Package 'ggblend'

July 22, 2025

Title Blending and Compositing Algebra for 'ggplot2'

Version 0.1.0

Description Algebra of operations for blending, copying, adjusting, and compositing layers in 'ggplot2'. Supports copying and adjusting the aesthetics or parameters of an existing layer, partitioning a layer into multiple pieces for re-composition, applying affine transformations to layers, and combining layers (or partitions of layers) using blend modes (including commutative blend modes, like multiply and darken). Blend mode support is particularly useful for creating plots with overlapping groups where the layer drawing order does not change the output; see Kindlmann and Scheidegger (2014) <doi:10.1109/TVCG.2014.2346325>.

License MIT + file LICENSE

Language en-US

Depends R (>= 4.2)

Imports methods, grid, ggplot2 (>= 3.4.0), rlang

Suggests covr, testthat (>= 3.0.0), fontquiver, showtext, sysfonts, ggnewscale

Config/testthat/edition 3

BugReports https://github.com/mjskay/ggblend/issues/new

URL https://mjskay.github.io/ggblend/, https://github.com/mjskay/ggblend/

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Author Matthew Kay [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-9446-0419>)

Maintainer Matthew Kay <mjskay@northwestern.edu>

Repository CRAN

Date/Publication 2023-05-22 08:30:05 UTC

Contents

`ggblend-package` *Blending and compositing for ggplot2*

Description

ggblend is an R package that adds support for R 4.2 blend modes (e.g. `"multiply"`, `"overlay"`, etc) to **ggplot2**.

Details

The primary support for blending is provided by the `blend()` function, which can be used to augment `ggplot()` layers/geoms or lists of layers/geoms in a `ggplot()` specification.

For example, one can replace something like this:

```
df |>
  ggplot(aes(x, y)) +
  geom_X(...) +
  geom_Y(...) +
  geom_Z(...)
```

With something like this:

```
df |>
  ggplot(aes(x, y)) +
  geom_X(...) +
  geom_Y(...) |> blend("multiply") +
  geom_Z(...)
```

In order to apply a "multiply" blend to the layer with `geom_Y(...)`.

Package options

The following global options can be set using `options()` to modify the behavior of **ggblend**:

- `"ggblend.check_blend"`: If `TRUE` (default), `blend()` will warn if you attempt to use a blend mode not supported by the current graphics device, as reported by `dev.capabilities()$compositing`. Since this check can be unreliable on some devices (they will report not support a blend mode that they do support), you can disable this warning by setting this option to `FALSE`.
- `"ggblend.check_affine_transform"`: If `TRUE` (default), `affine_transform()` will warn if you attempt to use a blend mode not supported by the current graphics device, as reported by `dev.capabilities()$transformation`. Since this check can be unreliable on some devices (they will report not support a blend mode that they do support), you can disable this warning by setting this option to `FALSE`.

`adjust` *Adjust layer params and aesthetics (Layer operation)*

Description

A layer operation for adjusting the params and aesthetic mappings of a layer-like object.

Usage

```
adjust(object, mapping = aes(), ...)
```

Arguments

`object`	One of: • A layer-like object: applies this operation to the layer. • A missing argument: creates an operation • Anything else: creates an operation, passing `object` along to the `mapping` argument
`mapping`	An aesthetic created using `aes()`. Mappings provided here will overwrite mappings in `ggplot2::layer()`s when this operation is applied to them.
`...`	`ggplot2::layer()` parameters, such as would be passed to a `geom_...()` or `stat_...()` call. Params provided here will overwrite params in layers when this operation is applied to them.

Value

A layer-like object (if `object` is layer-like) or an operation (if not).

See Also

operation for a description of layer operations.

Other layer operations: `affine_transform`, `blend`, `copy`, `nop`, `partition()`

Examples

```
library(ggplot2)

# Here we use adjust() with nop() ( + 1) to create a copy of
# the stat_smooth layer, putting a white outline around it.
set.seed(1234)
k = 1000
data.frame(
  x = seq(1, 10, length.out = k),
  y = rnorm(k, seq(1, 2, length.out = k) + c(0, 0.5)),
  g = c("a", "b")
) |>
  ggplot(aes(x, y, color = g)) +
  geom_point() +
  stat_smooth(method = lm, formula = y ~ x, linewidth = 1.5, se = FALSE) *
    (adjust(aes(group = g), color = "white", linewidth = 4) + 1) +
  scale_color_brewer(palette = "Dark2")

# (note this could also be done with copy_under())
```

`affine_transform` *Translate, scale, and rotate ggplot2 layers (Layer operation)*

Description

Transform objects within a single layer (geom) or across multiple layers (geoms) using affine transformations, like translation, scale, and rotation. Uses the built-in compositing support in graphical devices added in R 4.2.

Usage

```
affine_transform(object, x = 0, y = 0, width = 1, height = 1, angle = 0)
```

Arguments

`object`	One of: • A layer-like object: applies this operation to the layer. • A missing argument: creates an operation • A `numeric()` or `unit()` giving the x-axis translation, which takes the place of the `x` argument.
`x`	A `numeric()` or `unit()` giving the x translation to apply.
`y`	A `numeric()` or `unit()` giving the y translation to apply.
`width`	A `numeric()` or `unit()` giving the width.
`height`	A `numeric()` or `unit()` giving the height.
`angle`	A `numeric()` giving the angle to rotate, in degrees.

Details

Applies an affine transformation (translation, scaling, rotation) to a layer.

Note: due to limitations in the implementation of scaling and rotation, currently these operations can only be performed relative to the center of the plot. In future versions, the translation and rotation origin may be configurable.

Value

A layer-like object (if `object` is layer-like) or an operation (if not).

Supported devices

Transformation is not currently supported by all graphics devices. As of this writing, at least `png(type = "cairo")`, `svg()`, and `cairo_pdf()` are known to support blending.

`affine_transform()` attempts to auto-detect support for affine transformation using `dev.capabilities()`. You may receive a warning when using `affine_transform()` if it appears transformation is not supported by the current graphics device. This warning **either** means (1) your graphics device does not support transformation (in which case you should switch to one that does) or (2) your graphics device supports transformation but incorrectly reports that it does not. Unfortunately, not all graphics devices that support transformation appear to correctly *report* that they support transformation, so even if auto-detection fails, `blend()` will still attempt to apply the transformation, just in case.

If the warning is issued and the output is still correctly transformed, this is likely a bug in the graphics device. You can report the bug to the authors of the graphics device if you wish; in the mean time, you can use `options(ggblend.check_affine_transform = FALSE)` to disable the check.

References

Murrell, Paul (2021): Groups, Compositing Operators, and Affine Transformations in R Graphics. The University of Auckland. Report. doi:10.17608/k6.auckland.17009120.v1.

See Also

operation for a description of layer operations.

Other layer operations: `adjust`, `blend`, `copy`, `nop`, `partition()`

Examples

```
library(ggplot2)

# a simple dataset:
set.seed(1234)
data.frame(x = rnorm(100), y = rnorm(100)) |>
  ggplot(aes(x, y)) +
  geom_point() +
  xlim(-5, 5)

# we could scale and translate copies of the point cloud
# (though I'm not sure why...)
```

```
data.frame(x = rnorm(100), y = rnorm(100)) |>
  ggplot(aes(x, y)) +
  geom_point() * (
    affine_transform(x = -unit(100, "pt"), width = 0.5) |> adjust(color = "red") +
    affine_transform(width = 0.5) +
    affine_transform(x = unit(100, "pt"), width = 0.5) |> adjust(color = "blue")
  ) +
  xlim(-5, 5)
```

`blend` *Blend ggplot2 layers (Layer operation)*

Description

Blend objects within a single layer (geom) or across multiple layers (geoms) using graphical blending modes, such as `"multiply"`, `"overlay"`, etc. Uses the built-in compositing support in graphical devices added in R 4.2.

Usage

```
blend(object, blend = "over", alpha = 1)
```

Arguments

`object`	One of: • A layer-like object: applies this operation to the layer. • A missing argument: creates an operation • A string (character vector of length 1) giving the name of a blend, which takes the place of the `blend` argument.
`blend`	The blend mode to use. The default mode, `"over"`, corresponds to the "usual" blend mode of drawing objects on top of each other. The list of supported blend modes depends on your graphical device (see Murrell 2021), and are listed in `dev.capabilities()$compositing`. Blend modes can include: `"clear"`, `"source"`, `"over"`, `"in"`, `"out"`, `"atop"`, `"dest"`, `"dest.over"`, `"dest.in"`, `"dest.out"`, `"dest.atop"`, `"xor"`, `"add"`, `"saturate"`, `"multiply"`, `"screen"`, `"overlay"`, `"darken"`, `"lighten"`, `"color.dodge"`, `"color.burn"`, `"hard.light"`, `"soft.light"`, `"difference"`, and `"exclusion"` Blend modes like `"multiply"`, `"darken"`, and `"lighten"` are particularly useful as they are *commutative*: the result is the same whichever order they are applied in. A warning is issued if the current graphics device does not appear to support the requested blend mode. In some cases this warning may be spurious, so it can be disabled by setting `options(ggblend.check_blend = FALSE)`.
`alpha`	A numeric between `0` and `1` (inclusive). The opacity of a transparency mask applied to objects prior to blending.

Details

If `object` is a single layer / geometry and the `partition` aesthetic *is not* set, every graphical object (`grob()`) output by the geometry will be blended together using the `blend` blend mode. If `alpha != 1`, a transparency mask with the provided alpha level will be applied to each grob before blending.

If `object` is a single layer / geometry and the `partition` aesthetic *is* set, the geometry will be rendered for each subset of the data defined by the `partition` aesthetic, a transparency mask with the provided `alpha` level will be applied to each resulting group as a whole (if `alpha != 1`), then these groups will be blended together using the `blend` blend mode.

If `object` is a list of layers / geometries, those layers will be rendered separately, a transparency mask with the provided `alpha` level will be applied to each layer as a whole (if `alpha != 1`), then these layers will be blended together using the `blend` blend mode.

If a `blend()` is multiplied by a list of layers using `*`, it acts on each layer individually (as if each layer were passed to `blend()`).

Value

A layer-like object (if `object` is layer-like) or an operation (if not).

Supported devices

Blending is not currently supported by all graphics devices. As of this writing, at least `png(type = "cairo")`, `svg()`, and `cairo_pdf()` are known to support blending.

`blend()` attempts to auto-detect support for blending using `dev.capabilities()`. You may receive a warning when using `blend()` if it appears blending is not supported by the current graphics device. This warning **either** means (1) your graphics device does not support blending (in which case you should switch to one that does) or (2) your graphics device supports blending but incorrectly reports that it does not. Unfortunately, not all graphics devices that support blending appear to correctly *report* that they support blending, so even if auto-detection fails, `blend()` will still attempt to apply the blend, just in case.

If the warning is issued and the output is still correctly blended, this is likely a bug in the graphics device. You can report the bug to the authors of the graphics device if you wish; in the mean time, you can use `options(ggblend.check_blend = FALSE)` to disable the check.

References

Murrell, Paul (2021): Groups, Compositing Operators, and Affine Transformations in R Graphics. The University of Auckland. Report. doi:10.17608/k6.auckland.17009120.v1.

See Also

operation for a description of layer operations.

Other layer operations: `adjust`, `affine_transform`, `copy`, `nop`, `partition()`

Examples

```
library(ggplot2)

# create two versions of a dataset, where draw order can affect output
set.seed(1234)
df_a = data.frame(x = rnorm(500, 0), y = rnorm(500, 1), set = "a")
df_b = data.frame(x = rnorm(500, 1), y = rnorm(500, 2), set = "b")
df_ab = rbind(df_a, df_b) |>
  transform(order = "draw a then b")
df_ba = rbind(df_b, df_a) |>
  transform(order = "draw b then a")
df = rbind(df_ab, df_ba)

# Using the "darken" blend mode, draw order does not matter:
df |>
  ggplot(aes(x, y, color = set)) +
  geom_point(size = 3) |> blend("darken") +
  scale_color_brewer(palette = "Set2") +
  facet_grid(~ order)

# Using the "multiply" blend mode, we can see density within groups:
df |>
  ggplot(aes(x, y, color = set)) +
  geom_point(size = 3) |> blend("multiply") +
  scale_color_brewer(palette = "Set2") +
  facet_grid(~ order)

# blend() on a single geom by default blends all grobs in that geom together
# using the requested blend mode. If we wish to blend within specific data
# subsets using normal blending ("over") but between subsets using the
# requested blend mode, we can set the partition aesthetic. This will
# make "multiply" behave more like "darken":
df |>
  ggplot(aes(x, y, color = set, partition = set)) +
  geom_point(size = 3) |> blend("multiply") +
  scale_color_brewer(palette = "Set2") +
  facet_grid(~ order)

# We can also blend lists of geoms together; these geoms are rendered using
# normal ("over") blending (unless a blend() call is applied to a specific
# sub-layer, as in the first layer below) and then blended together using
# the requested blend mode.
df |>
  ggplot(aes(x, y, color = set)) +
  list(
    geom_point(size = 3) |> blend("darken"),
    geom_vline(xintercept = 0, color = "gray75", linewidth = 1.5),
    geom_hline(yintercept = 0, color = "gray75", linewidth = 1.5)
  ) |> blend("hard.light") +
  scale_color_brewer(palette = "Set2") +
  facet_grid(~ order)
```

copy *Copy layers then adjust params and aesthetics (Layer operation)*

Description

A layer operation for copying and then adjusting the params and aesthetic mappings of a layer-like object.

Usage

```
copy_over(object, mapping = aes(), ...)

copy_under(object, mapping = aes(), ...)
```

Arguments

`object`	One of: • A layer-like object: applies this operation to the layer. • A missing argument: creates an operation • Anything else: creates an operation, passing `object` along to the `mapping` argument
`mapping`	An aesthetic created using `aes()`. Mappings provided here will overwrite mappings in `ggplot2::layer()`s when this operation is applied to them.
`...`	`ggplot2::layer()` parameters, such as would be passed to a `geom_...()` or `stat_...()` call. Params provided here will overwrite params in layers when this operation is applied to them.

Details

These are shortcuts for duplicating a layer and then applying `adjust()`. Specifically:

- `copy_over(...)` is equivalent to `1 + adjust(...)`
- `copy_under(...)` is equivalent to `adjust(...) + 1`

Value

A layer-like object (if `object` is layer-like) or an operation (if not).

See Also

operation for a description of layer operations.

Other layer operations: `adjust`, `affine_transform`, `blend`, `nop`, `partition()`

Examples

```
library(ggplot2)

# here we use copy_under() to create a copy of
# the stat_smooth layer, putting a white outline around it.
set.seed(1234)
k = 1000
data.frame(
  x = seq(1, 10, length.out = k),
  y = rnorm(k, seq(1, 2, length.out = k) + c(0, 0.5)),
  g = c("a", "b")
) |>
  ggplot(aes(x, y, color = g)) +
  geom_point() +
  stat_smooth(method = lm, formula = y ~ x, linewidth = 1.5, se = FALSE) *
    copy_under(aes(group = g), color = "white", linewidth = 4) +
  scale_color_brewer(palette = "Dark2")
```

---

layer-like *ggplot2 layer-like objects*

Description

For technical reasons related to how **ggplot2** implements layers, there is no single class from which all valid **ggplot2** layers and lists of layers inherit. Thus, **ggblend** operations supports a variety of "layer-like" objects, documented here (see *Details*).

Usage

```
is_layer_like(x)

as_layer_like(x)

## Default S3 method:
as_layer_like(x)

## S3 method for class 'LayerInstance'
as_layer_like(x)

## S3 method for class 'list'
as_layer_like(x)

## S3 method for class 'layer_list'
as_layer_like(x)
```

Arguments

x	A layer-like object. See *Details*.

Details

ggblend operations can be applied to several `ggplot2::layer()`-like objects, including:

- objects of class `"LayerInstance"`; e.g. `stat`s and `geom`s.
- `list()`s of layer-like objects.
- `layer_list()`s, which are a more type-safe version of `list()`s of layer-like objects.

Anywhere in **ggblend** where a function parameter is documented as being layer-like, it can be any of the above object types.

Value

For `is_layer_like()`, a `logical`: `TRUE` if `x` is layer-like, `FALSE` otherwise.

For `as_layer_like()`, a `"LayerInstance"` or a `layer_list()`.

Functions

- `is_layer_like()`: checks if an object is layer-like according to **ggblend**.
- `as_layer_like()`: validates that an object is layer-like and converts it to a `"LayerInstance"` or `layer_list()`.

Examples

```
library(ggplot2)

is_layer_like(geom_line())
is_layer_like(list(geom_line()))
is_layer_like(list(geom_line(), scale_x_continuous()))
is_layer_like(list(geom_line(), "abc"))
```

---

layer_list *Lists of layer-like objects*

Description

A list of layer-like objects, which can be used in layer operations (through function application or multiplication) or added to a `ggplot2()` object.

Usage

```
layer_list(...)

as_layer_list(x)

## S3 method for class 'layer_list'
as_layer_list(x)
```

```
## S3 method for class 'list'
as_layer_list(x)

## S3 method for class 'LayerInstance'
as_layer_list(x)

## S4 method for signature 'layer_list,layer_list'
e1 + e2

## S4 method for signature 'layer_list'
show(object)
```

Arguments

x, ...	layer-like objects
object, e1, e2	layer_list()s

Details

For the most part, users of **ggblend** need not worry about this class. It is used internally to simplify multiple dispatch on binary operators, as the alternative (list()s of ggplot2::layer()s) is more cumbersome. **ggblend** converts input lists to this format as needed.

Value

An object of class "layer_list".

Examples

```
library(ggplot2)

# layer_list()s act just like list()s of layer()s in that they can
# be added to ggplot() objects
data.frame(x = 1:10) |>
  ggplot(aes(x, x)) +
  layer_list(
    geom_line(),
    geom_point()
  )
```

---

nop *Identity ("no-op") transformation (Layer operation)*

Description

A layer operation which returns the input layer-like object unchanged.

Usage

```
nop(object)
```

Arguments

object	One of: • A layer-like object: applies this operation to the layer. • A missing argument: creates an operation

Details

When `numeric()`s are used with operations, they are converted into sums of `nop()`s.

Value

A layer-like object (if `object` is layer-like) or an operation (if not).

See Also

operation for a description of layer operations.

Other layer operations: `adjust`, `affine_transform`, `blend`, `copy`, `partition()`

Examples

```
library(ggplot2)

# adding a nop to another operation is equivalent to adding a numeric
adjust() + nop()

# and vice versa
adjust() + 2

# here we use adjust() with nop() ( + 1) to create a copy of
# the stat_smooth layer, putting a white outline around it.
set.seed(1234)
k = 1000
data.frame(
  x = seq(1, 10, length.out = k),
  y = rnorm(k, seq(1, 2, length.out = k) + c(0, 0.5)),
  g = c("a", "b")
) |>
  ggplot(aes(x, y, color = g)) +
  geom_point() +
  stat_smooth(method = lm, formula = y ~ x, linewidth = 1.5, se = FALSE) *
    (adjust(aes(group = g), color = "white", linewidth = 4) + 1) +
  scale_color_brewer(palette = "Dark2")

# (note this could also be done with copy_under())
```

`operation-class` *Layer operations*

Description

Layer operations are composable transformations that can be applied to **ggplot2** layer-like objects, such as `stats`, `geoms`, and lists of `stats` and `geoms`; see the layer-like documentation page for a description of valid layer-like objects.

Usage

```
## S4 method for signature 'operation'
show(object)

## S4 method for signature 'operation'
format(x, ...)

## S4 method for signature 'adjust'
format(x, ...)

## S4 method for signature 'affine_transform'
format(x, ...)

## S4 method for signature 'blend'
format(x, ...)

## S4 method for signature 'operation_composition'
format(x, ...)

## S4 method for signature 'nop'
format(x, ...)

## S4 method for signature 'operation_product'
format(x, ...)
```

Arguments

`x, object`	An operation.
`...`	Further arguments passed to other methods.

Details

operations can be composed using the `+` and `*` operators (see operation_sum and operation_product). Addition and multiplication of operations and layer-like objects obeys the distributive law.

operations can be applied to layer-like objects using `*` or `|>`, with slightly different results:

- Using `*`, application of operations to a list of layer-like objects *is* distributive. For example, `list(geom_line(), geom_point()) * blend("multiply")` is equivalent to `list(geom_line() * blend("multiply"), geom_point() * blend("multiply"))`; i.e. it multiply-blends the contents of the two layers individually.
- Using `|>`, application of operations to a list of layer-like objects is *not* distributive (unless the only reasonable interpretation of applying the transformation is necessarily distributive; e.g. `adjust()`). For example, `list(geom_line(), geom_point()) |> blend("multiply")` would multiply-blend both layers together, rather than multiply-blending the contents of the two layers individually.

Value

For `show()`, an `invisible()` copy of the input.

For `format()`, a character string representing the input.

Methods (by generic)

- `show(operation)`: Print an operation.
- `format(operation)`: Format an operation for printing.

Examples

```
library(ggplot2)

# operations can stand alone
adjust(aes(color = x))

# they can also be applied to layers through multiplication or piping
geom_line() |> adjust(aes(color = x))
geom_line() * adjust(aes(color = x))

# layer operations act as a small algebra, and can be combined through
# multiplication and addition
(adjust(fill = "green") + 1) * blend("multiply")
```

`operation_composition` *Layer operation composition*

Description

operations can be composed together to form chains of operations, which when multiplied by (applied to) layer-like objects, return modified layer-like objects. In contrast to operation_products, compositions of operations are not distributive over sums of operations or layer-like objects.

Details

Operation composition is achieved through function application, typically using the pipe operator (`|>`); e.g. `operation1 |> operation2`.

The output of composing **ggblend** operations depends on the types of objects being composed:

- If you compose an operation with an operation, they are merged into a single operation that applies each operation in sequence, without distributing over layers.
- If you compose an operation with a layer-like object, that operation is applied to the layer, returning a new layer-like object. The operation is applied to the layer as a whole, not any sub-parts (e.g. sub-layers or graphical objects).

Value

An operation.

Examples

```
library(ggplot2)

# composing operations together chains them
adjust(color = "red") |> blend("multiply")

# unlike multiplication, composition does not follow the distributive law
mult_op = (adjust(aes(y = 11 -x), color = "skyblue") + 1) * blend("multiply")
mult_op

comp_op = (adjust(aes(y = 11 -x), color = "skyblue") + 1) |> blend("multiply")
comp_op

# multiplication by a geom returns a modified version of that geom
data.frame(x = 1:10) |>
  ggplot(aes(x = x, y = x)) +
  geom_line(linewidth = 10, color = "red") * comp_op
```

---

operation_product *Layer operation products*

Description

operations can be multiplied together to form chains of operations, which when multiplied by (applied to) layer-like objects, return modified layer-like objects.

Usage

```
## S4 method for signature 'operation,ANY'
e1 * e2

## S4 method for signature 'ANY,operation'
e1 * e2

## S4 method for signature 'adjust,adjust'
e1 * e2

## S4 method for signature 'nop,nop'
e1 * e2

## S4 method for signature 'operation,nop'
e1 * e2

## S4 method for signature 'operation_sum,nop'
e1 * e2

## S4 method for signature 'nop,operation'
e1 * e2

## S4 method for signature 'nop,operation_sum'
e1 * e2

## S4 method for signature 'operation'
prod(x, ..., na.rm = FALSE)

## S4 method for signature 'operation,operation'
e1 * e2

## S4 method for signature 'numeric,operation'
e1 * e2

## S4 method for signature 'operation,numeric'
e1 * e2

## S4 method for signature 'operation,operation_sum'
e1 * e2

## S4 method for signature 'operation_sum,operation'
e1 * e2

## S4 method for signature 'operation_sum,operation_sum'
e1 * e2
```

Arguments

e1	an operation, layer-like, or numeric()
e2	an operation, layer-like, or numeric()
x, ...	operations
na.rm	ignored

Details

Multiplication of **ggblend** operations depends on the types of objects being multiplied:

- If you multiply an operation with an operation, they are merged into a single operation that applies each operation in sequence.
- If you multiply an operation with a layer-like object, that operation is applied to the layer, returning a new layer-like object.
- If you multiply an operation by a numeric() *n*, a new operation that repeats the input operation is *n* times is returned.

Value

An operation.

Examples

```
library(ggplot2)

# multiplying operations by numerics repeats them...
adjust(color = "red") * 2

# multiplying operations together chains (or merges) them
adjust(color = "red") * adjust(linewidth = 2)

# multiplication obeys the distributive law
op = (adjust(aes(y = 11 -x), color = "skyblue") + 1) * (adjust(color = "white", linewidth = 4) + 1)
op

# multiplication by a geom returns a modified version of that geom
data.frame(x = 1:10) |>
  ggplot(aes(x = x, y = x)) +
  geom_line(linewidth = 2) * op
```

operation_sum *Layer operation sums*

Description

operations can be added together to form stacks of operations, which when multiplied by (applied to) layer-like objects, those layer-like objects are distributed over the operations (i.e. copied).

Usage

```
## S4 method for signature 'operation'
sum(x, ..., na.rm = FALSE)

## S4 method for signature 'operation,operation'
e1 + e2

## S4 method for signature 'operation,numeric'
e1 + e2

## S4 method for signature 'numeric,operation'
e1 + e2

## S4 method for signature 'operation_sum'
format(x, ...)
```

Arguments

x, ...	operations
na.rm	ignored
e1	an operation or numeric()
e2	an operation or numeric()

Details

Addition of **ggblend** operations depends on the types of objects being summed:

- If you add an operation to an operation, they are merged into a single operation that copies input layer-like objects, one for each operation.
- If you add an operation to a numeric() *n*, it is equivalent to adding * nop()s to that operation.

Value

An operation.

Examples

```
library(ggplot2)

# adding operations together creates a sum of operations
adjust(color = "red") + adjust(linewidth = 2)

# addition and multiplication obey the distributive law
op = (adjust(aes(y = 11 -x), color = "skyblue") + 1) * (adjust(color = "white", linewidth = 4) + 1)
op

# multiplication by a geom returns a modified version of that geom,
# distributed over the sum of the operations
data.frame(x = 1:10) |>
  ggplot(aes(x = x, y = x)) +
  geom_line(linewidth = 2) * op
```

---

partition *Partition a layer into subgroups (Layer operation)*

Description

A layer operation for adding a `partition` aesthetic to a layer.

Usage

```
partition(object, partition)
```

Arguments

`object`	One of: • A layer-like object: applies this operation to the layer. • A missing argument: creates an operation • Anything else: creates an operation, passing `object` along to the `partition` argument
`partition`	One of: • A list of quosures, such as returned by `vars()`, giving a (possibly multi-) column expression for the `partition` aesthetic. These expressions are combined using `interaction()` to be passed on to `aes(partition = ...)` • A one-sided formula, giving a single-column expression for the `partition` aesthetic, which is passed on to `aes_(partition = ...)`.

Details

This is a shortcut for setting the `partition` aesthetic of a layer.

- `partition(~ XXX)` is roughly equivalent to `adjust(aes(partition = XXX))`

- `partition(vars(X, Y, ...))` is roughly equivalent to `adjust(aes(partition = interaction(X, Y, ...)))`

When a layer with a `partition` aesthetic is used by the following operations, the effects of the operations are applied across groups:

- `blend()`: Blends graphical objects within the subgroups defined by the partition together using normal (`"over"`) blending before applying its blend between subgroups.

Value

A layer-like object (if `object` is layer-like) or an operation (if not).

See Also

operation for a description of layer operations.

Other layer operations: `adjust`, `affine_transform`, `blend`, `copy`, `nop`

Examples

```
library(ggplot2)

# create two versions of a dataset, where draw order can affect output
set.seed(1234)
df_a = data.frame(x = rnorm(500, 0), y = rnorm(500, 1), set = "a")
df_b = data.frame(x = rnorm(500, 1), y = rnorm(500, 2), set = "b")
df_ab = rbind(df_a, df_b) |>
  transform(order = "draw a then b")
df_ba = rbind(df_b, df_a) |>
  transform(order = "draw b then a")
df = rbind(df_ab, df_ba)

# Using the "multiply" blend mode, draw order does not matter, but
# the "multiply" blend is applied to all points, creating dark regions
# outside the intersection:
df |>
  ggplot(aes(x, y, color = set)) +
  geom_point(size = 3, alpha = 0.5) |> blend("multiply") +
  scale_color_brewer(palette = "Set1") +
  facet_grid(~ order)

# By partitioning (either through |> partition(vars(set)) or aes(partition = set))
# we will blend using the default blend mode (over) first, then we can apply the
# "multiply" blend just between the two sets, so the regions outside the
# intersection are not blended using "multiply":
df |>
  ggplot(aes(x, y, color = set, partition = set)) +
  geom_point(size = 3, alpha = 0.5) |> blend("multiply") +
  scale_color_brewer(palette = "Set1") +
  facet_grid(~ order)
```

Index