Package: ggtwotone (via r-universe)

July 19, 2026

Type Package

Title Dual-Tone and Contrast-Aware 'ggplot2' Geoms

Version 0.1.0

Description Provides dual-stroke and contrast-aware extensions to 'ggplot2', designed for improved visibility and accessibility in complex visualizations. Includes geoms for dual-stroke segments, regression lines, curved annotations, function plots, paths, and adaptive text. Also includes utility functions for computing contrast-aware color pairs and perceptually distinct highlight palettes using Web Content Accessibility Guidelines (WCAG)-based contrast logic.

License MIT + file LICENSE

URL https://github.com/bwanniarachchige2/ggtwotone

BugReports https://github.com/bwanniarachchige2/ggtwotone/issues

Depends R (>= 4.1.0), ggplot2 (>= 4.0.3)

Imports colorspace (>= 2.1.2), farver (>= 2.1.2), grDevices (>= 4.5.2), grid (>= 4.5.2), methods (>= 4.5.2), rlang (>= 1.1.7), stats (>= 4.5.2), tidyr (>= 1.3.2), utils (>= 4.5.2)

Suggests dplyr, knitr, rmarkdown, scales, spelling, testthat (>= 3.0.0), vdiffr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

VignetteBuilder knitr

Language en-US

NeedsCompilation no

Author Beenu Sareena [aut, cre] (ORCID: <https://orcid.org/0009-0002-7611-5499>), Heike Hofmann [aut] (ORCID: <https://orcid.org/0000-0001-6216-5183>)

Maintainer Beenu Sareena <79365709@nebraska.edu>

Repository https://cran.r-universe.dev

Date/Publication 2026-07-19 13:10:18 UTC

RemoteUrl https://github.com/cran/ggtwotone

RemoteRef HEAD

RemoteSha 66658932543887fb9011f879d75481f97af57dac

Contents

`adjust_contrast_pair` *Adjust Contrast Between Two Stroke Colors*

Description

Given a base color and a background, generate a pair of colors (light and dark) with sufficient perceptual contrast using WCAG or APCA methods.

Usage

```
adjust_contrast_pair(
  color,
  contrast = 4.5,
  method = "auto",
  background = "#FFFFFF",
  quiet = FALSE
)
```

Arguments

`color`	A base color, as a hex string or valid R color name (e.g., "#6699CC", "darkred").
`contrast`	Minimum desired contrast ratio (default is 4.5).
`method`	Contrast method: "WCAG", "APCA", or "auto" to try both.
`background`	Background color, as a hex string or valid R color name (default: "#FFFFFF").
`quiet`	Logical. If TRUE, suppresses warnings.

Value

A list with elements `light`, `dark`, `contrast`, and `method`.

Examples

```
adjust_contrast_pair("#777777", contrast = 4.5,
method = "auto", background = "#000000")

adjust_contrast_pair("#66CCFF", contrast = 4.5,
method = "APCA", background = "#FAFAFA")
```

---

geom_curve_dual *Dual-Stroke Curved Line Annotations*

Description

`geom_curve_dual()` draws a dual-tone curved line using two strokes (light/dark), with slight perpendicular offset to ensure visibility across mixed backgrounds.

Usage

```
geom_curve_dual(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  curvature = NULL,
  angle = NULL,
  ncp = NULL,
  base_color = NULL,
  base_colour = NULL,
  contrast = 4.5,
  method_contrast = "WCAG",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

`data` The data to be displayed in this layer. There are three options:

If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

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

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

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

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

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

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

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

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

An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.

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

`curvature` Bend of the curve (positive = counter-clockwise).

`angle` Angle between the two control points (default: 90).

`ncp` Number of control points (default: 5).

`base_color, base_colour` Base color to derive the dual-tone pair from.

`contrast` Minimum contrast ratio to aim for (default is 4.5).

`method_contrast` Contrast algorithm to use ("WCAG", "APCA", or "auto").

`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 that can be added to a plot with `+`. The returned layer draws dual-stroke curved annotations using the supplied data, aesthetic mappings, statistical transformation, and position adjustment.

Examples

```
library(ggplot2)

# Basic dual-stroke curve
ggplot() +
  geom_curve_dual(
    aes(x = 1, y = 1, xend = 4, yend = 4),
    curvature = 0.4,
    linewidth = 2
  ) +
  theme_void() +
  theme(panel.background = element_rect(fill = "black"))

b <- ggplot(mtcars, aes(wt, mpg)) +
  geom_point(size = 2) +
  theme_dark()
```

```
df <- data.frame(x1 = 2.62, x2 = 3.57, y1 = 21.0, y2 = 15.0)

b + geom_curve_dual(
  data = df,
  mapping = aes(x = x1, y = y1, xend = x2, yend = y2),
  curvature = -0.2,
  linewidth = 2,
  base_color = "green"
)

# Sine wave style dual-stroke curve with points
ggplot() +
  geom_curve_dual(
    aes(x = 2, y = 1, xend = 4, yend = 3),
    curvature = 0.3,
    linewidth = 2,
    base_color = "red"
  ) +
  geom_point(aes(x = 2, y = 1), colour = "red", size = 3) +
  geom_point(aes(x = 4, y = 3), colour = "blue", size = 3) +
  theme_dark(base_size = 14)

# Curve on a grayscale tile background
tile_df <- expand.grid(x = 1:6, y = 1:4)
tile_df$fill <- gray.colors(nrow(tile_df), start = 0, end = 1)

ggplot() +
  geom_tile(
    data = tile_df,
    aes(x = x, y = y, fill = fill),
    width = 1,
    height = 1
  ) +
  geom_curve_dual(
    data = data.frame(x = 1, y = 1, xend = 6, yend = 4),
    aes(x = x, y = y, xend = xend, yend = yend),
    colour1 = "white",
    colour2 = "black",
    curvature = 0.4,
    linewidth = 2
  ) +
  scale_fill_identity() +
  theme_void() +
  coord_fixed()
```

`geom_function_dual` *Dual-Tone Curved Function Lines*

Description

Draws a function (e.g., density or mathematical curve) using perceptually offset dual-stroke curved line segments.

Usage

```
geom_function_dual(
  fun,
  xlim = c(-3, 3),
  n = 701,
  curvature = 0,
  angle = 90,
  ncp = 5,
  colour1 = NULL,
  colour2 = NULL,
  base_color = NULL,
  contrast = 4.5,
  method_contrast = "WCAG",
  linewidth = 1.2,
  args = list(),
  smooth = TRUE,
  color1 = NULL,
  color2 = NULL,
  alpha = 1,
  ...
)
```

Arguments

`fun`	A function to evaluate (e.g., `dnorm`, `dt`).
`xlim`	Range of x-values to evaluate over (numeric vector of length 2).
`n`	Number of segments to compute (default: 201).
`curvature`, `angle`, `ncp`	Passed to underlying `geom_curve_dual` segments.
`colour1`, `colour2`	Fixed top/bottom stroke colours. If only colour1 given, colour2 is derived for contrast. (Aliases color1/color2 also accepted.)
`base_color`	Optional base color to derive a contrast pair (overrides colour1/colour2 if supplied).
`contrast`, `method_contrast`	Passed to adjust_contrast_pair() when deriving colors.
`linewidth`	Stroke width for the top line.
`args`	List of arguments passed to `fun` (for example, list(df = 1) for `dt`).
`smooth`	Use smooth dual-stroke curves (`geom_path`) instead of segmented curves (`geom_curve_dual`). Default is TRUE.

color1, color2	U.S.-spelling aliases for colour1/colour2. Identical in effect; prefer colour1/colour2 in code examples.
alpha	Overall opacity for both strokes (0–1).
...	Additional arguments passed to geom_curve_dual().

Value

A ggplot2 layer with curved segments.

Examples

```
library(ggplot2)

base <- ggplot() + xlim(-2.05,2.05)
base +
  geom_function_dual(
  fun = function(x) 0.5 * exp(-abs(x)),
  xlim = c(-2, 2),
  color1 = "#EEEEEE",
  color2 = "#222222",
  linewidth = 1,
  smooth = TRUE
  ) +
  theme_dark()


ggplot() +
  geom_function_dual(
    fun = dnorm,
    xlim = c(-5, 5),
    base_color = "green",
    linewidth = 1,
    smooth = TRUE
  ) +
  geom_function_dual(
    fun = dt,
    args = list(df = 1),
    xlim = c(-5, 5),
    base_color = "brown",
    linewidth = 1,
    smooth = TRUE
  ) +
  theme_dark()
```

---

geom_lm_dual *Dual-Tone Regression Line with Contrast-Aware Strokes*

Description

Draws a regression line with perceptually distinct dual-stroke coloring for improved visibility.

Usage

```
geom_lm_dual(
  data,
  mapping,
  method = "lm",
  formula = y ~ x,
  base_color = "#777777",
  contrast = 4.5,
  method_contrast = "WCAG",
  ...,
  linewidth = 1,
  show.legend = NA
)
```

Arguments

data	A data frame containing the variables.
mapping	Aesthetic mapping, must include x and y.
method	Regression method to use (default is "lm").
formula	Model formula (default is y ~ x).
base_color	Base color to derive the dual-tone pair from.
contrast	Minimum contrast ratio to aim for (default is 4.5).
method_contrast	Contrast algorithm to use ("WCAG", "APCA", or "auto").
...	Additional parameters passed to geom_segment_dual().
linewidth	Total visual line thickness in mm (both side strokes together).
show.legend	Whether to show legend.

Value

A ggplot2 layer containing the dual-stroke regression line.

Examples

```
library(ggplot2)

# Simple test with linear trend
set.seed(42)
df <- data.frame(x = 1:100, y = 0.5 * (1:100) + rnorm(100))
ggplot(df, aes(x, y)) +
  geom_point() +
  geom_lm_dual(data = df, mapping = aes(x = x, y = y)) +
  theme_minimal()
```

```
# Over grayscale tiles
x <- seq(1, 11, length.out = 100)
y <- 0.5 * x + rnorm(100, 0, 0.3)
df1 <- data.frame(x = x, y = y)

# Tile fill definitions
fill_colors <- data.frame(
  x = 1:11,
  fill = c("#000000", "#1b1b1b", "#444444", "#777777", "#aaaaaa",
           "#dddddd", "#D5D5D5", "#E5E5E5", "#F5F5F5", "#FAFAFA", "#FFFFFF")
)

# Expand tile grid and join with fill colors
tiles <- expand.grid(x = 1:11, y = seq(0, 1, length.out = 100)) |>
  merge(fill_colors, by = "x")

ggplot() +
  geom_tile(
    data = tiles, aes(x = x, y = y, fill = fill),
    width = 1, height = 10
  ) +
  scale_fill_identity() +
  geom_point(
    data = df1, aes(x = x, y = y),
    colour = "purple", size = 2
  ) +
  ## Uncomment to use points with frames:
  # geom_point(
  #   data = df1, aes(x = x, y = y),
  #   shape = 21, colour = "white", fill = "black", size = 3
  # ) +
  geom_lm_dual(
    data = df1, mapping = aes(x = x, y = y),
    linewidth = 2
  ) +
  coord_fixed() +
  theme_minimal()
```

`geom_path_dual` *Draw dual-stroke paths with side-by-side colours*

Description

Draws a path using two side-by-side strokes with separate colours, improving visibility across mixed or low-contrast backgrounds.

Usage

```
geom_path_dual(
```

```
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  colour1 = NULL,
  colour2 = NULL,
  linewidth = NULL,
  lineend = "round",
  linejoin = "round",
  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.

`colour1` Colour for one side of the path.

`colour2` Colour for the other side of the path.

`linewidth` Width of the full dual-stroke path.

`lineend` Line end style.

`linejoin` Line join style.

`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.

Examples

```
library(ggplot2)

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

ggplot(df, aes(x, y)) +
  geom_path_dual(
    colour1 = "#303030",
    colour2 = "#E8E8E8",
    linewidth = 8
  ) +
  theme_minimal()

library(grid)

# simple test data
df <- data.frame(
  x = seq(0, 10, length.out = 300)
)
df$y <- sin(df$x) + 0.2 * cos(3 * df$x)

# striped background for visibility testing
bg <- data.frame(
  xmin = seq(0, 9, by = 1),
  xmax = seq(1, 10, by = 1),
  ymin = -Inf,
  ymax = Inf,
  fill = gray(seq(0.15, 0.85, length.out = 10))
)

ggplot() +
  geom_rect(
    data = bg,
    aes(xmin = xmin, xmax = xmax, ymin = ymin, ymax = ymax, fill = fill),
   inherit.aes = FALSE
  ) +
  scale_fill_identity() +
  geom_path_dual(
    data = df,
    aes(x = x, y = y),
    color = "black",
    color2 = "white",
    linewidth = 2
  ) +
 theme_minimal(base_size = 14)
```

geom_segment_dual *Dual-Stroke Line Segments with Side-by-Side Offset*

Description

Draws two side-by-side line segments with separate colours for improved visibility on varied backgrounds.

Usage

```
geom_segment_dual(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  colour1 = NULL,
  colour2 = NULL,
  linewidth = NULL,
  lineend = "butt",
  aspect_ratio = 1,
  ...,
  arrow = NULL,
  arrow.fill = NULL,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

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

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

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

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

`colour1` Colour for one side of the dual stroke.

`colour2` Colour for the other side of the dual stroke.

`linewidth` Width of the total dual stroke (in mm).

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

`aspect_ratio` Aspect ratio hint (currently unused by the grob logic but reserved for future layout tuning).

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

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

`arrow` 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.
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().

Details

Dual-Stroke Line Segments with Side-by-Side Offset

Draws two side-by-side line segments with separate colours for improved visibility on varied backgrounds.

Value

A ggplot2 layer that can be added to a plot with +. The returned layer draws dual-stroke line segments using the supplied data, aesthetic mappings, statistical transformation, and position adjustment.

Examples

```
# Simple black background test
ggplot(data.frame(x = 1, xend = 2, y = 1, yend = 2),
       aes(x = x, y = y, xend = xend, yend = yend)) +
  geom_segment_dual(colour1 = "white", colour2 = "black", linewidth = 2) +
  theme_void() +
  theme(panel.background = element_rect(fill = "gray20"))

# Dual-stroke diagonal lines crossing contrasting backgrounds
bg <- data.frame(
  xmin = c(0, 5),
  xmax = c(5, 10),
  ymin = 0,
  ymax = 5,
  fill = c("black", "white")
)

line_data <- data.frame(
  x = c(1, 9),
  y = c(1, 1),
  xend = c(9, 1),
  yend = c(4, 4),
  colour1 = c("#D9D9D9", "#D9D9D9"),  # light stroke
  colour2 = c("#333333", "#333333")   # dark stroke
```

```
)

ggplot() +
  geom_rect(data = bg,
  aes(xmin = xmin, xmax = xmax, ymin = ymin, ymax = ymax, fill = fill),
  inherit.aes = FALSE) +
  scale_fill_identity() +
  geom_segment_dual(
    data = line_data,
    aes(x = x, y = y, xend = xend, yend = yend),
    colour1 = line_data$colour1,
    colour2 = line_data$colour2,
    linewidth = 1,
    inherit.aes = FALSE
  ) +
  theme_void() +
  coord_fixed() +
  ggtitle("Two Diagonal Dual-Stroke Lines in Opposite Directions")

# Multiple dual-stroke segments with arrowheads and grouping
df <- data.frame(
  x = c(1, 2, 3),
  xend = c(2, 3, 4),
  y = c(1, 2, 1),
  yend = c(2, 1, 2),
  colour1 = rep("white", 3),
  colour2 = rep("black", 3),
  group = factor(c("A", "B", "C"))
)

ggplot(df) +
  geom_segment_dual(
    aes(x = x, y = y, xend = xend, yend = yend, group = group),
    colour1 = df$colour1,
    colour2 = df$colour2,
    linewidth = 1,
    arrow = arrow(length = unit(0.15, "inches"), type = "closed")
  ) +
  coord_fixed() +
  theme_dark()
```

`geom_text_contrast` *Contrast-Aware Text Geom*

Description

Automatically adjusts text colour for readability on varying background colours using WCAG or APCA contrast.

Usage

```
geom_text_contrast(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  method = "auto",
  contrast = 4.5,
  base_colour = NULL,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  background = NULL
)
```

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.

`method` Contrast method to use: `"WCAG"`, `"APCA"`, or `"auto"`.

`contrast` Threshold to ensure between text and background (defaults to 4.5).

`base_colour` Base text colour used to generate light/dark variants.

`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()`.

`background` A character vector of background fill colours (hex codes), used for contrast computation.

Value

A `ggplot2` layer that can be added to a plot with `+`. The returned layer draws text labels with foreground colors automatically selected to improve contrast against the specified background colors.

Examples

```
library(ggplot2)

# Grayscale A–E tiles: testing contrast
df <- data.frame(
  x = 1:5,
  y = 1,
  label = LETTERS[1:5],
  fill = c("#000000", "#222222", "#666666", "#DDDDDD", "#FFFFFF")
)
ggplot(df, aes(x, y)) +
  geom_tile(aes(fill = fill), width = 1, height = 1) +
  geom_text_contrast(
    aes(label = label),
    background = df$fill,
    size = 7
  ) +
  scale_fill_identity() +
  coord_fixed() +
  theme_void() +
  labs(title = "Contrast-Aware Text on Varying Backgrounds")

library(dplyr)
library(scales)

set.seed(1)
classes <- c("Cat", "Dog", "Rabit")
cm <- expand.grid(True = classes, Predicted = classes)
cm$Count <- sample(5:200, size = nrow(cm), replace = TRUE)

cm <- cm %>%
  group_by(True) %>%
  mutate(Accuracy = Count / sum(Count))

pal <- c("#313695", "#74add1", "#fdae61", "#a50026")
col_fun <- col_numeric(palette = pal, domain = c(0, 1))

cm$fill_hex <- col_fun(cm$Accuracy)
cm$label <- sprintf("%.1f%%", 100 * cm$Accuracy)

ggplot(cm, aes(Predicted, True)) +
  geom_tile(aes(fill = Accuracy), color = "white", linewidth = 0.8) +
  geom_text_contrast(
    aes(label = label),
    background = cm$fill_hex,
    base_colour = "#004488",
    method = "auto",
    contrast = 4.5,
    size = 5, fontface = "bold"
  ) +
  scale_fill_gradientn(
    colours = pal,
```

```
    limits = c(0, 1),
    name = "Accuracy"
  ) +
  coord_fixed() +
  labs(
    title = "Confusion Matrix with Auto-Contrast Labels",
    x = "Predicted Class", y = "True Class"
  ) +
  theme_minimal(base_size = 13) +
  theme(
    panel.grid = element_blank(),
    axis.text.x = element_text(angle = 45, hjust = 1)
  )


# Simulated region × risk category
df_risk <- expand.grid(
  region = LETTERS[1:6],
  zone = paste0("Z", 1:6)
)

df_risk$risk_level <- sample(
  c("Low", "Moderate", "High", "Critical", "Severe", "Extreme"),
  size = nrow(df_risk), replace = TRUE
)
df_risk$label <- paste(df_risk$region, df_risk$zone)

risk_colours <- c(
  "Low" = "gray80",
  "Moderate" = "skyblue",
  "High" = "orange",
  "Critical" = "firebrick",
  "Severe" = "darkred",
  "Extreme" = "navy"
)

df_risk$fill_colour <- risk_colours[df_risk$risk_level]

ggplot(df_risk, aes(x = region, y = zone, fill = risk_level)) +
  geom_tile(colour = "white") +
  geom_text_contrast(
    aes(label = label),
    background = df_risk$fill_colour,
    size = 3,
    fontface = "bold"
  ) +
  scale_fill_manual(values = risk_colours) +
  labs(
    title = "Simulated Risk Map (Auto Contrast Labels)",
    fill = "Risk Level"
  ) +
  theme_minimal()
```

highlight_colors *Generate a high-contrast, well-separated highlight palette (global search)*

Description

Changes from your previous version:

- Global hue sampling by default (`hue_targets = "auto"`)
- Optional hue repulsion near background/base (`repel_from`, `repel_band`)
- Smart auto-bias toward opponent hues when bg & base are far apart (`auto_bias`)
- NEW: `contrast_method = "auto"` picks APCA for dark-on-light and WCAG for light-on-dark

Usage

```
highlight_colors(
  n,
  background = "#F0F0F0",
  base_color = "#4a1919",
  contrast_bg = 4.5,
  contrast_base = 3,
  min_deltaE = 35,
  hue_targets = "auto",
  anchor_band = 180,
  hcl_L_range = c(25, 75),
  hcl_C_range = c(40, 90),
  min_hue_sep = 35,
  n_candidates = 1e+05,
  relax = TRUE,
  quiet = TRUE,
  repel_from = c("background", "base"),
  repel_band = 50,
  auto_bias = TRUE,
  bias_width = 120,
  bias_frac = 0.8,
  contrast_method = c("option", "auto")
)
```

Arguments

n	Integer, number of colors.
background	Hex/R color for the plot background.
base_color	Hex/R color used for non-highlight elements.
contrast_bg	Required contrast vs background (WCAG ratio or APCA Lc depending on backend). Default 4.5.

contrast_base	Required contrast vs base_color (set 0 to skip). Default 3.0.
min_deltaE	Minimum CIEDE2000 separation between selected colors. Default 35.
hue_targets	"auto", "anchored", "spread", or numeric vector of hue angles (0-360).
anchor_band	Degrees around base hue to sample when hue_targets = "anchored". Default 180.
hcl_L_range	Allowed HCL lightness range. Default c(25, 75).
hcl_C_range	Allowed HCL chroma range. Default c(40, 90).
min_hue_sep	Minimum separation in degrees between chosen hues. Default 35.
n_candidates	Number of random candidates before selection. Default 100000.
relax	Progressively relax min_hue_sep then min_deltaE if infeasible. Default TRUE.
quiet	Suppress relaxation messages. Default TRUE.
repel_from	Character vector among c("background", "base") to avoid those hue neighborhoods.
repel_band	Degrees excluded around the repelled hues (+/- band). Default 50.
auto_bias	Logical; if TRUE and bg/base are far apart, bias sampling toward opponent hues.
bias_width	Width (degrees) of opponent sampling window when auto_bias applies. Default 120.
bias_frac	Fraction of samples drawn from the biased window when auto_bias applies. Default 0.8.
contrast_method	"option" (use global option, default) or "auto" (pick WCAG/APCA per polarity).

Details

Creates n colors by: (1) sampling HCL colors (global or biased), (2) filtering by WCAG/APCA contrast vs background (and optionally vs base_color), (3) selecting a maximally separated subset by ΔE_{2000} and minimum hue spacing.

Set contrast_base = 0 to ignore base contrast (keeps only background contrast). Use hue_targets = "anchored" to sample around base_color (legacy-like).

NOTE: Switch contrast backend via: options(ggtwotone.contrast_method = "WCAG") options(ggtwotone.contrast_m = "APCA") or pass contrast_method = "auto" to pick per polarity.

Value

Character vector of hex colors (length <= n) with an info attribute.

Examples

```
highlight_colors(
  n = 4,
  background = "#222222",
  base_color = "#eeeeee"
)
```

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)

  bg_hex <- "#F7F7F7"
  base_hex <- "#222222"

  set.seed(7)
  pal <- highlight_colors(
    n = 3,
    background = bg_hex,
    base_color = base_hex,
    contrast_method = "auto",
    contrast_bg = 60,
    contrast_base = 45,
    quiet = TRUE
  )

  classes <- c("compact", "suv", "midsize", "pickup", "minivan", "2seater")
  counts <- c(25, 38, 42, 31, 10, 5)
  highlight_classes <- c("compact", "suv", "midsize")

  col_map <- setNames(rep(base_hex, length(classes)), classes)
  col_map[highlight_classes] <- pal

  df <- data.frame(
    class = classes,
    count = counts,
    fill = unname(col_map[classes])
  )

  ggplot(df, aes(x = reorder(class, count), y = count, fill = fill)) +
    geom_col(width = 0.8) +
    scale_fill_identity() +
    coord_flip() +
    labs(
      title = "Dark base on light background (auto -> APCA)",
      x = NULL,
      y = "Count"
    ) +
    theme_minimal(base_size = 12) +
    theme(
      panel.grid.major.y = element_blank(),
      plot.background = element_rect(fill = bg_hex, color = NA),
      panel.background = element_rect(fill = bg_hex, color = NA),
      plot.title = element_text(color = base_hex, hjust = 0.5),
      axis.text = element_text(color = base_hex)
    )
}

if (requireNamespace("ggplot2", quietly = TRUE)) {
  library(ggplot2)

  bg_hex <- "#222222"
```

```
  base_hex <- "#EEEEEE"

  set.seed(7)
  pal <- highlight_colors(
    n = 3,
    background = bg_hex,
    base_color = base_hex,
    contrast_method = "auto",
    contrast_bg = 4.5,
    contrast_base = 3.0,
    quiet = TRUE
  )

  classes <- c("compact", "suv", "midsize", "pickup", "minivan", "2seater")
  counts <- c(25, 38, 42, 31, 10, 5)
  highlight_classes <- c("compact", "suv", "midsize")

  col_map <- setNames(rep(base_hex, length(classes)), classes)
  col_map[highlight_classes] <- pal

  df <- data.frame(
    class = classes,
    count = counts,
    fill = unname(col_map[classes])
  )

  ggplot(df, aes(x = reorder(class, count), y = count, fill = fill)) +
    geom_col(width = 0.8) +
    scale_fill_identity() +
    coord_flip() +
    labs(
      title = "Light base on dark background (auto -> WCAG)",
      x = NULL,
      y = "Count"
    ) +
    theme_minimal(base_size = 12) +
    theme(
      panel.grid.major.y = element_blank(),
      plot.background = element_rect(fill = bg_hex, color = NA),
      panel.background = element_rect(fill = bg_hex, color = NA),
      plot.title = element_text(color = base_hex, hjust = 0.5),
      axis.text = element_text(color = base_hex)
    )
}
```

`highlight_info` *Inspect effective palette constraints*

Description

Inspect effective palette constraints

Usage

```
highlight_info(pal)
```

Arguments

pal	A palette returned by highlight_colors()

Value

A named list of effective thresholds and counts.

Index