`

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_bar()` and `ggplot2::geom_col()` for fully opaque bar charts, `geom_histogram_fade()` for the histogram chart equivalent.

Examples

```
library(ggplot2)

df <- data.frame(
  x = c("A", "B", "C", "D", "E"),
  y = c(3, 4, -2, -0.5, 1)
)

ggplot(df, aes(x, y)) +
  geom_col_fade() +
  theme_minimal()

# Rounded bar charts are supported too
ggplot(df, aes(x, y)) +
  geom_col_fade(radius = unit(10, "pt")) +
  theme_minimal()

# Start at 90% opacity and keep some opacity at the baseline
ggplot(df, aes(x, y)) +
  geom_col_fade(
    alpha = 0.9,
    alpha_fade_to = 0.1
  ) +
  theme_minimal()

# Horizontal bars are supported
ggplot(df, aes(y, x)) +
  geom_col_fade() +
  theme_minimal()


# Multiple groups with different alpha scopes
p <- ggplot(diamonds, aes(color, fill = cut)) +
  scale_fill_viridis_d(guide = "none") +
  labs(x = NULL, y = NULL) +
```

```
  theme_minimal()

# By default each bar has its own alpha scope
p + geom_bar_fade()

# With alpha_scope = "x", bars at same x aesthetic share same alpha scope
p + geom_bar_fade(alpha_scope = "x")

# With alpha_scope = "fill", alpha is defined by what is mapped to the fill aesthetic
p + geom_bar_fade(alpha_scope = "fill")

# With alpha_scope = "global", the maximum absolute
# value across the whole dataset is fully opaque
p + geom_bar_fade(alpha_scope = "global")

# same examples for position dodge with varying alpha scope:
p + geom_bar_fade(alpha_scope = "bar", position = "dodge")
p + geom_bar_fade(alpha_scope = "x", position = "dodge")
p + geom_bar_fade(alpha_scope = "fill", position = "dodge")
p + geom_bar_fade(alpha_scope = "global", position = "dodge")

# Polar coordinates are supported too, if you need it
ggplot(diamonds, aes(x = factor(1), fill = cut)) +
  geom_bar_fade(width = 1) +
  coord_polar(theta = "y") +
  theme_void()
```

`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 = "fourier",
  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 hit every data point exactly
ggplot(head(economics, 25), aes(date, unemploy)) +
  geom_fourier() +
```

```
  geom_point()  +
  geom_curve_fade(
    data = data.frame(
      x    = as.Date("1967-10-01"),
      xend = as.Date("1968-01-01"),
      y    = 2750,
      yend = 2850
    ),
    aes(x = x, xend = xend, y = y, yend = yend),
    arrow = arrow(),
    colour = "tomato"
    )

# ... in extreme cases a warning is emitted
df4 <- data.frame(
  x = c(1:10, 19:20),
  y = sin(seq_len(12))
)

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

`geom_gridline` *(Grid) Lines Drawn on Top of Other Layers*

Description

`geom_gridline()` draws horizontal and vertical grid lines as a regular ggplot2 layer, so they appear *above* bar charts or any other geom in your plot.

The line positions are read directly from the trained scale (via `panel_params`), and the line properties are read from the theme; so `geom_gridline()` always matches the grid line positions and properties by default — but you are free to override every property, of course.

This was inspired by Observable Plot's Grid mark: https://observablehq.com/plot/marks/grid#grid-mark.

Usage

```
geom_gridline(
  mapping = NULL,
  data = NULL,
  grids = "y",
  lines = "major",
  colour = NULL,
  linewidth = NULL,
  linetype = NULL,
  lineend = NULL,
```

```
  alpha = NA,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = FALSE,
  ...
)
```

Arguments

mapping, data	Present for ggplot2 layer-signature compatibility but unused: geom_gridline() reads break positions from the panel scales rather than from a layer mapping or data. Passing a non-default value for either emits a warning.
grids	Character vector specifying which "grid" lines to draw: "x", "y" (default), or c("x", "y") for both.
lines	Character vector specifying which line type(s) to draw: "major" (default), "minor", or c("major", "minor") for both.
colour, linewidth, linetype, lineend	Line aesthetics. Default NULL inherits each property by walking ggplot2's documented theme chain: panel.grid.major.x (or .y) → panel.grid.major → panel.grid → line. Pass explicit values to override individual properties.
alpha	Opacity in [0, 1]. Default NA (fully opaque).
na.rm	If FALSE (default) missing values are silently dropped.
show.legend	Logical. Should this layer appear in the legends? Default FALSE (grid lines rarely need a legend entry).
inherit.aes	If FALSE, overrides the default aesthetics.
...	Other arguments passed to ggplot2::layer().

Value

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

Line properties

By default geom_gridline() inherits each property by walking ggplot2's documented theme chain: panel.grid.major.x (or .y) → panel.grid.major → panel.grid → line so that by default lines look exactly like the grid would. If you blank panel.grid the layer picks up styling from theme(line = ...). Pass an explicit colour to override, see *Examples*.

Rendering order

geom_gridline() follows a specific Z-order convention to ensure maximum visibility:

1. Major grid lines are always drawn **on top** of minor grid lines.
2. Y-aesthetic grid lines are drawn **on top** of X-aesthetic grid lines.

This means the final drawing sequence (from bottom to top) is: Minor X, Minor Y, Major X, Major Y.

See Also

ggplot2::geom_hline(), ggplot2::geom_vline() for fixed reference lines; ggplot2::theme() for controlling the underlying panel grid.

Examples

```
library(ggplot2)

# Basic example - geom_gridline() is just another layer
# plotted in the order you add them to your ggplot
p <- ggplot(mpg, aes(class)) +
  geom_bar()
p + geom_gridline()

# Note: geom_gridline() does not touch the theme. To draw only the layer's
# lines (no theme grid underneath), blank the panel grid yourself.
bf <- theme_grey()$panel.background@fill
p +
  geom_gridline(linewidth = 0.4, colour = bf) +
  theme_minimal() +
  theme(panel.grid = element_blank())

# Horizontal bars: flip axes, draw gridlines atop x-grid at custom breaks
ggplot(mpg, aes(y = class)) +
  geom_bar() +
  geom_gridline(grids = "x", colour = "tomato", linewidth = 2) +
  scale_x_continuous(breaks = c(5, 10, 20, 40))

# Line properties are inherited from theme
# their positions from the scale
p +
  geom_gridline() +
  scale_y_continuous(breaks = c(10, 20)) +
  theme_gray(paper = "cornsilk", ink = "navy")

# When you explicitly set properties in geom_gridline
# they will overwrite theme properties
p +
  geom_gridline(lines = c("major", "minor")) +
  scale_y_sqrt(breaks = c(10, 20)) +
  theme_gray(paper = "cornsilk", ink = "navy")
```

---

geom_histogram_fade *Histograms with Fading Gradient*

Description

Visualise the distribution of a single continuous variable as a histogram with a fading alpha gradient. Counts are drawn with rounded, gradient-filled bars (like geom_col_fade() paired with

ggplot2::stat_bin()). Accepts all binning parameters forwarded to ggplot2::stat_bin() (bins, binwidth, center, boundary, ...).

Usage

```
geom_histogram_fade(
  mapping = NULL,
  data = NULL,
  stat = "bin",
  position = "stack",
  ...,
  binwidth = NULL,
  bins = NULL,
  alpha_fade_to = 0,
  alpha_scope = "bar",
  orientation = NA,
  radius = grid::unit(0, "pt"),
  lineend = "butt",
  linejoin = "mitre",
  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	Use to override the default connection between geom_histogram_fade()/geom_freqpoly_fade() and stat_bin().
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.

`binwidth` Width of each bin in data units. When supplied, takes precedence over `bins`. Forwarded to `ggplot2::stat_bin()`.

`bins` Number of bins. Overridden by `binwidth`. Defaults to 30. Forwarded to `ggplot2::stat_bin()`.

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

`alpha_scope` How to choose the per-bar reference height that the gradient normalises against. The histogram family's vocabulary differs from `geom_col_fade()` / `geom_bar_fade()` because `x` is continuous (a binned variable) rather than a discrete category:

- `"bar"` (default), `"group"`, `"fill"`, `"colour"`, `"global"` – same meaning as in `geom_col_fade()`.
- `"bin"` – every bar in the same bin shares an alpha range. Useful under `position = "dodge"` for highlighting the tallest group within each bin (e.g. "which species dominates each Sepal.Width bin").

The `"x"` / `"y"` scopes accepted by `geom_col_fade()` are **not** available here – on a continuous binned axis they would key on `round(data$x)`, which buckets bins by integer rounding rather than by bin identity. Use `"bin"` for the per-bin meaning.

`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.

radius	Corner radius passed to grid::roundrectGrob(). A grid::unit() object (e.g. unit(4, "pt")); a bare number is interpreted as points. Defaults to unit(0, "pt") (matching geom_bar() / geom_col()).
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().

Value

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

Aesthetics

geom_histogram_fade() understands the same aesthetics as geom_col_fade() (it is GeomHistogramFade, a subclass of GeomColFade, paired with ggplot2::stat_bin()). See ?geom_col_fade for the full aesthetics table.

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.

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

geom_col_fade() / geom_bar_fade() for the bar-chart equivalents, geom_area_fade() for the general area-fade geom, ggplot2::geom_histogram() and ggplot2::geom_freqpoly() for the non-fading originals.

Examples

```
library(ggplot2)

# By default each bar has its own alpha scope
p <- ggplot(faithful, aes(waiting))
p + geom_histogram_fade()

# when all bars shall share the same alpha scope,
# set alpha_scope = "global"
p +
  geom_histogram_fade(
    alpha_scope = "global",
    alpha = 0.75,
    alpha_fade_to = 0.1,
    radius = unit(3, "pt"),
    colour = "#333333"
  ) +
  theme_minimal()

# Stacked histogram with groups
ggplot(iris, aes(Sepal.Length, fill = Species)) +
  geom_histogram_fade(alpha_fade_to = 0.25) +
  theme_minimal()

# Stacked histogram with groups and global alpha scope
ggplot(iris, aes(Sepal.Length, fill = Species)) +
  geom_histogram_fade(
    alpha_fade_to = 0.25,
    alpha_scope = "global"
  )

# Per-fill scope under position = "dodge": each fill cluster has its own
# alpha range, so the tallest sub-bar in every bin reaches full opacity.
ggplot(iris, aes(Sepal.Width, fill = Species)) +
  geom_histogram_fade(
    position = "dodge",
    bins = 10,
    alpha_scope = "fill"
  )
```

`geom_lexis` *Lexis diagrams*

Description

This geom can be used to plot 45 deg 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 deg 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* 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 → via theme()
- fill → via theme()
- group → inferred
- linetype → via theme()
- linewidth → via theme()

- shape → via theme()
- size → via theme()
- stroke → via theme()

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

Examples

```
library(ggplot2)
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, colour = 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, colour = 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_path_fade *Paths and Lines with a Fading Gradient*

Description

`geom_path_fade()` connects observations in the order they appear in the data (like `ggplot2::geom_path()`) and applies a linear alpha gradient so that one or both ends of the path fade to transparent.

`geom_line_fade()` is identical to `geom_path_fade()` but sorts observations by x before drawing (like `ggplot2::geom_line()`).

`geom_step_fade()` is identical to `geom_path_fade()` but draws a staircase-step path (like `ggplot2::geom_step()`). The `direction` argument controls the step shape: `"hv"` (horizontal then vertical, the default), `"vh"` (vertical then horizontal), or `"mid"` (step at the midpoint).

Usage

```
geom_path_fade(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  alpha_fade_to = 0,
  fade_direction = "start",
  alpha_mode = "auto",
  arrow = NULL,
  arrow.fill = NULL,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_line_fade(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  alpha_fade_to = 0,
  fade_direction = "start",
  alpha_mode = "auto",
  arrow = NULL,
  arrow.fill = NULL,
  lineend = "butt",
```

```
  linejoin = "round",
  linemitre = 10,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_step_fade(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  alpha_fade_to = 0,
  fade_direction = "start",
  alpha_mode = "auto",
  direction = "hv",
  arrow = NULL,
  arrow.fill = NULL,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  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 the fading end(s). Defaults to `0` (fully transparent).

`fade_direction` Which end(s) of the path fade out. A character vector containing `"start"`, `"end"`, or both `c("start", "end")`. Defaults to `"start"`. Invalid values are silently dropped with a warning; if nothing valid remains, falls back to `"start"`.

`alpha_mode` How the alpha gradient is rendered. One of `"step"` (default) or `"gradient"`. See section **Rendering modes** 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().
direction	Direction of the steps. One of "hv" (default, horizontal then vertical), "vh" (vertical then horizontal), or "mid" (step at the midpoint between adjacent x values).

Value

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

Rendering modes

The alpha_mode argument controls how the alpha gradient is rendered. All three values compute a target alpha at each vertex from cumulative distance along the path – the fade follows the path regardless of direction or shape.

"auto" (default): pick per sub-path based on vertex count n. n <= 50 -> "gradient" (smooth within-segment fade; protects the common "few thick segments" case). n > 50 -> "step" (stepping is invisible at that density and gradient's per-segment cost grows linearly with n). The threshold is chosen so that worst-case "gradient" render time stays under ~0.4 s per panel. Resolved per sub-path – a multi-group plot can mix modes. Users who want deterministic rendering (snapshots, reproducible builds) should set "step" or "gradient" explicitly.

"step": each segment is drawn inside a viewport clipped to its bisector-cut polygon, carrying a single uniform alpha – the average of its two endpoint alphas. The fade is an illusion created by stepping through discrete alpha values across adjacent segments. Fast (~0.4 s for a 200-point path).

"gradient": each segment's clipped viewport contains a panel-sized rectGrob with its own direction-aligned linearGradient fill. The alpha transitions smoothly within each segment – a real continuous gradient, not a step approximation. Slower (~1 s for 200 points, ~4 s for 1000 points); scales linearly with n and multiplies under facets.

All modes require a device that supports clipping paths (e.g. ragg::agg_png(), grDevices::svg()). On devices that don't, the geom falls back to per-segment segmentsGrob rendering with combined alpha (no linejoin).

The base grDevices::pdf() and grDevices::postscript() devices *advertise* clipping-path support but have an upstream R heap-corruption bug at dev.off() once enough clipping or gradient operations accumulate (reproducible with pure grid on R 4.5.3). To keep these devices safe the geom routes through the flat per-segment fallback on pdf() / postscript() regardless of alpha_mode. Use grDevices::cairo_pdf() (or ragg::agg_png() / grDevices::svg() for raster/vector output) if you need the smooth-gradient rendering.

The visual difference between "step" and "gradient" is only noticeable with very few, thick segments: in "step" mode each segment is visibly a solid colour, while "gradient" interpolates smoothly within each segment too. At typical point counts (>= ~50) both modes look identical – which is what "auto" exploits.

As a special case, a single-segment path (exactly two observations) is always rendered as a gradient even when alpha_mode = "step", because step mode's per-segment alpha would otherwise collapse the fade to a uniform mid-alpha stroke.

RStudio "Unknown group" warning

On the RStudio plot pane (and any device that caches and replays the rendered grob tree), step-mode rendering may emit grid warnings of the form Warning in .useGroup(ref, NULL) : Unknown group, N on repeat draws – typically when the pane is resized and replays a cached tree. The plot itself is correct; the warning is cosmetic noise.

What is happening: step mode uses grid's compositing primitive (groupGrob() with operator "dest.in") to trim the alpha mask onto the polyline. Each draw allocates fresh device-level group IDs. When the plot pane replays a cached grob tree, that tree still references the old IDs while grid's device table has been reset – the lookup fails and grid prints Unknown group, N. The pixels you see are correct because the composited result was already resolved when the cache was built; grid is just reporting that the replay couldn't repeat the lookup, not that anything is missing on screen.

The warning fires inside grid's drawing machinery after our render code has already returned, so we cannot wrap it in suppressWarnings() from within the package. The only clean alternative would gate the fast compositing path off in RStudio, which makes the interactive plot pane roughly five times slower (~1.5 s -> ~7.5 s on a five-group × 573-segment dataset) – we judged that the worse trade-off. If the warnings are intolerable for a particular layer, set alpha_mode = "gradient"; gradient mode uses viewport clipping instead of compositing and does not trigger the warning.

Device-independent alternative

A device-independent alternative to this geom is to densify the path before plotting by interpolating x, y with stats::approx() , and a matching alpha column along the path; then pass the result to a plain ggplot2::geom_path(). Because ggplot2 draws each segment between adjacent points with a uniform aesthetic value (see the ggplot2 book, section 4.4), enough interpolated points make the stepping invisible and produce a smooth apparent fade without any compositing:

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

# Parameterise by cumulative arc length, then densify
arc <- with(df, c(0, cumsum(sqrt(diff(x)^2 + diff(y)^2))))
arc <- arc / max(arc)
grid_pos <- seq(0, 1, length.out = 200)
```

```
dense <- data.frame(
  x     = approx(arc, df$x, xout = grid_pos)$y,
  y     = approx(arc, df$y, xout = grid_pos)$y,
  alpha = 1 - grid_pos   # fade towards end
)

ggplot(dense, aes(x, y, alpha = alpha)) +
  geom_path(linewidth = 2) +
  scale_alpha_identity()
```

The trade-off: this requires manual labour, and it does not generalise to paths with multiple groups or to `fade_direction = c("start", "end")` without additional bookkeeping. `geom_path_fade()` handles all of this internally.

Self-crossing paths

When a faded path crosses itself, the pixels at the crossing are rasterised once per overlapping segment, and the per-segment alpha values compound. Where the two strands carry different alphas (one near the faded end, one near the opaque end), the crossing appears noticeably darker than either strand alone.

This behaviour is inherent to how semi-transparent strokes are alpha-blended at the device level, not specific to `geom_path_fade()` – the same effect appears with `ggplot2::geom_path(alpha = 0.5)`. There is no general workaround at the rendering layer; if a clean intersection matters for your plot, the practical options are:

- Raise `alpha_fade_to` so the strands at both ends are closer in opacity (smaller delta -> less visible darkening).
- Use a fully opaque stroke (no fade) for paths known to self-cross.
- Restructure the data so the crossing is split across separate layers.

Applies equally to `geom_segment_fade()` and `geom_curve_fade()` when two segments / curves overlap at the same pixel.

Aesthetics

`geom_path_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()`
- `group` → inferred
- `linetype` → via `theme()`
- `linewidth` → via `theme()`

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

References

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

geom_line_fade() which sorts by x first, geom_segment_fade() for individual fading segments, ggplot2::geom_path() for the unfaded version.

Examples

```
library(ggplot2)

# Path that doubles back -- fade follows the drawing order
theta <- seq(1.3, -1.3, length.out = 101)
df_ichthys <- data.frame(
  x = theta^2,
  y = 0.5 * theta * (theta^2 - 1)
)

p <- ggplot(df_ichthys, aes(x, y)) +
  geom_pointless(
    location = c("first", "last"),
    aes(colour = after_stat(location)),
    size = 4
  ) +
  coord_fixed() +
  theme_minimal()

p + geom_path_fade(
    linewidth = 1.5,
    fade_direction = "start" # default
  )

p + geom_path_fade(
    linewidth = 1.5,
    fade_direction = c("start", "end")
  )

# With few thick segments the default `"auto"` picks `"gradient"` for
# you, because at n <= 50 the smoother within-segment fade matters more
# than the (negligible) extra compute time.
df_thick <- data.frame(
  x = c(0, 1, 1.5, 1, 0),
  y = c(0, 0.5, 1, 1.5, 1)
)
```

```
p <- ggplot(df_thick, aes(x, y)) +
  coord_equal() +
  theme_minimal()

# Auto -> gradient (n = 5, well below the 50-vertex threshold)
p + geom_path_fade(
  linewidth = 8,
  colour = "#e63946"
  )

# Force `"step"` to see the per-segment stepping for comparison.
p + geom_path_fade(
  linewidth = 8,
  colour = "#e63946",
  alpha_mode = "step"
  )

# Explicit `"gradient"`, in this example, does the same thing
# `"auto"` picked above; for large n (> 200) this gets slow with
# not much gain visually.
p + geom_path_fade(
  linewidth = 8,
  colour = "#e63946",
  alpha_mode = "gradient"
  )

# Using stat_function
ggplot() +
  stat_function(
    alpha = 0.5,
    fun = dnorm,
    n = 100,
    xlim = c(-4, 4),
    geom = "area_fade",
    outline.type = "none" # remove solid outline
  ) +
  # Add fading outline instead
  stat_function(
    fun = dnorm, n = 100,
    xlim = c(-4, 4),
    geom = "path_fade",
    fade_direction = c("start", "end")
  )


nile_df <- data.frame(year = time(datasets::Nile), value = c(datasets::Nile))
ggplot(nile_df, aes(year, value)) +
  geom_line_fade()

# NA values split the path into sub-paths -- just like geom_line().
# The fade is computed over the concatenated arc length of all visible
# pieces, so the alpha just before a gap equals the alpha just after,
# as if the path were "pulled apart" at the NA.
```

```
df <- data.frame(x = c(1, 2, 3, 3, 4, 5), y = c(1, 2, NA, 3, 4, 5))

ggplot(df, aes(x, y)) +
  geom_line_fade(alpha_mode = "gradient", linewidth = 2)


# Fading step function
set.seed(42)
d <- data.frame(
  x   = rep(1:10, 2),
  y   = c(cumsum(rnorm(10)), cumsum(rnorm(10))),
  grp = rep(c("a", "b"), each = 10)
)

ggplot(d, aes(x, y, colour = grp)) +
  geom_step_fade(linewidth = 1, direction = "vh")
```

`geom_pointless` *Emphasise some observations with points*

Description

This is a wrapper around `ggplot2::geom_point()` with one additional argument: `location`. This geom aims to emphasise some observations, and is not particularly useful on its own — hence its name — but it shines 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().

When a single observation matches multiple `location` criteria – for example, the last point is also the maximum – it is emitted once with a composite label joined by `", "` (e.g. `"last, maximum"`). The row order in the output – which drives draw order and legend order – follows the order given in `location`; for `"all"` the canonical order is `"first"`, `"last"`, `"minimum"`, `"maximum"`. 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

```
library(ggplot2)
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 to aesthetics. Pair `colour` with `shape` for redundant
# encoding -- the four roles stay distinguishable in greyscale and under
# colour-vision deficiencies.
p + geom_pointless(
  aes(colour = after_stat(location), shape = 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 `ggplot2::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`. Either a scalar or a numeric vector whose length matches the number of points.

`glow_colour` Colour of the glow. If `NA` (default), it inherits the colour of the point itself. Either a scalar colour or a character vector whose length matches the number of points.

`glow_size` Glow radius in `ggplot2` size units (same scale as the `size` aesthetic in `ggplot2::geom_point()`). If `NA` (default), the glow is rendered at nine times the point's `size`. Either a scalar or a numeric vector whose length matches the number of points.

For the halo to be visible, `glow_size` must exceed the point's `size` – otherwise the point grob (drawn on top) fully covers the glow. If this happens the geom emits a one-shot informational message at draw time pointing you at the fix (enlarge the glow or shrink the point). See examples.

`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()`.

Coordinate systems

`geom_point_glow()` works in all coordinate systems. The glow effect remains point-centric and circular in device space, even in non-linear coordinates like `ggplot2::coord_polar()`.

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)

# Tiny dataset on purpose: each glow point becomes its own
# `grid::radialGradient` pattern, and the example runner accumulates
# patterns from every package example into a single pdf() device.  On
# large pdf runs that pattern cache can trigger an upstream R bug at
# `dev.off()` -- unrelated to your real plots.
df <- head(mtcars, 10)

# Basic usage -- the default glow is 9x the point's `size` aesthetic,
# so it's always visibly larger than the point itself.
ggplot(df, aes(wt, mpg, colour = factor(cyl))) +
  geom_point_glow()


  # Customising the glow: fixed values applied to every point, while
  # point colour is set to transparent
  ggplot(df, aes(wt, mpg)) +
    geom_point_glow(
    colour = "transparent",
    glow_colour = "tomato",
    glow_alpha = .75,
    glow_size = 20
  )

  # Per-point glow: pass a length-N vector for `glow_colour`, `glow_alpha`,
  # or `glow_size`.
  ggplot(df, aes(wt, mpg)) +
    geom_point_glow(glow_colour = rainbow(nrow(df)), glow_size = 15)

  # Use the Geom with another Stat to glow only specific observations:
  ggplot(head(economics), aes(date, uempmed)) +
    geom_line() +
    stat_pointless(
```

```
  geom = "PointGlow",
  glow_colour = "tomato",
  glow_size = 10,
  location = c("first", "last")
)
```

`geom_rect_fade` *Rectangles with a Fading Gradient and Rounded Corners*

Description

`geom_rect_fade()` draws axis-aligned rectangles and fills each one with a linear gradient that fades one edge to transparent. The direction is controlled by `fade_direction`. Corners can be rounded via the `radius` argument, enabling rounded rectangles and smooth-cornered visual elements. The default of `0 pt` produces plain rectangles.

Usage

```
geom_rect_fade(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  alpha_fade_to = 0,
  fade_direction = "vertical",
  radius = grid::unit(0, "pt"),
  lineend = "butt",
  linejoin = "mitre",
  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 — Use to override the default connection between geom_rect_fade() and stat_identity().

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 the fading edge of each rectangle. Defaults to 0 (fully transparent).

fade_direction — Direction of the alpha gradient. One of:

"vertical" (default) Top edge is opaque (ymax), bottom edge fades to alpha_fade_to (ymin).

"horizontal" Left edge is opaque (xmin), right edge fades to alpha_fade_to (xmax).

radius — Corner radius passed to grid::roundrectGrob(). A grid::unit() object (e.g. unit(4, "pt")); a bare number is interpreted as points. Defaults to unit(0, "pt") (sharp corners).

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().

Value

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

Polar coordinates

Under ggplot2::coord_polar() / ggplot2::coord_radial() each rectangle is bent into an annular segment. A radial alpha gradient – transparent at the inner radius, opaque at the outer – is rendered when the fade direction aligns with the radial axis:

- theta = "x" (default) + fade_direction = "vertical": ymin/ymax map to inner/outer radius and fade radially.
- theta = "y" + fade_direction = "horizontal": xmin/xmax map to inner/outer radius and fade radially.

Any other combination (for example theta = "x" with fade_direction = "horizontal") would require an angular / conic gradient, which grid does not yet expose. Such plots fall back to plain ggplot2::geom_rect() rendering and emit a one-time warning. Rounded corners (radius) are ignored in polar coordinates since arcs do not carry corner geometry.

Legend key under coord_flip

The legend key glyph always shows the canonical (data-axis) fade direction – vertical for the default orientation, horizontal under orientation = "y". Under ggplot2::coord_flip() the rendered geom rotates correctly but the legend key does *not*: ggplot2's legend builder is coord-independent by design (draw_key has no access to the coord). For a legend key that matches a horizontal layout, prefer aes(y = ...) with auto-detected orientation = "y" over aes(x = ...) + coord_flip().

Aesthetics

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

- x ***or*** width ***or*** xmin ***or*** xmax
- y ***or*** height ***or*** ymin ***or*** ymax

- 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. (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_rect() for plain rectangles, geom_col_fade() for bar charts with per-bar gradient scaling and orientation support.

Examples

```
library(ggplot2)

# With geom_rect_fade() you can draw arbitrary rectangles
ggplot(head(economics, 25), aes(date, unemploy)) +
  geom_rect_fade(
    data = data.frame(
      xmin = as.Date("1968-07-01"),
      xmax = as.Date("1969-07-01"),
      ymin = -Inf, ymax = 2800
    ),
    inherit.aes = FALSE,
    alpha = 0,
    alpha_fade_to = 0.3,
    aes(xmin = xmin, xmax = xmax, ymin = ymin, ymax = ymax)
  ) +
  stat_fourier(geom = "line_fade", fade_direction = "start", alpha_fade_to = 0.2) +
  geom_point(size = 3, alpha = 0.2) +
  theme_minimal()
```

---

geom_ridgeline_fade *Ridgeline Plots with Fading Gradient*

Description

`geom_ridgeline_fade()` draws ridgeline plots: multiple area shapes stacked at different vertical offsets and adds a vertical alpha gradient that fades from opaque at the peaks to transparent at each ridge's baseline.

The gradient machinery is shared with `geom_area_fade()`; the difference is that each group's baseline is its own `y` value rather than zero, enabling the characteristic overlapping-ridges layout.

`geom_ridgeline_density_fade()` is a convenience wrapper around `geom_ridgeline_fade()` that uses `ggplot2::stat_density()` to compute a kernel density estimate, mapping the result to `height` automatically via `aes(height = after_stat(density))`. Adjust `scale` manually so adjacent ridges reach the desired overlap.

`geom_ridgeline_freqpoly_fade()` and `geom_ridgeline_histogram_fade()` are the ridgeline counterparts of `geom_freqpoly_fade()` and `geom_histogram_fade()`. Both bin the `x` aesthetic per categorical `y` baseline; they differ only in how consecutive bins connect: freqpoly draws linear connections between bin midpoints (polyline); histogram draws stepped flat-top rectangles with vertical jumps at the bin edges.

Usage

```
geom_ridgeline_fade(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = NULL,
  ...,
  alpha_fade_to = 0,
  alpha_scope = "group",
  scale = NULL,
  min_height = NULL,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_ridgeline_density_fade(
  mapping = NULL,
  data = NULL,
  stat = "density",
  ...,
  alpha_fade_to = 0,
  alpha_scope = "group",
  scale = NULL,
  min_height = NULL,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
```

```
)

geom_ridgeline_freqpoly_fade(
  mapping = NULL,
  data = NULL,
  position = NULL,
  ...,
  bins = 30,
  binwidth = NULL,
  center = NULL,
  boundary = NULL,
  closed = c("right", "left"),
  pad = TRUE,
  alpha_fade_to = 0,
  alpha_scope = "group",
  scale = NULL,
  min_height = NULL,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_ridgeline_histogram_fade(
  mapping = NULL,
  data = NULL,
  position = NULL,
  ...,
  bins = 30,
  binwidth = NULL,
  center = NULL,
  boundary = NULL,
  closed = c("right", "left"),
  pad = TRUE,
  alpha_fade_to = 0,
  alpha_scope = "group",
  scale = NULL,
  min_height = NULL,
  na.rm = FALSE,
  orientation = NA,
  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.

`...` Additional arguments passed to `geom_ridgeline_fade()`, including smoothing parameters (`bw`, `adjust`, `kernel`, `n`, `trim`, `bounds`) forwarded to `ggplot2::stat_density()`.

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

`alpha_scope` How to scale alpha across ridges. Vocabulary aligned with `geom_area_fade()`:

- `"group"` (default): every ridge independently uses the full alpha range from `alpha_fade_to` to full opacity. Each ridge is its own reference.
- `"global"`: alpha is scaled relative to the tallest ridge in the entire layer, **including across facet panels**. Shorter ridges fade in proportion.

`scale` Height multiplier applied to `height`. The default `NULL` auto-scales the layer so the tallest ridge overlaps its neighbour by ~50% (to `2 / max(abs(height))`). The auto-resolved value is reported via `cli::cli_inform()` so you have a starting point if you want to override.

`min_height` Minimum `height` value to draw. Points with `height < min_height` are dropped, creating gaps in the ridgeline. Defaults to `0`.

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

`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.

`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()`.

`bins` Number of bins. Overridden by `binwidth`. Defaults to 30. Forwarded to `ggplot2::stat_bin()`.

`binwidth` Width of each bin in data units. When supplied, takes precedence over `bins`. Forwarded to `ggplot2::stat_bin()`.

`center`, `boundary`, `closed`, `pad`
Forwarded to `ggplot2::stat_bin()`. `pad = TRUE` (the default for ridgeline forms) prepends/appends a zero-count bin at each end so the ribbon touches the baseline cleanly.

Value

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

Coordinate systems

`geom_ridgeline_fade()` only supports linear gradients. When used with `ggplot2::coord_polar()` or `ggplot2::coord_radial()`, the geom falls back to standard ridgeline rendering (equivalent to `ggplot2::geom_ribbon()`), which means no gradient fill is added. The geom emits a warning in this case.

Legend key order

Ridges are rendered back-to-front: the ridge with the **highest** y-baseline is drawn first (furthest back) and the ridge with the **lowest** y-baseline is drawn last (on top). When `fill` tracks `y`, the default fill legend lists levels in ascending order – placing the lowest y at the top of the legend – which is the **reverse** of the spatial top-to-bottom reading order (highest y at top of chart, lowest y at bottom).

To align the legend with the chart, reverse the legend keys:

```
+ guides(fill = guide_legend(reverse = TRUE))
```

alpha_scope = "global" under faceting

`alpha_scope = "global"` ties opacity to absolute height across the whole layer, so two ridges / areas / bars of equal height render at equal alpha regardless of which panel they're in. This is meaningful only when panels share a common y scale. Under `facet_wrap(scales = "free_y")` (or `facet_grid(rows = ..., scales = "free")`) each panel rescales y independently, so the visual height of a shape no longer reflects its data height; the alpha encoding then conflicts with what the eye reads from the panel size. For comparable alpha across free-y panels you have two options: stick to the default `scales = "fixed"`, or accept that under free scales `alpha_scope = "group"` is the more honest choice (each shape independently uses its own alpha range).

Legend key under coord_flip

The legend key glyph always shows the canonical (data-axis) fade direction – vertical for the default orientation, horizontal under `orientation = "y"`. Under `ggplot2::coord_flip()` the rendered geom rotates correctly but the legend key does *not*: ggplot2's legend builder is coord-independent by design (`draw_key` has no access to the coord). For a legend key that matches a horizontal layout, prefer `aes(y = ...)` with auto-detected `orientation = "y"` over `aes(x = ...) + coord_flip()`.

Aesthetics

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

- `x`
- `y`
- height
- `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

geom_ridgeline_density_fade() for the convenience density-ridgeline wrapper, geom_area_fade() for area plots with the same gradient effect.

geom_ridgeline_fade() for the lower-level geom. This wrapper mirrors the design of ggridges::geom_density_ridges(from the **ggridges** package by Claus O. Wilke, which is the full-featured original; ggpointless adds the alpha-gradient rendering layer.

Examples

```
library(ggplot2)

totals <- aggregate(
  sales ~ year + month,
  data = subset(txhousing, year <= 2004),
  FUN = sum,
  na.rm = TRUE
)

p <- ggplot(totals, aes(x = month, y = year, group = year, height = sales))
p + geom_ridgeline_fade(outline.type = "none")

# increase overlap using the scale parameter
p + geom_ridgeline_fade(outline.type = "none", scale = 0.0001)

# flip orientation
p + aes(y = month, x = year) +
  geom_ridgeline_fade()

# Map a variable to `fill` to get a 2D gradient
# and use stat_chaikin to smooth curves
p +
  geom_ridgeline_fade(
    aes(fill = after_stat(height)),
    alpha_scope = "global",
    outline.type = "none",
    stat = "chaikin"
  )

# Average monthly temperatures at Nottingham, 1920-1939
dmn <- list(
  month.abb,
  time(datasets::nottem) |> floor() |> unique()
)

df_nottem <- datasets::nottem |>
  matrix(data = _, 12, dimnames = dmn) |>
  as.data.frame() |>
```

```
  stack() |>
  cbind(month = factor(month.abb, levels = month.abb))

# Shared base plot reused by the three ridgeline variants below
p <- ggplot(df_nottem, aes(x = values, y = month)) +
  labs(
    x = NULL,
    y = NULL,
   caption = "Average air temperatures at Nottingham Castle in degrees Fahrenheit (1920-1939)"
  ) +
  scale_fill_gradient(low = "navy", high = "tomato", guide = "none") +
  scale_y_discrete(expand = expansion(add = c(0.2, 1.2)))

# Density ridgelines -- convenience wrapper for the stat = "density"
p +
  geom_ridgeline_density_fade(
    aes(fill = after_stat(x)),
    outline.type = "none"
  )

# freqpoly uses stat = "bin"
p +
  geom_ridgeline_freqpoly_fade(
    aes(fill = after_stat(x)),
    alpha_scope = "global",
    bins = 40
  )
# ridgeline histogram uses stat = "bin" too
p +
  geom_ridgeline_histogram_fade(
    aes(fill = after_stat(x)),
    alpha_scope = "global",
    bins = 40
  )
```

`geom_segment_fade` *Line Segments with a Fading Gradient*

Description

`geom_segment_fade()` draws line segments like `ggplot2::geom_segment()` but applies an alpha gradient along each segment so that one or both ends fade to transparent. The gradient direction follows the segment (from `(x, y)` to `(xend, yend)`), so it works at any angle. Rendering requires a graphics device that supports clipping paths (e.g. ragg, cairo, svg).

Usage

```
geom_segment_fade(
  mapping = NULL,
  data = NULL,
```

```
  stat = "identity",
  position = "identity",
  ...,
  alpha_fade_to = 0,
  fade_direction = "start",
  arrow = NULL,
  arrow.fill = NULL,
  lineend = "butt",
  linejoin = "round",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_curve_fade(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  alpha_fade_to = 0,
  fade_direction = "start",
  curve_count_cap = 200,
  curvature = 0.5,
  angle = 90,
  ncp = 5,
  arrow = NULL,
  arrow.fill = NULL,
  lineend = "butt",
  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 the fading end(s). Defaults to `0` (fully transparent).

`fade_direction` Which end(s) of each segment fade? A character vector containing `"start"` (the `(x, y)` end fades), `"end"` (the `(xend, yend)` end fades), or both `c("start", "end")`. Defaults to `"start"`.

arrow	specification for arrow heads, 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).
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().
curve_count_cap	Soft cap on the number of curves geom_curve_fade() will composite per panel. One groupGrob(op = "dest.in") is built per curve, so render cost scales roughly linearly at ~30 ms / curve. Past the cap the layer falls back to plain ggplot2::geom_curve() (no fade) and emits a warning. Default 200 (~6 s worst case). Pass Inf to disable. For bulk fading, use geom_path_fade() or geom_segment_fade() instead.
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.

Details

Non-linear coordinate systems:

Under non-linear coordinates such as ggplot2::coord_polar() and ggplot2::coord_radial() the user-supplied endpoints (x, y) and (xend, yend) are transformed to device space and connected by a straight chord – exactly as ggplot2::geom_segment() does. The fade is then applied along that chord, so both endpoint positions and the fade direction are consistent with the unfaded geom.

If you want the line to *follow* a curve implied by the coord system (for example a circle at constant y under polar), use geom_hline_fade() / geom_vline_fade() / geom_abline_fade() instead – those geoms subdivide the line in data space so the fade tracks the curve, not a chord.

Value

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

Self-crossing paths

When a faded path crosses itself, the pixels at the crossing are rasterised once per overlapping segment, and the per-segment alpha values compound. Where the two strands carry different alphas (one near the faded end, one near the opaque end), the crossing appears noticeably darker than either strand alone.

This behaviour is inherent to how semi-transparent strokes are alpha-blended at the device level, not specific to `geom_path_fade()` – the same effect appears with `ggplot2::geom_path(alpha = 0.5)`. There is no general workaround at the rendering layer; if a clean intersection matters for your plot, the practical options are:

- Raise `alpha_fade_to` so the strands at both ends are closer in opacity (smaller delta -> less visible darkening).
- Use a fully opaque stroke (no fade) for paths known to self-cross.
- Restructure the data so the crossing is split across separate layers.

Applies equally to `geom_segment_fade()` and `geom_curve_fade()` when two segments / curves overlap at the same pixel.

Aesthetics

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

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

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

References

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

`geom_path_fade()` for connected paths with alpha fading, `ggplot2::geom_segment()` and `ggplot2::geom_curve()` for the unfaded versions.

Examples

```
library(ggplot2)

b <- ggplot(mtcars, aes(wt, mpg)) +
  geom_point()

df <- data.frame(x1 = 2.62, x2 = 3.57, y1 = 21.0, y2 = 15.0)
b +
  geom_curve_fade(
    aes(x = x1, y = y1, xend = x2, yend = y2, colour = "curve"),
    data = df
  ) +
  geom_segment_fade(
    aes(x = x1, y = y1, xend = x2, yend = y2, colour = "segment"),
    data = df
  )

b +
  geom_curve_fade(
    aes(x = x1, y = y1, xend = x2, yend = y2),
    data = df,
    curvature = 1,
    fade_direction = "start",
    arrow = grid::arrow()
  )

df <- data.frame(x1 = 1, x2 = 9, y1 = 1, y2 = 1)
p <- ggplot(df, aes(x)) +
  theme_void()

# Basic example with default fade_direction
p +
  geom_segment_fade(
    aes(x = x1, y = y1, xend = x2, yend = y2),
    fade_direction = "start", # default
    linewidth = 10
  )

# Change fade_direction towards start
p +
  geom_segment_fade(
    aes(x = x1, y = y1, xend = x2, yend = y2),
    fade_direction = "end",
    linewidth = 10
  )

# Fade from centre to both sides
p +
  geom_segment_fade(
    aes(x = x1, y = y1, xend = x2, yend = y2),
    fade_direction = c("start", "end"),
    linewidth = 10
```

```
)
```

`geom_unit_bar` *Unit Bar Charts*

Description

Unit bar charts represent data as discrete cells, where each cell represents one unit of data. They follow the same x/y conventions as `ggplot2::geom_bar()`, `ggplot2::geom_col()`, and `ggplot2::geom_histogram()`:

- `geom_unit_bar()` counts observations (one row = one cell), like `ggplot2::geom_bar()`. Map `x` (or `y` for horizontal bars) to the grouping variable; `fill` to colour by a second variable.
- `geom_unit_col()` uses pre-computed `y` values, like `ggplot2::geom_col()`. Fractional values are supported: `y = 3.7` draws 3 full unit cells (height 1 in data space) plus a partial cell of height 0.7 at the top.

For binning continuous data, see `geom_unit_histogram()`.

All position adjustments supported by `ggplot2::geom_bar()` work here: `"stack"` (default), `"dodge"`, `"fill"`, `position_stack(reverse = TRUE)`, etc. Although `"fill"` rarely makes sense for these geoms; see the examples below for why.

Usage

```
geom_unit_bar(
  mapping = NULL,
  data = NULL,
  stat = "count",
  position = "stack",
  ...,
  just = 0.5,
  radius = grid::unit(0, "pt"),
  orientation = NA,
  width = 1,
  cell_size = 1,
  cell_padding = 0.05,
  cell_count_cap = 10000,
  lineend = "butt",
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_unit_col(
  mapping = NULL,
  data = NULL,
```

```
  stat = "identity",
  position = "stack",
  ...,
  just = 0.5,
  radius = grid::unit(0, "pt"),
  orientation = NA,
  width = 1,
  cell_size = 1,
  cell_padding = 0.05,
  cell_count_cap = 10000,
  lineend = "butt",
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by ggplot2::aes(). For geom_unit_bar(), x (or y for horizontal bars) is required. For geom_unit_col(), both x and y are required. Map fill to colour segments.
data	A data frame.
stat	The statistical transformation to use. Override the default to use a different stat, e.g. stat = "bin" for a tiled histogram.
position	A position adjustment to use on the data. Default "stack" stacks bars on top of each other. Use "dodge" for side-by-side bars, "fill" for proportional stacking, or position_stack(reverse = TRUE) to reverse the stacking order.
...	Other arguments passed to ggplot2::layer().
just	Justification of the bar relative to its x position. 0.5 (default) centres the bar on x, 0 aligns the left edge, 1 aligns the right edge. Same as ggplot2::geom_bar().
radius	Corner radius for each cell as a grid::unit(). Default grid::unit(0, "pt") gives sharp corners. Only used with linear coordinates; non-linear coordinates (e.g. ggplot2::coord_polar()) fall back to sharp corners.
orientation	The orientation of the layer. Default (NA) is guessed from the aesthetic mapping. Set to "x" for vertical bars (value on y) or "y" for horizontal bars (value on x). Same as ggplot2::geom_bar().
width	Bar width in data units. Default 1. With the package defaults (width = 1, cell_size = 1), coord_equal() already renders cells as squares. For non-default width or cell_size, see the position = "dodge" subsection below for the general coord_equal(ratio = ...) formula (it also covers the non-dodge n_groups = 1 case). Same as ggplot2::geom_bar().
cell_size	Number of data units one cell represents. Default 1 (one cell per unit, the original "isotype" / pictogram pattern). Set to a larger value to aggregate units, e.g. cell_size = 1e4 so each cell stands for one thousand. Each cell is then cell_size tall in data space. With the package defaults (width = 1, cell_size

= 1) `coord_equal()` already renders cells as squares; for non-default `cell_size` (or under `position = "dodge"`) the `coord_equal(ratio)` must scale with `cell_size` — see the `position = "dodge"` subsection below for the formula. The value axis still shows the original data values; pair with `scale_*_continuous(labels = label_cells(cell_s` to show cell counts instead.

`cell_padding` Inset applied per side of each cell, in CSS `padding` style. On linear value scales the vertical inset is a fraction of `cell_size` (data space); on non-linear value scales (`log10`, `sqrt`, ...) it becomes a fraction of each cell's *panel* extent so the gap looks visually uniform under compression – see "Log and other non-linear value scales" below. The horizontal inset is always a fraction of the bar's `width` (the bar axis is never the transformed one). Labels are in canonical (vertical-bar) orientation; under `orientation = "y"` or `coord_flip()` the on-screen roles swap, but element 1 always pads the value axis and element 2 always pads the bar axis.

- length 1 (default `0.05`) – same fraction on all four sides.
- length 2, unnamed – `c(vertical, horizontal)`; `vertical` is the inset between vertically-stacked cells, `horizontal` is the inset between each cell and its bar's left/right edge.
- named (length 1 or 2) – positional independence; allowed names are `"vertical"` and `"horizontal"`. A missing axis falls back to the default `0.05`. So `c(horizontal = 0.2, vertical = 0.1)`, `c(vertical = 0.1, horizontal = 0.2)`, and `c(0.1, 0.2)` are all equivalent. Unknown names error rather than silently default – a typo would otherwise be hard to spot in the rendered plot.

Each element must be finite and in `[0, 0.5)`. Set `cell_padding = 0` for cells that touch (the borderless isotype style); increase it for a waffle-like grid of separated cells. The inset is applied uniformly to every cell, including the cells at the bar's outer edges – each cell represents one data unit, so cells must render at identical size regardless of whether they sit at the floor, in the middle, or at the top of a bar. As a consequence the bar's outer edges sit slightly inside the data extent: by `cell_padding * cell_size` vertically and `cell_padding * width` horizontally on linear scales, and proportionally less on non-linear value scales (where the inset is panel-proportional rather than data-proportional).

`cell_count_cap` Soft cap on the total number of cells drawn per panel. A defensive safety net: this geom renders one grob per cell, so very large `y` values can freeze the graphics device. When the cap is exceeded, the layer falls back to solid bars (one rectangle per bar) and emits a warning. Default `1e4`; pass `Inf` to disable. For large `y` you might want to set a larger `cell_size`, see *Examples*.

`lineend` Line end style for the cell border when `colour` is set. One of `"round"`, `"butt"` (default), or `"square"`. Same as `ggplot2::geom_bar()`.

`linejoin` Line join style for the cell border. One of `"round"`, `"mitre"` (default), or `"bevel"`. Same as `ggplot2::geom_bar()`.

`na.rm` If `FALSE` (default), rows with missing `y` are dropped with a warning. Non-positive `y` values are kept: `y = 0` produces an empty segment, and `y < 0` stacks cells downward from the baseline.

`show.legend` logical. Should this layer appear in the legends?

`inherit.aes` If `FALSE`, overrides the default aesthetics.

Value

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

Cell rendering caveats

A few details are easy to overlook. See the *Caveats worth knowing* section of `vignette("ggpointless", package = "ggpointless")` for worked examples and visuals.

- `position = "fill"` collapses every stack to a single cell (the unit semantics disappear). Use `"stack"` or `"dodge"` instead.
- `position = "dodge"` shrinks each sub-bar to `width / n_groups`. To restore square cells, pair with `coord_equal(ratio = width / (n_groups * cell_size))` for vertical bars, or the inverse `ratio = n_groups * cell_size / width` for horizontal. With `preserve = "single"`, `n_groups` is the **max groups per cluster**, not `nlevels(fill)`.
- Non-linear value scales (`log10`, `sqrt`, ...): cells tile in **data space**, so the "1 cell = `cell_size` observations" contract is preserved. Cell heights become non-uniform under compression.
- Tiny panels: the default 5 % gap can collapse below 1 px and cells visually fuse. Enlarge the panel or reduce `cell_size`.
- Polar coordinates: cells become annular segments. Rounded corners are dropped under polar (see `radius`).

Performance

Cost scales with total cell count, not input rows — one grid rect per cell. The defensive `cell_count_cap = 1e4` falls back to plain bars when exceeded; pass `Inf` to disable. For intrinsically large data (populations, currencies, ...), set `cell_size` to aggregate units into single cells. Rounded corners (`radius > 0`) add a `roundrectGrob` per cell and are the most expensive path; leave `radius` at its default for large plots.

Aesthetics

`geom_unit_bar()` 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()`
- `width` → `0.9`

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

- x ***or*** y
- group → inferred
- weight → 1

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

Note

Add `ggplot2::coord_equal()` to ensure cells render as squares. Use `coord_equal(ratio = r)` for non-square cells.

See Also

`ggplot2::geom_bar()` and `ggplot2::geom_col()` for the regular (non-unit) counterparts. `geom_unit_histogram()` for binning continuous data.

Examples

```
library(ggplot2)

# Basic example: count observations with geom_unit_bar()
p <- ggplot(mtcars, aes(reorder(cyl, cyl, length))) +
  labs(y = NULL)
p + geom_unit_bar()

# Let's make cells look square by adding coord_equal()
p <- p + coord_equal()
p + geom_unit_bar()

# Rounded corners are supported too
p + geom_unit_bar(radius = unit(5, "pt"))

# When a variable is mapped to fill
# aesthetic, bars are stacked by default
p + geom_unit_bar(aes(fill = factor(vs)))

# But you might want bars to be dodged
p +
  geom_unit_bar(
    aes(fill = factor(vs)),
    position = position_dodge(preserve = "single")
  ) +
  coord_equal(ratio = 1 / 2)

# Dodging + facets: getting the coord ratio right.
# With `preserve = "single"` every sub-bar is sized to
# `width / max_groups_per_cluster` -- the largest number of fill levels
# appearing at any *one* x-cluster, NOT the total nlevels(fill). In
# `penguins` `fill = species` has three levels, but each island holds
# at most two species (Biscoe: Adelie + Gentoo; Dream: Adelie + Chinstrap;
# Torgersen: Adelie only), so the effective n_groups is 2.
```

```
# The square-cell formula for horizontal bars is
# `ratio = n_groups * cell_size / width`, hence `ratio = 2 * 1 / 1 = 2`
# (not 3, which is what nlevels(fill) would suggest).
if (getRversion() >= "4.5.0") {
  p2 <- ggplot(datasets::penguins, aes(y = island))
  p2 +
    geom_unit_bar(
      aes(fill = species),
      radius = unit(1, "pt"),
      position = position_dodge(preserve = "single"),
      colour = "#333333",
      na.rm = TRUE
    ) +
    labs(x = NULL, y = NULL) +
    facet_wrap(~year, ncol = 1) +
    # max 2 species per island -> ratio = 2, not 3
    coord_equal(ratio = 2) +
    theme(legend.position = "bottom")
}

# Note: position dodge2 adds extra padding by default, but provides
# an option to set this to 0; use the cell_padding argument
# instead for full control of vertical and horizontal padding
p +
  geom_unit_bar(
    aes(fill = factor(vs)),
    position = position_dodge2(preserve = "single", padding = 0),
    cell_padding = c(0.025, 0.1)
  ) +
  coord_equal(ratio = 1 / 2)

# Increase the cell padding (default is 0.05)
p + geom_unit_bar(cell_padding = c(
  "vertical"   = 0.1,
  "horizontal" = 0.05
  )
)

# When you map the categorical to y aesthetic,
# the orientation is auto-detected
ggplot(mtcars, aes(y = reorder(cyl, cyl, length))) +
  geom_unit_bar() +
  coord_equal()

# `scale_*_binned()` belongs on the *mapped continuous variable*, not on the
# count axis. Bin a continuous variable into discrete intervals, then count
# observations per bin -- a unit-cell histogram in two lines:
ggplot(iris, aes(y = Sepal.Length)) +
  # the continuous variable (Sepal.Length) lives on y; binning it ...
  geom_unit_bar() +
  # ... discretises y into intervals so `stat = "count"` can tally each one.
  scale_y_binned()
# Using `scale_y_binned()` on the count axis instead would render an empty
```

```
# plot -- the count axis is already discrete via `stat_count`, so binning
# it again has nothing to bin.


# Plot pre-computed counts with geom_unit_col() (like geom_col() does)
# by default 1 cell represents 1 observation
df <- data.frame(x = c("A", "B", "C"), y = c(10, 12, 8))
ggplot(df, aes(x, y)) + geom_unit_col()

# Too many cells might freeze the graphics device. When cell_count_cap
# is exceeded, the geom falls back to its ggplot2 sibling with a warning.
# For large y, divide at the aes level (e.g. `aes(x, y / 1e3)`) so each
# cell represents a meaningful number of observations.
df <- data.frame(x = c("A", "B", "C"), y = c(10000, 12000, 8000))
ggplot(df, aes(x, y)) + geom_unit_col()

# The aes-level division pattern:
cs <- 1000
ggplot(df, aes(x, y / cs)) +
  geom_unit_col() +
  labs(caption = sprintf("Each cell represents %d observations", cs)) +
  coord_equal()

# Flat cells with rounded corners via coord_equal(ratio = ...)
ggplot(df, aes(x, y / cs)) +
  geom_unit_col(radius = unit(5, "pt")) +
  labs(caption = sprintf("Each cell represents %d observations", cs)) +
  coord_equal(ratio = 1 / 10)
```

`geom_unit_histogram` *Unit Chart Histograms*

Description

A unit-cell version of `ggplot2::geom_histogram()`: bins a continuous variable and draws each bin as a stack of unit cells. With `cell_size = 1` (the default) one cell is one observation, so the bar's height reads as a literal count.

Like `ggplot2::geom_histogram()` this is a thin convenience wrapper around `stat = "bin"`. Pass `bins` or `binwidth` to control the binning; everything else (rendering, position, padding) follows `geom_unit_bar()` – see there for `cell_size`, `cell_padding`, `cell_count_cap`, `radius`, the rendering caveats, and the performance notes.

Usage

```
geom_unit_histogram(
  mapping = NULL,
  data = NULL,
  stat = "bin",
```

```
  position = "stack",
  ...,
  binwidth = NULL,
  bins = NULL,
  orientation = NA,
  radius = grid::unit(0, "pt"),
  cell_size = 1,
  cell_padding = 0.05,
  cell_count_cap = 10000,
  lineend = "butt",
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by ggplot2::aes(). For geom_unit_histogram(), x (or y for horizontal histograms) is required and supplies the continuous variable to be binned. Map fill to colour cell segments.
data	A data frame.
stat	The statistical transformation to use. Override the default to use a different stat, e.g. stat = "bin" for a tiled histogram.
position	A position adjustment to use on the data. Default "stack" stacks bars on top of each other. Use "dodge" for side-by-side bars, "fill" for proportional stacking, or position_stack(reverse = TRUE) to reverse the stacking order.
...	Other arguments passed to ggplot2::layer().
binwidth	Width of each bin in data units. When supplied, takes precedence over bins. Forwarded to ggplot2::stat_bin().
bins	Number of bins. Overridden by binwidth. Defaults to 30. Forwarded to ggplot2::stat_bin().
orientation	The orientation of the layer. Default (NA) is guessed from the aesthetic mapping. Set to "x" for vertical bars (value on y) or "y" for horizontal bars (value on x). Same as ggplot2::geom_bar().
radius	Corner radius for each cell as a grid::unit(). Default grid::unit(0, "pt") gives sharp corners. Only used with linear coordinates; non-linear coordinates (e.g. ggplot2::coord_polar()) fall back to sharp corners.
cell_size	Number of data units one cell represents. Default 1 (one cell per unit, the original "isotype" / pictogram pattern). Set to a larger value to aggregate units, e.g. cell_size = 1e4 so each cell stands for one thousand. Each cell is then cell_size tall in data space. With the package defaults (width = 1, cell_size = 1) coord_equal() already renders cells as squares; for non-default cell_size (or under position = "dodge") the coord_equal(ratio) must scale with cell_size — see the position = "dodge" subsection below for the formula. The value axis still shows the original data values; pair with scale_*_continuous(labels = label_cells(cell_s to show cell counts instead.

`cell_padding` Inset applied per side of each cell, in CSS `padding` style. On linear value scales the vertical inset is a fraction of `cell_size` (data space); on non-linear value scales (`log10`, `sqrt`, ...) it becomes a fraction of each cell's *panel* extent so the gap looks visually uniform under compression – see "Log and other non-linear value scales" below. The horizontal inset is always a fraction of the bar's `width` (the bar axis is never the transformed one). Labels are in canonical (vertical-bar) orientation; under `orientation = "y"` or `coord_flip()` the on-screen roles swap, but element 1 always pads the value axis and element 2 always pads the bar axis.

- length 1 (default `0.05`) – same fraction on all four sides.
- length 2, unnamed – `c(vertical, horizontal)`; `vertical` is the inset between vertically-stacked cells, `horizontal` is the inset between each cell and its bar's left/right edge.
- named (length 1 or 2) – positional independence; allowed names are `"vertical"` and `"horizontal"`. A missing axis falls back to the default `0.05`. So `c(horizontal = 0.2, vertical = 0.1)`, `c(vertical = 0.1, horizontal = 0.2)`, and `c(0.1, 0.2)` are all equivalent. Unknown names error rather than silently default – a typo would otherwise be hard to spot in the rendered plot.

Each element must be finite and in `[0, 0.5)`. Set `cell_padding = 0` for cells that touch (the borderless isotype style); increase it for a waffle-like grid of separated cells. The inset is applied uniformly to every cell, including the cells at the bar's outer edges – each cell represents one data unit, so cells must render at identical size regardless of whether they sit at the floor, in the middle, or at the top of a bar. As a consequence the bar's outer edges sit slightly inside the data extent: by `cell_padding * cell_size` vertically and `cell_padding * width` horizontally on linear scales, and proportionally less on non-linear value scales (where the inset is panel-proportional rather than data-proportional).

`cell_count_cap` Soft cap on the total number of cells drawn per panel. A defensive safety net: this geom renders one grob per cell, so very large `y` values can freeze the graphics device. When the cap is exceeded, the layer falls back to solid bars (one rectangle per bar) and emits a warning. Default `1e4`; pass `Inf` to disable. For large `y` you might want to set a larger `cell_size`, see *Examples*.

`lineend` Line end style for the cell border when `colour` is set. One of `"round"`, `"butt"` (default), or `"square"`. Same as `ggplot2::geom_bar()`.

`linejoin` Line join style for the cell border. One of `"round"`, `"mitre"` (default), or `"bevel"`. Same as `ggplot2::geom_bar()`.

`na.rm` If `FALSE` (default), rows with missing `y` are dropped with a warning. Non-positive `y` values are kept: `y = 0` produces an empty segment, and `y < 0` stacks cells downward from the baseline.

`show.legend` logical. Should this layer appear in the legends?

`inherit.aes` If `FALSE`, overrides the default aesthetics.

Value

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

Aesthetics

geom_unit_bar() 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()
- width → 0.9

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

- x ***or*** y
- group → inferred
- weight → 1

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

See Also

geom_unit_bar() for the discrete counterparts and the shared rendering options. ggplot2::geom_histogram() for the regular (non-unit) histogram.

Examples

```
library(ggplot2)

# Bin a continuous variable; one cell per observation.
# Basic example
p <- ggplot(iris, aes(Sepal.Length, fill = Species))
p + geom_unit_histogram(bins = 40)

# Square cells:
# With `cell_size = 1` (the default), the bin width in data units is
# the `coord_equal()` ratio that makes cells render as squares
bins <- 30
bw <- diff(range(iris$Sepal.Length)) / bins
cs <- 1 # cell_size's default for demonstration here only

# Use bw (bin width) and cs (cell_size) here in coord_equal()
p <- p + coord_equal(ratio = bw / cs)
p + geom_unit_histogram()
```

```
# Rounded corners are supported too
p + geom_unit_histogram(radius = unit(3, "pt"))

# Horizontal orientation, and ratio: 1/2
ggplot(iris, aes(y = Sepal.Length, fill = Species)) +
  geom_unit_histogram() +
  coord_equal(ratio = (1/bw) / 2 ) +
  theme(legend.position = "bottom")

# When bin counts are large, set `cell_size` so each cell aggregates
# multiple observations. Pair with `label_cells()` to relabel the axis.
set.seed(1)
big <- data.frame(l = rnorm(1e5))
bw <- diff(range(big$l)) / 30
cs <- 1000

ggplot(big, aes(l)) +
  geom_unit_histogram(cell_size = cs) +
  scale_y_continuous(
    breaks = seq(0, 10, 2) * cs,
    labels = label_cells(cs, suffix = "k")
  ) +
  coord_equal(ratio = bw / cs) +
  labs(
    x = NULL,
    y = NULL,
    caption = "One cell equals 1.000 observations."
  )
```

`label_cells` *Axis labeller for unit-cell charts*

Description

A thin wrapper around `scales::label_number()` anchored to a `cell_size`: divides each axis break by `cell_size` and formats the result. Use with `scale_*_continuous(labels = ...)` when the corresponding `geom_unit_*()` layer was given a non-default `cell_size` and you want the axis to read in *cell counts* (or in a natural-unit scale like thousands / millions, via `suffix`).

Because `label_cells()` is a wrapper, every option that `scales::label_number()` accepts (`accuracy`, `big.mark`, `decimal.mark`, `scale_cut`, `style_positive`, ...) is available via `...`.

Usage

```
label_cells(cell_size = 1, prefix = "", suffix = "", ...)
```

Arguments

cell_size	The same cell_size value passed to geom_unit_bar() / geom_unit_col() / geom_unit_histogram(). Must be a positive finite scalar. Translated internally to scale = 1 / cell_size.
prefix, suffix	Character strings to wrap each label. Default "" (no decoration). Useful when cell_size matches a natural unit – e.g. cell_size = 1e3 with suffix = "k" produces "1k", "2k", ...; cell_size = 1e6 with suffix = "M" produces "1M", "3M", ...
...	Other arguments forwarded to scales::label_number() – e.g. accuracy = 0.1, big.mark = ",", scale_cut = scales::cut_short_scale() (auto SI prefix).

Value

A function suitable for the labels argument of ggplot2::scale_y_continuous() / ggplot2::scale_x_continuous().

See Also

scales::label_number() for the underlying formatter and the full list of forwardable options; geom_unit_bar() for the geoms that consume cell_size.

Examples

```
library(ggplot2)

# cell_size = 1,000 -> axis reads "1k", "2k", ... (one cell = 1,000)
df_k <- data.frame(x = c("A", "B", "C"), y = c(4000, 11000, 8000))
ggplot(df_k, aes(x, y)) +
  geom_unit_col(cell_size = 1e3) +
  scale_y_continuous(labels = label_cells(1e3, suffix = "k")) +
  labs(
    x = NULL,
    y = NULL,
    caption = "One cell equals 1,000 observations.") +
  coord_equal(ratio = 1 / 1e3)

# cell_size = 1,000,000 -> axis reads "1M", "3M", ... (one cell = 1,000,000)
# Flipped orientation: bars run along x, baselines on y.
df_M <- data.frame(x = c("A", "B", "C"), y = c(2.4e6, 1.1e6, 3.8e6))
ggplot(df_M, aes(y = x, x = y)) +
  geom_unit_col(cell_size = 1e6) +
  scale_x_continuous(labels = label_cells(1e6, suffix = "M")) +
  labs(
    x = NULL,
    y = NULL,
    caption = "One cell equals 1,000,000 observations.") +
  coord_equal(ratio = 1e6)
```

Index