Package 'ggfx'

July 22, 2025

Title Pixel Filters for 'ggplot2' and 'grid'

Version 1.0.1

Description Provides a range of filters that can be applied to layers from the 'ggplot2' package and its extensions, along with other graphic elements such as guides and theme elements. The filters are applied at render time and thus uses the exact pixel dimensions needed.

License MIT + file LICENSE

Encoding UTF-8

Imports magick (>= 2.7.1), ragg, grid, ggplot2, grDevices, gtable, rlang

RoxygenNote 7.2.1

URL https://ggfx.data-imaginist.com, https://github.com/thomasp85/ggfx

BugReports https://github.com/thomasp85/ggfx/issues

Suggests covr, knitr, rmarkdown, farver (>= 2.1.0)

VignetteBuilder knitr

NeedsCompilation no

Author Thomas Lin Pedersen [aut, cre] (ORCID: <https://orcid.org/0000-0002-5147-4711>), RStudio [cph, fnd]

Maintainer Thomas Lin Pedersen <thomasp85@gmail.com>

Repository CRAN

Date/Publication 2022-08-22 08:00:06 UTC

Contents

`as_colourspace` *Collect channels into a single layer of a specific colourspace*

Description

If you need to work on single channels one by one you can use the different ch_*() selectors. If the result needs to be combined again into a colour layer you can use `as_colourspace` and pass in the required channels to make up the colourspace. By default the alpha channel will be created as the combination of the alpha channels from the provided channel layers. Alternatively you can set `auto_opacity = FALSE` and provide one additional channel which will then be used as alpha.

Usage

```
as_colourspace(
  ...,
  colourspace = "sRGB",
  auto_opacity = TRUE,
  id = NULL,
  include = is.null(id)
)
```

Arguments

...	A range of layers to combine. If there are no channel spec set the luminosity will be used
colourspace	Which colourspace should the provided colour channels be interpreted as coming from.
auto_opacity	Should the opacity be derived from the input layers or taken from a provided alpha channel
id	A string identifying this layer for later use
include	Should the layer itself be included in the graphic

Value

A list of Layer objects

See Also

Other layer references: as_group(), as_reference()

Examples

```
library(ggplot2)

segments <- data.frame(
  x = runif(300),
  y = runif(300),
  xend = runif(300),
  yend = runif(300)
)

# We use 'white' as that is the maximum value in all channels
ggplot(mapping = aes(x, y, xend = xend, yend = yend)) +
  as_colourspace(
    geom_segment(data = segments[1:100,], colour = 'white'),
    geom_segment(data = segments[101:200,], colour = 'white'),
    geom_segment(data = segments[201:300,], colour = 'white'),
    colourspace = 'CMY'
  )
```

---

as_group *Collect layers into a group that can be treated as a single layer*

Description

While you often want to apply filters to layers one by one, there are times when one filter should be applied to a collection of layers as if they were one. This can be achieved by first combining all the layers into a group with as_group() and applying the filter to the resulting group. This can only be done to ggplot2 layers and grobs as the other supported objects are not part of a graphic stack.

Usage

```
as_group(..., id = NULL, include = is.null(id))
```

Arguments

...	A range of layers to combine
id	A string identifying this layer for later use
include	Should the layer itself be included in the graphic

Value

A list of Layer objects or a gTree depending on the input

See Also

Other layer references: as_colourspace(), as_reference()

Examples

```
library(ggplot2)

# With no grouping the filters on layers are applied one by one
ggplot(mtcars, aes(mpg, disp)) +
  with_shadow(geom_smooth(alpha = 1), sigma = 4) +
  with_shadow(geom_point(), sigma = 4)

# Grouping the layers allows you to apply a filter on the combined result
ggplot(mtcars, aes(mpg, disp)) +
  as_group(
    geom_smooth(alpha = 1),
    geom_point(),
    id = 'group_1'
  ) +
  with_shadow('group_1', sigma = 4)
```

---

as_reference *Create a reference to a layer for use in other filters*

Description

This function is basically synonymous with with_raster() but exist to make the intend of marking a layer with a specific id clear.

Usage

```
as_reference(x, id = NULL, include = is.null(id))
```

Arguments

x	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
id	A string identifying this layer for later use
include	Should the layer itself be included in the graphic

Value

Depending on the input, either a grob, Layer, list of Layers, guide, or element object. Assume the output can be used in the same context as the input.

See Also

Other layer references: as_colourspace(), as_group()

Examples

```
library(ggplot2)

ggplot() +
  as_reference(
    geom_point(aes(20, 300), size = 100, colour = 'white'),
    id = 'mask_layer'
  ) +
  with_mask(
    geom_point(aes(mpg, disp), mtcars, size = 5),
    mask = 'mask_layer'
  )
```

Channels *Set a channel of interest from a layer*

Description

Some effects uses a particular channel for specific parameters, such as with_displacement(), which grabs the relative x and y displacements from different channels in some other layer. To facilitate specifying which channel to use from a layer (which is always multichannel), you can wrap the specification in a channel specifier given below. If a filter requires a specific channel and none is specified it will default to luminance (based on the hcl colour space)

Usage

```
ch_red(x, colourspace = "sRGB", invert = FALSE)

ch_green(x, colourspace = "sRGB", invert = FALSE)
```

```
ch_blue(x, colourspace = "sRGB", invert = FALSE)

ch_alpha(x, colourspace = "sRGB", invert = FALSE)

ch_hue(x, colourspace = "HCL", invert = FALSE)

ch_chroma(x, colourspace = "HCL", invert = FALSE)

ch_luminance(x, colourspace = "HCL", invert = FALSE)

ch_saturation(x, colourspace = "HSL", invert = FALSE)

ch_lightness(x, colourspace = "HSL", invert = FALSE)

ch_cyan(x, colourspace = "CMYK", invert = FALSE)

ch_magenta(x, colourspace = "CMYK", invert = FALSE)

ch_yellow(x, colourspace = "CMYK", invert = FALSE)

ch_black(x, colourspace = "CMYK", invert = FALSE)

ch_key(x, colourspace = "CMYK", invert = FALSE)

ch_custom(x, channel, colourspace, invert = FALSE)
```

Arguments

x	Any object interpretable as a layer
colourspace	The colourspace the channel should be extracted from.
invert	Should the channel values be inverted before use
channel	The name of a channel in the given colourspace

Value

x with a channel spec attached

Examples

```
library(ggplot2)
volcano_long <- data.frame(
  x = as.vector(col(volcano)),
  y  = as.vector(row(volcano)),
  z = as.vector(volcano)
)

# invert the green channel
ggplot(volcano_long, aes(y, x)) +
  as_reference(
```

```
  geom_contour_filled(aes(z = z, fill = after_stat(level))),
  id = 'contours'
) +
as_colourspace(
  ch_red('contours'),
  ch_green('contours', invert = TRUE),
  ch_blue('contours')
)
```

`object_support` *Supported object types*

Description

The different filters provided by ggfx are applicable to a wide range of object types. Rather than documenting how to use them with each type in every documentation entry, the information is collected here. While the examples will use `with_blur()` they are general and applicable to all filters in ggfx.

Value

All filters will generally return a new version of the same object, the only exception being filtering of rasters, functions, and references which returns a Layer object

Method specific arguments

- `id`: A string that identifies the result of this filter, to be referenced by other filters in the same graphic.
- `include`: A logical flag that indicates whether the filtered image should be displayed. By default, the result will not be displayed if it is given an `id` (as it is assumed that it is meant for later use), but this can be overewritten by setting `include = TRUE`.
- `ignore_background`: Should the background of the plot be removed before applying the filter and re-added afterwards?
- `background`: A grob to draw below the result of the filter. Mainly for internal use for supporting `ignore_background`.

Filtering layers

This is perhaps the most common and obvious use of ggfx, and the one show-cased in the respective docs of each filter. In order to apply a filter to a ggplot2 layer you wrap it around the layer constructor (usually a `geom_*()` function) and pass in additional parameters after it:

```
ggplot(mtcars) +
  with_blur(
    geom_point(aes(x = mpg, y = disp)),
    sigma = 4
  )
```

Apart from the arguments specific to the filter, layer filters also take an `id`, and `include` argument. Providing an id (as a string) will make this filter be referable by other filters. By default this turns of rendering of the result, but setting `include = TRUE` will turn rendering back on (while still making it referable). Referable layers should **always** come before whatever other layer ends up referring to them, since ggfx does not have control over the rendering order. Not following this rule will have undefined consequences (either an error or a weird plot - or maybe the correct result)

Filtering layer references

While the first argument to a filter is mostly some sort of graphic generating object, it can also be a text string referring to another filter. This allows you to string together filters, should you so choose. The big caveat is that filtering a reference will always result in a layer - i.e. it is not compatible outside of ggplot2.

```
ggplot(mtcars) +
  with_blur(
    geom_point(aes(x = mpg, y = disp)),
    sigma = 4,
    id = 'blurred_points'
  ) +
  with_shadow(
    'blurred_points'
  )
```

Filtering guides

ggplot2 does not only consist of layers - there are all sort of other graphic elements around them. Guides are one such type of element and these can be filtered by wrapping the filter around the guide constructor:

```
ggplot(mtcars) +
  geom_point(aes(x = mpg, y = disp, colour = gear)) +
  guides(colour = with_blur(guide_colourbar(), sigma = 4))
```

There is a caveat here in that it is not possible to use this with the string shorthand (i.e. `with_blur('colourbar')` won't work) — you have to use the functional form.

Filtering theme elements

Theme elements, like guides, is another non-layer graphic that is amenable to filtering. It can be done by wrapping the `element_*()` constructor with a filter:

```
ggplot(mtcars) +
  geom_point(aes(x = mpg, y = disp)) +
  ggtitle("A blurry title") +
  theme(plot.title = with_blur(element_text(), sigma = 4))
```

There is a caveat here as well. The filtering doesn't get carried through inheritance so you cannot set filtering at a top-level element and expect all child elements to be filtered.

Filtering ggplots

While you normally only want to add a filter to a part of the plot, it is also possible to add it to everthing, simply by wrapping the filter function around the plot. You can elect to remove the background element while applying the filter and add it back on afterwards by setting `ignore_background = TRUE` on the filter

```
p <- ggplot(mtcars) +
  geom_point(aes(x = mpg, y = disp))

with_blur(p, sigma = 4)
```

An alternative is to put the filter around the `ggplot()` call, which will have the same effect and may fit better with your plot construction code

```
with_blur(ggplot(mtcars), sigma = 4) +
  geom_point(aes(x = mpg, y = disp))
```

Filtering grobs

At the lowest level, it is possible to apply a filter to a grob. This is what powers all of the above at some level and that power is also available to you. It is done in the same manner as all of the above, by wrapping the grob in a filter:

```
blurred_circle <- with_blur(circleGrob(), sigma = 4)

grid.newpage()
grid.draw(blurred_circle)
```

As with layers, filters applied to grobs also take an `id` and `include` argument and they have the same effect. It should be noted that it can be difficult to grasp the rendering order of elements in a manually created grid graphics, so take care when using filters that refer to each other as the rule about the rendering order still applies.

There are not a lot of people who use grid directly, but if you develop ggplot2 extensions the ability to apply filters to grobs means that you can create geoms with filters build right into them!

`raster_placement` *Control placements of raster in the plot*

Description

When using raster objects directly you need to somehow define how it should be located in resized in the plot. These function can be used to inform the filter on how it should be used. They only work on `raster` type object, so cannot be used around functions or layer id's.

Usage

```
ras_fill(raster, align_to = "canvas")

ras_fit(raster, align_to = "canvas")

ras_stretch(raster, align_to = "canvas")

ras_place(raster, align_to = "canvas", anchor = "topleft", offset = c(0, 0))

ras_tile(
  raster,
  align_to = "canvas",
  anchor = "topleft",
  offset = c(0, 0),
  flip = FALSE
)
```

Arguments

raster	A raster or nativeRaster object or an object coercible to a raster object
align_to	Should the raster be positioned according to the canvas or the current viewport
anchor	Where should the raster be placed relative to the alignment area
offset	A unit or numeric vector giving an additional offset relative to the anchor. Positive values moves right/down and negative values move left/up
flip	Should every other repetition be flipped

Value

The input with additional information attached

Examples

```
library(ggplot2)
logo <- as.raster(magick::image_read(
  system.file('help', 'figures', 'logo.png', package = 'ggfx')
))

# Default is to fill the viewport area, preserving the aspect ratio of the
# raster
ggplot(mtcars) +
  with_blend(
    geom_point(aes(mpg, disp)),
    logo
  )

# But you can change that with these functions:
ggplot(mtcars) +
  with_blend(
    geom_point(aes(mpg, disp)),
```

```
    ras_place(logo, 'vp', 'bottomright')
  )

# Here we tile it with flipping, centering on the middle of the canvas
ggplot(mtcars) +
  with_blend(
    geom_point(aes(mpg, disp)),
    ras_tile(logo, anchor = 'center', flip = TRUE)
  )
```

`render_context` *Rendering information*

Description

These utility functions can help when creating custom filters (using `with_custom()`) as they can provide information about the current rendering context.

Usage

```
viewport_location()

index_raster(raster, cols, rows)

get_raster_area(raster, xmin, ymin, xmax, ymax)

set_raster_area(raster, value, xmin, ymin)

get_viewport_area(raster)

set_viewport_area(raster, value)

viewport_is_clipping()

current_resolution()

to_pixels(x, y_axis = FALSE, location = FALSE)

from_pixels(x)
```

Arguments

`raster`	A `raster` or `nativeRaster` object
`cols, rows`	Column and row indices
`xmin, ymin, xmax, ymax`	Boundaries of the area in pixels. 0,0 is the top-left corner

value	An object of the same type as raster
x	A numeric or unit object
y_axis	is the unit pertaining to the y-axis? Defaults to FALSE (i.e. it is measured on the x-axis)
location	is the unit encoding a location? Defaults to FALSE (i.e. it is encoding a dimension). Pixel locations are encoded based on a top-left starting point, as opposed to grid's bottom-left coordinate system. This means that y-axis locations will flip around when converted to pixels.

Details

- viewport_location(): Returns the bounding box defining the current viewport in pixels in the order xmin, ymin, xmax, ymax
- index_raster(): Is a version of the classic [,] indexing that is aware of the row-major order of rasters
- get_raster_area(): Extracts an area of a raster based on a bounding box
- set_raster_area(): Sets an area of a raster to a new raster value
- get_viewport_area(): A version of get_raster_area() that specifically extract the area defined by the current viewport
- set_viewport_area(): A version of set_raster_area() that specifically sets the area defined by the current viewport
- viewport_is_clipping(): Returns TRUE if the current viewport has clipping turned on
- current_resolution(): Returns the resolution of the active device in ppi (pixels-per-inch)
- to_pixels(x): Converts x to pixels if x is given as a unit object. It is assumed that x encodes a dimension and not a location. If x is a numeric it is assumed to already be in pixels
- from_pixels: Converts a numeric giving some pixel dimension to a unit object.

Value

Depends on the function - see details.

Examples

```
# These functions are intended to be used inside filter functions, e.g.
library(ggplot2)

flip_raster <- function(raster, horizontal = TRUE) {
  # Get the viewport area of the raster
  vp <- get_viewport_area(raster)

  # Get the columns and rows of the raster - reverse order depending on
  # the value of horizontal
  dims <- dim(vp)
  rows <- seq_len(dims[1])
  cols <- seq_len(dims[2])
  if (horizontal) {
    cols <- rev(cols)
```

```
  } else {
    rows <- rev(rows)
  }

  # change the order of columns or rows in the viewport raster
  vp <- index_raster(vp, cols, rows)

  # Assign the modified viewport back
  set_viewport_area(raster, vp)
}

ggplot() +
  with_custom(
    geom_text(aes(0.5, 0.75, label = 'Flippediflop!'), size = 10),
    filter = flip_raster,
    horizontal = TRUE
  )
```

`with_blend` *Blend a layer with a reference*

Description

This filter blends the layer with a reference according to one of many rules as laid out in the *Details* section.

Usage

```
with_blend(
  x,
  bg_layer,
  blend_type = "over",
  flip_order = FALSE,
  alpha = NA,
  ...
)
```

Arguments

`x`	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
`bg_layer`	The background layer to use. Can either be a string identifying a registered filter, or a raster object. The map will be resized to match the dimensions of `x`.
`blend_type`	The type of blending to perform. See *Details*
`flip_order`	Should the order of the background and the overlay be flipped so that `bg_layer` is treated as being on top and `x` being below.

`alpha`	For non-Duff-Porter blends the alpha channel may become modified. This argument can be used to set the resulting alpha channel to that of the source (`"src"`) or destination (`"dst"`)
`...`	Arguments to be passed on to methods. See the documentation of supported object for a description of object specific arguments.

Details

Two images can be blended in a variety of ways as described below. In the following *source* will refer to the top-most image, and *destination* to the bottom-most image. Note that which is which can be controlled with the `flip_order` argument.

Duff-Porter alpha blend modes:

This is a set of well-defined blend types for composing two images, taking their opacity into account:

- `"source"`: Completely disregards the destination, leaving only the source
- `"destination"`: Completely disregards the source, leaving only the destination
- `"clear"`: Disregards both destination and source
- `"xor"`: Composes source on top of destination, setting shared areas to transparent
- `"over"`: Composes source on top of destination
- `"in"`: Shows source, but only where the destination is opaque
- `"out"`: Shows source but only where the destination is transparent
- `"atop"`: Composes source on top of destination, keeping the transparency of destination
- `"copy"`: Like source, but will only affect the area occupied by the source image

Mathmathical blend modes:

These blend modes perform often complex channel operations based on the different channel values in the source and destination:

- `"multiply"`: Multiplies the channel values of source and destination together (after scaling them to 0-1) to obtain new channel values
- `"screen"`: As multiply except that the channels are scaled to 1-0 before multiplication, and the result is reversed again before being used
- `"bumpmap"`: Like multiple, except source is converted to greyscale first
- `"divide"`: Divide the channel values in source by the channel values in destination
- `"plus"`: Add the channel values together *including the alpha channel*
- `"minus"`: Subtracts the destination channels from the source channels
- `"modulus_plus"`: As plus, but overflow will wrap around instead of being capped
- `"modulus_minus"`: As minus but overflow (underflow) will wrap around instead of being capped
- `"difference"`: Takes the absolute difference in channel values between source and destination
- `"exclusion"`: `source + destination - 2*source*destination`. A sort of averaged difference
- `"lighten"`: Will pick the lightest pixel at each pixel
- `"darken"`: Will pick the darkest pixel at each pixel

- "lighten_intensity": Will pick the most intense colour at each pixel
- "darken_intensity": Will pick the least intense colour at each pixel

Lighting blend modes:

These blend modes are designed to provide different lighting effects:

- "overlay": Simultaneously multiplies and screens at the same time based on the colour values of the destination. Will colorize midtones in the destination with the source
- "hard_light": The inverse of overlay (i.e. the source acts as the destination and vice versa)
- "soft_light": Like overlay but will extent the range of colorization past the midtones
- "pegtop_light": Like soft-light, but without any discontinuity in the blend
- "linear_light": Combines dodging and burning so that the destination will be dodged (lightened) when the source is light and burned (darkened) when the source is dark
- "vivid_light": A refinement of linear-light that better avoids shading intense colours
- "pin_light": Preserves midtones of the destination and only shades lighter and darker parts, resulting in harsh, contrasty lightning.
- "linear_dodge": Lighten the destination if the source is light
- "linear_burn": Darked the destination if the source is dark
- "color_dodge": Like linear-dodge, but preserves blacks in the destination image
- "color_burn": Like linear-burn but preserve whites in the destination image

Channel copying blends:

These blend modes copies a single channel from the source to the destination

- "copy_opacity": Will set the opacity of destination to the grayscale version of source. To copy the opacity of source into destination use blend_type = "in" with flip_order = TRUE.
- "copy_red": Copies the red channel in source into the red channel in destination
- "copy_green": Copies the green channel in source into the green channel in destination
- "copy_blue": Copies the blue channel in source into the blue channel in destination
- "hue": Replaces the hue of the destination with the hue of the source
- "saturate": Replaces the saturation of the destination with the saturation of the source
- "luminize": Replaces the luminance of the destination with the luminance of the source
- "colorize": Combines hue and saturate

Special blends:

- "unique": Only keep pixels in the source that differ from the destination.

The above is obviously a very quick overview. More information can be found in https://legacy.imagemagick.org/Usage/compose/

Value

Depending on the input, either a grob, Layer, list of Layers, guide, or element object. Assume the output can be used in the same context as the input.

See Also

Other blend filters: with_blend_custom(), with_interpolate(), with_mask()

Examples

```
library(ggplot2)
ggplot() +
  as_reference(
    geom_text(aes(0.5, 0.5, label = 'Blend Modes!'), size = 10, fontface = 'bold'),
    id = "text"
  ) +
  with_blend(
    geom_polygon(aes(c(0, 1, 1), c(0, 0, 1)), colour = NA, fill = 'magenta'),
    bg_layer = "text",
    blend_type = 'xor'
  )
```

`with_blend_custom` *Create a custom blend type*

Description

Many of the blend types available in `with_blend()` are variations over the formula: `a*src*dst + b*src + c*dst + d`, where `src` stands for the channel value in the source image and `dst` stands for the destination image (the background). Multiply is e.g. defined as `a:1, b:0, c:0, d:0`. This filter gives you free reign over setting the coefficient of the blend calculation.

Usage

```
with_blend_custom(
  x,
  bg_layer,
  a = 0,
  b = 0,
  c = 0,
  d = 0,
  flip_order = FALSE,
  alpha = NA,
  ...
)
```

Arguments

`x`	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
`bg_layer`	The background layer to use. Can either be a string identifying a registered filter, or a raster object. The map will be resized to match the dimensions of `x`.
`a, b, c, d`	The coefficients defining the blend operation
`flip_order`	Should the order of the background and the overlay be flipped so that `bg_layer` is treated as being on top and `x` being below.

alpha	For non-Duff-Porter blends the alpha channel may become modified. This argument can be used to set the resulting alpha channel to that of the source ("src") or destination ("dst")
...	Arguments to be passed on to methods. See the documentation of supported object for a description of object specific arguments.

Value

Depending on the input, either a grob, Layer, list of Layers, guide, or element object. Assume the output can be used in the same context as the input.

See Also

Other blend filters: with_blend(), with_interpolate(), with_mask()

Examples

```
library(ggplot2)
ggplot(mpg, aes(class, hwy)) +
  as_reference(geom_boxplot(fill = 'green'), 'box') +
  with_blend_custom(geom_point(colour = 'red'),
                    bg_layer = 'box', a = -0.5, b = 1, c = 1)
```

---

with_bloom *Apply bloom to your layer*

Description

Bloom is the effect of strong light sources spilling over into neighbouring dark areas. It is used a lot in video games and movies to give the effect of strong light, even though the monitor is not itself capable of showing light at that strength.

Usage

```
with_bloom(
  x,
  threshold_lower = 80,
  threshold_upper = 100,
  sigma = 5,
  strength = 1,
  keep_alpha = TRUE,
  ...
)
```

Arguments

x	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
threshold_lower, threshold_upper	The lowest channel value to consider emitting light and the highest channel value that should be considered maximum light strength, given in percent
sigma	The standard deviation of the gaussian kernel used for the bloom. Will affect the size of the halo around light objects
strength	A value between 0 and 1 to use for changing the strength of the effect.
keep_alpha	Should the alpha channel of the layer be kept, effectively limiting the bloom effect to the filtered layer. Setting this to false will allow the bloom to spill out to the background, but since it is not being blended correctly with the background the effect looks off.
...	Arguments to be passed on to methods. See the documentation of supported object for a description of object specific arguments.

Value

Depending on the input, either a grob, Layer, list of Layers, guide, or element object. Assume the output can be used in the same context as the input.

Examples

```
library(ggplot2)
points <- data.frame(
  x = runif(1000),
  y = runif(1000),
  col = runif(1000)
)
ggplot(points, aes(x, y, colour = col)) +
  with_bloom(
    geom_point(size = 10),
  ) +
  scale_colour_continuous(type = 'viridis')
```

---

with_blur *Apply a gaussian blur to your layer*

Description

This filter adds a blur to the provided ggplot layer. The amount of blur can be controlled and the result can optionally be put underneath the original layer.

Usage

```
with_blur(x, sigma = 0.5, stack = FALSE, ...)
```

Arguments

x	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
sigma	The standard deviation of the gaussian kernel. Increase it to apply more blurring. If a numeric it will be interpreted as given in pixels. If a unit object it will automatically be converted to pixels at rendering time
stack	Should the original layer be placed on top?
...	Arguments to be passed on to methods. See the documentation of supported object for a description of object specific arguments.

Value

Depending on the input, either a `grob`, `Layer`, list of `Layers`, `guide`, or `element` object. Assume the output can be used in the same context as the input.

See Also

Other blur filters: `with_motion_blur()`, `with_variable_blur()`

Examples

```
library(ggplot2)
ggplot(mtcars, aes(mpg, disp)) +
  with_blur(geom_point(data = mtcars, size = 3), sigma = 3)
```

---

with_circle_dither *Dither image using a threshold dithering map*

Description

These filters reduces the number of colours in your layer and uses various threshold maps along with a dithering algorithm to disperse colour error.

Usage

```
with_circle_dither(
  x,
  map_size = 7,
  levels = NULL,
  black = TRUE,
  colourspace = "sRGB",
  offset = NULL,
  ...
)

with_custom_dither(
```

```
  x,
  map = "checks",
  levels = NULL,
  colourspace = "sRGB",
  offset = NULL,
  ...
)

with_halftone_dither(
  x,
  map_size = 8,
  levels = NULL,
  angled = TRUE,
  colourspace = "sRGB",
  offset = NULL,
  ...
)

with_ordered_dither(x, map_size = 8, levels = NULL, colourspace = "sRGB", ...)
```

Arguments

x	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
map_size	One of 2, 3, 4, or 8. Sets the threshold map used for dithering. The larger, the better approximation of the input colours
levels	The number of threshold levels in each channel. Either a single integer to set the same number of levels in each channel, or 3 values to set the levels individually for each colour channel
black	Should the map consist of dark circles expanding into the light, or the reverse
colourspace	In which colourspace should the dithering be calculated
offset	The angle offset between the colour channels
...	Arguments to be passed on to methods. See the documentation of supported object for a description of object specific arguments.
map	The name of the threshold map to use as understood by `magick::image_ordered_dither()`
angled	Should the halftone pattern be at an angle or orthogonal

Value

Depending on the input, either a `grob`, `Layer`, list of `Layers`, `guide`, or `element` object. Assume the output can be used in the same context as the input.

See Also

Other dithering filters: `with_dither()`

Examples

```
library(ggplot2)

# Ordered dither
ggplot(faithfuld, aes(waiting, eruptions)) +
  with_ordered_dither(
    geom_raster(aes(fill = density), interpolate = TRUE)
  ) +
  scale_fill_continuous(type = 'viridis')

# Halftone dither
ggplot(faithfuld, aes(waiting, eruptions)) +
  with_halftone_dither(
    geom_raster(aes(fill = density), interpolate = TRUE)
  ) +
  scale_fill_continuous(type = 'viridis')

# Circle dither with offset
ggplot(faithfuld, aes(waiting, eruptions)) +
  with_circle_dither(
    geom_raster(aes(fill = density), interpolate = TRUE),
    offset = 29,
    colourspace = 'cmyk'
  ) +
  scale_fill_continuous(type = 'viridis')
```

---

with_custom *Apply a custom filter*

Description

This function allows you to apply a custom filtering function to a layer. The function must take a `nativeRaster` object as the first argument along with any other arguments passed to `....` Be aware that the raster spans the full device size and not just the viewport currently rendered to. This is because graphics may extend outside of the viewport depending on the clipping settings. You can use `get_viewport_area()` along with all the other raster helpers provided by ggfx to facilitate working with the input raster. See the example below for some inspiration.

Usage

```
with_custom(x, filter, ...)
```

Arguments

`x`	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
`filter`	A function taking a `nativeRaster` object as the first argument along with whatever you pass in to `...`
`...`	Additional arguments to `filter`

Value

Depending on the input, either a `grob`, `Layer`, list of `Layers`, `guide`, or `element` object. Assume the output can be used in the same context as the input.

Examples

```
library(ggplot2)
flip_raster <- function(raster, horizontal = TRUE) {
  # Get the viewport area of the raster
  vp <- get_viewport_area(raster)

  # Get the columns and rows of the raster - reverse order depending on
  # the value of horizontal
  dims <- dim(vp)
  rows <- seq_len(dims[1])
  cols <- seq_len(dims[2])
  if (horizontal) {
    cols <- rev(cols)
  } else {
    rows <- rev(rows)
  }

  # change the order of columns or rows in the viewport raster
  vp <- index_raster(vp, cols, rows)

  # Assign the modified viewport back
  set_viewport_area(raster, vp)
}

ggplot() +
  with_custom(
    geom_text(aes(0.5, 0.75, label = 'Flippediflop!'), size = 10),
    filter = flip_raster,
    horizontal = TRUE
  )

ggplot() +
  with_custom(
    geom_text(aes(0.5, 0.75, label = 'Flippediflop!'), size = 10),
    filter = flip_raster,
    horizontal = FALSE
  )
```

`with_displacement` *Apply a displacement map to a layer*

Description

This filter displaces the pixels based on the colour values of another layer or raster object. As such it can be used to distort the content of the layer.

Usage

```
with_displacement(x, x_map, y_map = x_map, x_scale = 1, y_scale = x_scale, ...)
```

Arguments

x	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
x_map, y_map	The displacement maps to use. Can either be a string identifying a registered filter, or a raster object. The maps will be resized to match the dimensions of x. Only one channel will be used - see the docs on channels for info on how to set them.
x_scale, y_scale	How much displacement should a maximal channel value correspond to? If a numeric it will be interpreted as pixel dimensions. If a unit object it will be converted to pixel dimension when rendered.
...	Arguments to be passed on to methods. See the documentation of supported object for a description of object specific arguments.

Value

Depending on the input, either a grob, Layer, list of Layers, guide, or element object. Assume the output can be used in the same context as the input.

Examples

```
library(ggplot2)
ggplot() +
  as_reference(
    geom_polygon(aes(c(0, 1, 1), c(0, 0, 1)), colour = NA, fill = 'magenta' ),
    id = "displace_map"
  ) +
  with_displacement(
    geom_text(aes(0.5, 0.5, label = 'Displacements!'), size = 10),
    x_map = ch_red("displace_map"),
    y_map = ch_blue("displace_map"),
    x_scale = unit(0.025, 'npc'),
    y_scale = unit(0.025, 'npc')
  )
```

---

with_dither *Dither image using Floyd-Steinberg error correction dithering*

Description

This filter reduces the number of colours in your layer and uses the Floyd-Steinberg algorithm to even out the error introduced by the colour reduction.

Usage

```
with_dither(x, max_colours = 256, colourspace = "sRGB", ...)
```

Arguments

x	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
max_colours	The maximum number of colours to use. The result may contain fewer colours but never more.
colourspace	In which colourspace should the dithering be calculated
...	Arguments to be passed on to methods. See the documentation of supported object for a description of object specific arguments.

Value

Depending on the input, either a grob, Layer, list of Layers, guide, or element object. Assume the output can be used in the same context as the input.

See Also

Other dithering filters: with_circle_dither()

Examples

```
library(ggplot2)
ggplot(faithfuld, aes(waiting, eruptions)) +
  with_dither(
    geom_raster(aes(fill = density), interpolate = TRUE),
    max_colours = 10
  ) +
  scale_fill_continuous(type = 'viridis')
```

---

with_inner_glow *Apply an inner glow to your layer*

Description

This filter adds an inner glow to your layer with a specific colour and size. The best effect is often had by drawing the stroke separately so the glow is only applied to the fill.

Usage

```
with_inner_glow(x, colour = "black", sigma = 3, expand = 0, ...)
```

Arguments

x	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
colour	The colour of the glow
sigma	The standard deviation of the gaussian kernel. Increase it to apply more blurring. If a numeric it will be interpreted as given in pixels. If a unit object it will automatically be converted to pixels at rendering time
expand	An added dilation to the glow mask before blurring it
...	Arguments to be passed on to methods. See the documentation of supported object for a description of object specific arguments.

Value

Depending on the input, either a `grob`, `Layer`, list of `Layers`, `guide`, or `element` object. Assume the output can be used in the same context as the input.

See Also

Other glow filters: `with_outer_glow()`

Examples

```
library(ggplot2)

ggplot(mtcars, aes(as.factor(gear), disp)) +
  with_inner_glow(
    geom_boxplot(),
    colour = 'red',
    sigma = 10
  )

# This gives a red tone to the lines as well which may not be desirable
# This can be fixed by drawing fill and stroke separately
ggplot(mtcars, aes(as.factor(gear), disp)) +
  with_inner_glow(
    geom_boxplot(colour = NA),
    colour = 'red',
    sigma = 10
  ) +
  geom_boxplot(fill = NA)
```

---

with_interpolate *Blend two layerrs together by averaging them out*

Description

Two layers can be blended together in the literal sense (not like `with_blend()`) so that the result is the average of the two. This is the purpose of `with_interpolate()`.

Usage

```
with_interpolate(x, bg_layer, src_percent, bg_percent = 100 - src_percent, ...)
```

Arguments

x	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
bg_layer	The layer to blend with
src_percent, bg_percent	The contribution of this layer and the background layer to the result. Should be between 0 and 100
...	Arguments to be passed on to methods. See the documentation of supported object for a description of object specific arguments.

Value

Depending on the input, either a grob, Layer, list of Layers, guide, or element object. Assume the output can be used in the same context as the input.

See Also

Other blend filters: with_blend_custom(), with_blend(), with_mask()

Examples

```
library(ggplot2)
ggplot(mpg, aes(class, hwy)) +
  as_reference(geom_boxplot(), 'box') +
  with_interpolate(geom_point(), bg_layer = 'box', src_percent = 70)
```

---

with_kernel *Apply a gaussian blur to your layer*

Description

This filter allows you to apply a custom kernel to your layer, thus giving you more control than e.g. with_blur() which is also applying a kernel.

Usage

```
with_kernel(
  x,
  kernel = "Gaussian:0x2",
  iterations = 1,
  scaling = NULL,
  bias = NULL,
  stack = FALSE,
  ...
)
```

Arguments

x	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
kernel	either a square matrix or a string. The string can either be a parameterized kerneltype such as: "DoG:0,0,2" or "Diamond" or it can contain a custom matrix (see examples)
iterations	number of iterations
scaling	string with kernel scaling. The special flag "!" automatically scales to full dynamic range, for example: "50%!"
bias	output bias string, for example "50%"
stack	Should the original layer be placed on top?
...	Arguments to be passed on to methods. See the documentation of supported object for a description of object specific arguments.

Value

Depending on the input, either a grob, Layer, list of Layers, guide, or element object. Assume the output can be used in the same context as the input.

Examples

```
library(ggplot2)
# Add directional blur using the comet kernel
ggplot(mtcars, aes(mpg, disp)) +
  with_kernel(geom_point(size = 3), 'Comet:0,10')
```

---

with_mask *Apply a mask to a layer*

Description

This filter applies a mask to the given layer, i.e. sets the opacity of the layer based on another layer

Usage

```
with_mask(x, mask, invert = FALSE, ...)
```

Arguments

x	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
mask	The layer to use as mask. Can either be a string identifying a registered filter, or a raster object. Will by default extract the luminosity of the layer and use that as mask. To pick another channel use one of the channel specification function.
invert	Should the mask be inverted before applying it
...	Arguments to be passed on to methods. See the documentation of supported object for a description of object specific arguments.

Value

Depending on the input, either a `grob`, `Layer`, list of `Layers`, `guide`, or `element` object. Assume the output can be used in the same context as the input.

See Also

Other blend filters: `with_blend_custom()`, `with_blend()`, `with_interpolate()`

Examples

```
library(ggplot2)
volcano_raster <- as.raster((volcano - min(volcano))/diff(range(volcano)))
circle <- data.frame(
  x = cos(seq(0, 2*pi, length.out = 360)),
  y = sin(seq(0, 2*pi, length.out = 360))
)

ggplot() +
  as_reference(
    geom_polygon(aes(x = x, y = y), circle),
    id = 'circle'
  ) +
  with_mask(
    annotation_raster(volcano_raster, -1, 1, -1, 1, TRUE),
    mask = ch_alpha('circle')
  )

# use invert = TRUE to flip the mask
ggplot() +
  as_reference(
    geom_polygon(aes(x = x, y = y), circle),
    id = 'circle'
  ) +
  with_mask(
    annotation_raster(volcano_raster, -1, 1, -1, 1, TRUE),
    mask = ch_alpha('circle'),
    invert = TRUE
  )
```

---

with_motion_blur *Apply a motion blur to your layer*

Description

This filter adds a directional blur to the provided ggplot layer. The amount of blur, as well as the angle, can be controlled.

Usage

```
with_motion_blur(x, sigma = 0.5, angle = 0, ...)
```

Arguments

x	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
sigma	The standard deviation of the gaussian kernel. Increase it to apply more blurring. If a numeric it will be interpreted as given in pixels. If a unit object it will automatically be converted to pixels at rendering time
angle	Direction of the movement in degrees (0 corresponds to a left-to-right motion and the angles move in clockwise direction)
...	Arguments to be passed on to methods. See the documentation of supported object for a description of object specific arguments.

Value

Depending on the input, either a `grob`, `Layer`, list of `Layers`, `guide`, or `element` object. Assume the output can be used in the same context as the input.

See Also

Other blur filters: `with_blur()`, `with_variable_blur()`

Examples

```
library(ggplot2)
ggplot(mtcars, aes(mpg, disp)) +
  with_motion_blur(
    geom_point(size = 3),
    sigma = 6,
    angle = -45
  )
```

---

with_outer_glow *Apply an outer glow to your layer*

Description

This filter adds an outer glow to your layer with a specific colour and size. For very thin objects such as text it may be beneficial to add some expansion. See the examples for this.

Usage

```
with_outer_glow(x, colour = "black", sigma = 3, expand = 0, ...)
```

Arguments

x	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
colour	The colour of the glow
sigma	The standard deviation of the gaussian kernel. Increase it to apply more blurring. If a numeric it will be interpreted as given in pixels. If a unit object it will automatically be converted to pixels at rendering time
expand	An added dilation to the glow mask before blurring it
...	Arguments to be passed on to methods. See the documentation of supported object for a description of object specific arguments.

Value

Depending on the input, either a grob, Layer, list of Layers, guide, or element object. Assume the output can be used in the same context as the input.

See Also

Other glow filters: with_inner_glow()

Examples

```
library(ggplot2)

ggplot(mtcars, aes(as.factor(gear), disp)) +
  with_outer_glow(
    geom_boxplot(),
    colour = 'red',
    sigma = 10
  )

# For thin objects (as the whiskers above) you may need to add a bit of
# expansion to make the glow visible:

ggplot(mtcars, aes(mpg, disp)) +
  geom_point() +
  with_outer_glow(
    geom_text(aes(label = rownames(mtcars))),
    colour = 'white',
    sigma = 10,
    expand = 10
  )
```

with_raster *Convert a layer to a raster*

Description

This filter simply converts the given layer, grob, or ggplot to a raster and inserts it back again. It is useful for vector graphics devices such as svglite if a layer contains a huge amount of primitives that would make the file slow to render. `as_reference(x, id)` is a shorthand for `with_raster(x, id = id, include = FALSE)` that makes the intent of using this grob or layer as only a filter reference clear.

Usage

```
with_raster(x, ...)
```

Arguments

`x`	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
`...`	Arguments to be passed on to methods. See the documentation of supported object for a description of object specific arguments.

Value

Depending on the input, either a `grob`, `Layer`, list of `Layers`, `guide`, or `element` object. Assume the output can be used in the same context as the input.

Examples

```
library(ggplot2)
ggplot(mtcars, aes(mpg, disp)) +
  with_raster(geom_point(data = mtcars, size = 3))
```

with_shade *Apply a gaussian blur to your layer*

Description

This filter adds a blur to the provided ggplot layer. The amount of blur can be controlled and the result can optionally be put underneath the original layer.

Usage

```
with_shade(
  x,
  height_map,
  azimuth = 30,
  elevation = 30,
  strength = 10,
  sigma = 0,
  blend_type = "overlay",
  ...
)
```

Arguments

x	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
height_map	The layer to use as a height_map. Can either be a string identifying a registered filter, or a raster object. Will by default extract the luminosity of the layer and use that as mask. To pick another channel use one of the channel specification function.
azimuth, elevation	The location of the light source.
strength	The strength of the shading. A numeric larger or equal to 1
sigma	The sigma used for blurring the shading before applying it. Setting it to 0 turns off blurring. Using a high strength may reveal artefacts in the calculated shading, especially if the height_map is low-detail. Adding a slight blur may remove some of those artefacts.
blend_type	A blend type as used in with_blend() for adding the calculated shading to the layer. Should generally be left as-is
...	Arguments to be passed on to methods. See the documentation of supported object for a description of object specific arguments.

Value

Depending on the input, either a grob, Layer, list of Layers, guide, or element object. Assume the output can be used in the same context as the input.

Examples

```
library(ggplot2)
volcano_long <- data.frame(
  x = as.vector(col(volcano)),
  y  = as.vector(row(volcano)),
  z = as.vector(volcano)
)
ggplot(volcano_long, aes(y, x)) +
  as_reference(
    geom_raster(aes(alpha = z), fill = 'black', interpolate = TRUE, show.legend = FALSE),
    id = 'height_map'
```

```
  ) +
  with_shade(
    geom_contour_filled(aes(z = z, fill = after_stat(level))),
    height_map = ch_alpha('height_map'),
    azimuth = 150,
    height = 5,
    sigma = 10
  ) +
  coord_fixed() +
  guides(fill = guide_coloursteps(barheight = 10))
```

`with_shadow` *Apply a drop shadow to a layer*

Description

This filter applies the familiar drop-shadow effect on elements in a layer. It takes the outline of each shape, offsets it from its origin and applies a blur to it.

Usage

```
with_shadow(
  x,
  colour = "black",
  x_offset = 10,
  y_offset = 10,
  sigma = 1,
  stack = TRUE,
  ...
)
```

Arguments

`x`	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
`colour`	The colour of the shadow
`x_offset, y_offset`	The offset of the shadow from the origin as numerics
`sigma`	The standard deviation of the gaussian kernel. Increase it to apply more blurring. If a numeric it will be interpreted as given in pixels. If a unit object it will automatically be converted to pixels at rendering time
`stack`	Should the original layer be placed on top?
`...`	Arguments to be passed on to methods. See the documentation of supported object for a description of object specific arguments.

Value

Depending on the input, either a `grob`, `Layer`, list of `Layers`, `guide`, or `element` object. Assume the output can be used in the same context as the input.

Examples

```
library(ggplot2)
ggplot(mtcars, aes(mpg, disp)) +
  with_shadow(geom_point(colour = 'red', size = 3), sigma = 3)
```

`with_variable_blur` *Apply a variable blur to a layer*

Description

This filter will blur a layer, but in contrast to `with_blur()` the amount and nature of the blur need not be constant across the layer. The blurring is based on a weighted ellipsoid, with width and height based on the values in the corresponding `x_sigma` and `y_sigma` layers. The angle of the ellipsoid can also be controlled and further varied based on another layer.

Usage

```
with_variable_blur(
  x,
  x_sigma,
  y_sigma = x_sigma,
  angle = NULL,
  x_scale = 1,
  y_scale = x_scale,
  angle_range = 0,
  ...
)
```

Arguments

`x`	A ggplot2 layer object, a ggplot, a grob, or a character string naming a filter
`x_sigma`, `y_sigma`, `angle`	The layers to use for looking up the sigma values and angledefining the blur ellipse at every point. Can either be a string identifying a registered filter, or a raster object. The maps will be resized to match the dimensions of x. Only one channel will be used - see the docs on channels for info on how to set them.
`x_scale`, `y_scale`	Which sigma should a maximal channel value correspond to? If a numeric it will be interpreted as pixel dimensions. If a unit object it will be converted to pixel dimension when rendered.

angle_range	The minimum and maximum angle that min and max in the angle layer should correspond to. If angle == NULL or only a single value is provided to angle_range the rotation will be constant across the whole layer
...	Arguments to be passed on to methods. See the documentation of supported object for a description of object specific arguments.

Value

Depending on the input, either a grob, Layer, list of Layers, guide, or element object. Assume the output can be used in the same context as the input.

See Also

Other blur filters: with_blur(), with_motion_blur()

Examples

```
library(ggplot2)
cos_wave <- function(width, height) {
  x <- matrix(0, ncol = width, nrow = height)
  x <- cos(col(x)/100)
  as.raster((x + 1) / 2)
}
ggplot() +
  as_reference(
    cos_wave,
    id = "wave"
  ) +
  with_variable_blur(
    geom_point(aes(disp, mpg), mtcars, size = 4),
    x_sigma = ch_red("wave"),
    y_sigma = ch_alpha("wave"),
    angle = ch_red("wave"),
    x_scale = 15,
    y_scale = 15,
    angle_range = c(-45, 45)
  )
```

Index