Package: ggWebGL (via r-universe)

June 6, 2026

Title Browser-Native 'WebGL' Rendering for R Graphics

Date 2026-06-02

Version 0.8.0

Author Frederic Bertrand [cre, aut]
(<https://orcid.org/0000-0002-0837-8281>)

Maintainer Frederic Bertrand <frederic.bertrand@lecnam.net>

Description Provides browser-native 'WebGL' rendering for R graphics through 'htmlwidgets'. The package supports grammar-style graphics workflows and renderer-ready specifications for dense analytical and scientific scenes, including point, line, trajectory, raster, vector, mesh, and surface layers, shader-driven display modes, timeline controls, structured views, selection metadata, and publication-oriented static export helpers. Rendering stays in the browser, and the core package remains cross-platform without requiring 'CUDA', 'Metal', or 'OpenCL' toolchains.

License GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.1)

Imports ggplot2, htmltools, htmlwidgets, jsonlite, rlang, utils

Suggests boids4R, chromote, knitr, magick, MASS, pkgdown, pkgload, plotly, processx, rmarkdown, shiny, testthat (>= 3.0.0), XGeoRTR

VignetteBuilder knitr

Config/testthat/edition 3

Config/Needs/website pkgdown

URL https://fbertran.github.io/ggWebGL/, https://github.com/fbertran/ggWebGL

BugReports https://github.com/fbertran/ggWebGL/issues

Config/pak/sysreqs cmake make libuv1-dev

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

Date/Publication 2026-06-06 22:08:15 UTC

RemoteUrl https://github.com/fbertran/ggwebgl

RemoteRef HEAD

RemoteSha ca22e9fb466d64cab0ad7f4634094a4d6aba0153

Contents

`animation_spec` *Build Animation Timeline Metadata*

Description

`animation_spec()` is a user-facing alias for `ggwebgl_timeline()`. It keeps one canonical timeline contract while providing a concise name for animation examples and adapter code.

Usage

```
animation_spec(...)
```

Arguments

`...`	Arguments passed to `ggwebgl_timeline()`, such as `frames`, `time`, `values`, `source`, `filter`, `autoplay`, `speed`, `loop`, `controls`, and `fps`.

Value

A ggwebgl_timeline list.

Examples

```
animation_spec(frames = 1:3, autoplay = FALSE)
```

---

as_ggwebgl_spec *Convert backend objects to a ggWebGL renderer specification*

Description

ggWebGL exposes a renderer-adapter protocol for converting explicit backend inputs into normalized primitive scenes. Backend-specific methods must resolve semantics before the widget consumes the payload.

Usage

```
as_ggwebgl_spec(x, ...)
```

Arguments

x	Input object.
...	Passed to method-specific implementations.

Value

A normalized ggWebGL renderer specification.

Examples

```
point_layer <- ggwebgl_layer_points(
  data.frame(x = c(0, 1), y = c(1, 0)),
  x = "x",
  y = "y"
)
spec <- ggwebgl_spec(layers = list(point_layer))

as_ggwebgl_spec(spec)
```

`as_ggwebgl_spec.xgeo_state` *Convert an* `xgeo_state` *object to a ggWebGL renderer specification*

Description

Convert an `xgeo_state` object to a ggWebGL renderer specification

Usage

```
## S3 method for class 'xgeo_state'
as_ggwebgl_spec(
  x,
  embedding = NULL,
  primitive = c("points", "density", "surface"),
  lod = NULL,
  webgl = list(),
  labels = list(),
  point_size = 4,
  alpha = 0.85,
  ...
)
```

Arguments

`x`	An `xgeo_state` object.
`embedding`	Optional embedding name. Defaults to the active embedding.
`primitive`	Primitive family to project to renderer payloads.
`lod`	Optional LOD selector for `primitive = "density"`. Accepts `NULL`, a single bundle name, a `bundle/level` string, or a list with `name` and optional `level`.
`webgl`	Renderer options passed through `normalise_webgl_options()`.
`labels`	Optional labels list (`title`, `subtitle`, `x`, `y`) that overrides metadata-derived defaults.
`point_size`	Point size used for point payloads.
`alpha`	Alpha used for generated payload colors.
`...`	Reserved for future adapters.

Value

A normalized ggWebGL renderer specification.

Examples

```
toy_state <- list(
  attributes = list(
    embeddings = list(
      active = "toy",
      items = list(
        toy = list(
          coords = data.frame(
            point_id = paste0("p", 1:4),
            dim1 = c(0, 1, 0, 1),
            dim2 = c(0, 0, 1, 1)
          )
        )
      )
    ),
    explanations = data.frame(
      point_id = paste0("p", 1:4),
      value = c(0.2, 0.4, 0.8, 0.5)
    )
  ),
  metadata = list(title = "Toy backend state")
)
class(toy_state) <- "xgeo_state"

xgeo_spec <- as_ggwebgl_spec(toy_state, primitive = "points")
xgeo_spec$render$primitives


toy_state <- XGeoRTR::as_xgeo_state(
  data.frame(
    point_id = rep(paste0("p", 1:4), each = 2),
    feature = rep(c("f1", "f2"), times = 4),
    value = c(0.2, 0.7, 0.4, 0.1, 0.8, 0.6, 0.5, 0.3),
    x = c(0, 0, 1, 1, 0, 0, 1, 1),
    y = c(0, 0, 0, 0, 1, 1, 1, 1),
    z = 0
  ),
  x_col = "x",
  y_col = "y",
  z_col = "z",
  value_col = "value",
  feature_col = "feature",
  point_id_col = "point_id"
)

xgeo_spec <- as_ggwebgl_spec(toy_state, primitive = "points")
xgeo_spec$render$primitives
```

---

as_mesh_webgl *Convert Explicit Mesh Data to a ggWebGL Mesh Object*

Description

as_mesh_webgl() converts CRAN-safe explicit mesh inputs into a lightweight helper object accepted by ggwebgl_layer_mesh(). Core ggWebGL does not convert external mesh package classes in this milestone.

Usage

```
as_mesh_webgl(x, ...)
```

Arguments

x	Object to convert. Supported inputs are ggwebgl_mesh objects, lists with vertices and triangles, and data frames containing explicit x, y, z, i, j, and k columns.
...	Additional arguments passed to ggwebgl_mesh().

Value

A ggwebgl_mesh object.

Examples

```
vertices <- data.frame(x = c(0, 1, 0), y = c(0, 0, 1), z = c(0, 0, 0))
triangles <- data.frame(i = 1L, j = 2L, k = 3L)
as_mesh_webgl(list(vertices = vertices, triangles = triangles))
```

---

compose_ggwebgl_figure

Compose a Publication Figure from ggWebGL Panels

Description

Capture one or more ggWebGL scenes and assemble them into a single clean publication image.

Usage

```
compose_ggwebgl_figure(
  panels,
  file,
  width = 1800L,
  height = 1200L,
  format = NULL,
  dpi = 300,
  background = "white",
  layout = c("single", "row", "grid"),
  labels = NULL,
  inset = NULL,
  annotations = NULL,
```

```
  preset = c("clean", "publication"),
  selfcontained = FALSE,
  wait_seconds = 3,
  nrow = NULL,
  ncol = NULL,
  elementId = NULL
)
```

Arguments

panels	A list of panel sources. Each element may be a ggplot, ggWebGL widget, ggwebgl_spec, raw renderer payload, image path, or a list with source plus optional wait_seconds, show_panel_overlay, and preset overrides.
file	Output file path.
width, height	Output size in pixels.
format	Optional image format. When omitted, it is inferred from file.
dpi	Output density metadata used when writing the image.
background	Background colour used for the final flattened image.
layout	One of "single", "row", or "grid".
labels	Optional character vector of panel labels drawn in the top-left corner of each occupied panel cell.
inset	Optional list with a panel source, fractional left, top, width, height, and optional border, border_colour, and border_alpha.
annotations	Optional list of text annotations. Each entry should contain text plus fractional x and y, with optional size, colour, font, hjust, and vjust.
preset	Export preset. "publication" adds subtle panel borders and muted label styling.
selfcontained	Passed through to htmlwidgets::saveWidget() for temporary widget captures.
wait_seconds	Default render delay before capture.
nrow, ncol	Optional grid dimensions used when layout = "grid".
elementId	Optional DOM element id passed when panel sources must first be turned into widgets.

Value

The normalized output file path, invisibly.

Examples

```
old <- options(ggwebgl.reset_processx_supervisor = TRUE)
on.exit(options(old), add = TRUE)

point_spec <- ggwebgl_spec(
  layers = list(
    ggwebgl_layer_points(
```

```
      data.frame(x = c(0.15, 0.48, 0.82), y = c(0.22, 0.76, 0.38)),
      x = "x",
      y = "y",
      colour = c("#0f766e", "#f97316", "#2563eb"),
      alpha = 0.8,
      size = 5
    )
  ),
  webgl = list(shader = "default", interactions = character())
)
line_spec <- ggwebgl_spec(
  layers = list(
    ggwebgl_layer_lines(
      data.frame(x = c(0.1, 0.45, 0.8), y = c(0.25, 0.75, 0.35)),
      x = "x",
      y = "y",
      colour = "#334155",
      alpha = 0.9,
      width = 2
    )
  ),
  webgl = list(shader = "default", interactions = character())
)

out <- tempfile(fileext = ".jpg")
compose_ggwebgl_figure(
  panels = list(
    point_spec,
    line_spec
  ),
  file = out,
  layout = "row",
  labels = c("points", "lines"),
  width = 480,
  height = 240,
  format = "jpeg",
  preset = "clean",
  wait_seconds = 0.25
)
file.exists(out)
```

`coord_webgl_3d` *Request a 3D WebGL Coordinate System*

Description

`coord_webgl_3d()` is a ggplot-addable helper that marks the WebGL payload as `cartesian3d` and installs a structured 3D view contract. It does not replace ggplot2's 2D coordinate system; the standard ggplot object remains valid, and the 3D interpretation is applied by `ggplot_webgl()`. Panel ranges and fixed-scale facet metadata still come from ggplot2's built plot object.

Usage

```
coord_webgl_3d(
  projection = c("perspective", "orthographic"),
  camera = c("orbit", "trackball"),
  depth_test = TRUE,
  state = list()
)
```

Arguments

projection	Projection mode, "perspective" or "orthographic".
camera	3D camera controller, "orbit" or "trackball".
depth_test	Logical scalar; whether the browser renderer should enable depth testing for this scene.
state	Optional camera state list passed to ggwebgl_view().

Value

A ggplot-addable ggwebgl_coord_3d object.

Examples

```
ggplot2::ggplot(
  data.frame(x = 1:3, y = 1:3, z = c(0, 1, 0)),
  ggplot2::aes(x, y, z = z)
) +
  geom_point_webgl() +
  coord_webgl_3d()
```

---

geom_area_webgl *WebGL Area Layer*

Description

Add a filled area layer tagged for the ggWebGL renderer. Stacking and alignment are delegated to ggplot2; the WebGL layer consumes the built ribbon boundaries.

Usage

```
geom_area_webgl(
  mapping = NULL,
  data = NULL,
  stat = "align",
  position = "stack",
  ...,
  orientation = NA,
  outline.type = "upper",
```

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

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used to override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

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

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

`outline.type` Type of the outline of the area; `"both"` draws both the upper and lower lines, `"upper"`/`"lower"` draws the respective lines only. `"full"` draws a closed polygon around the area.

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

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

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

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

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

Value

A `Layer` ready for `ggplot2`.

Examples

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

```
ggplot2::ggplot(area_data, ggplot2::aes(x, y)) +
  geom_area_webgl(fill = "#0f766e", alpha = 0.7)
```

`geom_bar_webgl` *WebGL Bar Layer*

Description

Add a bar layer tagged for the ggWebGL renderer. Counts and rectangle boundaries are produced by ggplot2 through the selected stat and position.

Usage

```
geom_bar_webgl(
  mapping = NULL,
  data = NULL,
  stat = "count",
  position = "stack",
  ...,
  just = 0.5,
  lineend = "butt",
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created. A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`stat`	Statistical transformation to use. Defaults to `"count"` so ggplot2 computes bar heights before WebGL serialization.
`position`	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

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

`...` Additional arguments forwarded through `ggplot2::layer()`, including `ggplot2::stat_count()` parameters such as `width`, plus static aesthetics.

`just` Adjustment for column placement. Set to `0.5` by default, meaning that columns will be centered about axis breaks. Set to `0` or `1` to place columns to the left/right of axis breaks. Note that this argument may have unintended behaviour when used with alternative positions, e.g. `position_dodge()`.

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

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

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

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

Value

A `Layer` ready for `ggplot2`.

Examples

```
bar_data <- data.frame(group = c("a", "a", "b", "c", "c", "c"))

ggplot2::ggplot(bar_data, ggplot2::aes(group)) +
  geom_bar_webgl(fill = "#2563eb")
```

`geom_bin2d_webgl` *WebGL 2D Binned Rectangles*

Description

Add a 2D binned layer tagged for the `ggWebGL` renderer. Binning is delegated to `ggplot2::StatBin2d`; the WebGL layer consumes the built rectangle boundaries and count/density metadata.

Usage

```
geom_bin2d_webgl(
  mapping = NULL,
  data = NULL,
  stat = "bin2d",
  position = "identity",
  ...,
  lineend = "butt",
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	Statistical transformation to use. Defaults to "bin2d" so ggplot2 computes two-dimensional bins before WebGL serialization.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Additional arguments forwarded through ggplot2::layer(), including ggplot2::stat_bin_2d() parameters such as bins, binwidth, breaks, and drop, plus static aesthetics.
lineend	Line end style (round, butt, square).
linejoin	Line join style (round, mitre, bevel).
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

Value

A Layer ready for ggplot2.

Examples

```
bin_data <- data.frame(
  x = c(0.1, 0.2, 0.7, 1.2, 1.8, 2.1),
  y = c(0.1, 0.5, 0.6, 1.1, 1.3, 1.8)
)

ggplot2::ggplot(bin_data, ggplot2::aes(x, y)) +
  geom_bin2d_webgl(binwidth = c(1, 1))
```

---

geom_boxplot_webgl *WebGL Boxplot Layer*

Description

Add a boxplot layer tagged for the ggWebGL renderer. Boxplot statistics are computed by ggplot2; the renderer serializes the built box body as rectangles, medians/whiskers as pure segments, and outliers as points.

Usage

```
geom_boxplot_webgl(
  mapping = NULL,
  data = NULL,
  stat = "boxplot",
  position = "dodge2",
  ...,
  outliers = TRUE,
  outlier.colour = NULL,
  outlier.color = NULL,
  outlier.fill = NULL,
  outlier.shape = NULL,
  outlier.size = NULL,
  outlier.stroke = 0.5,
  outlier.alpha = NULL,
```

```
  whisker.colour = NULL,
  whisker.color = NULL,
  whisker.linetype = NULL,
  whisker.linewidth = NULL,
  staple.colour = NULL,
  staple.color = NULL,
  staple.linetype = NULL,
  staple.linewidth = NULL,
  median.colour = NULL,
  median.color = NULL,
  median.linetype = NULL,
  median.linewidth = NULL,
  box.colour = NULL,
  box.color = NULL,
  box.linetype = NULL,
  box.linewidth = NULL,
  notch = FALSE,
  notchwidth = 0.5,
  staplewidth = 0,
  varwidth = FALSE,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	Statistical transformation to use. Defaults to "boxplot" so ggplot2 computes boxplot summary statistics before WebGL serialization.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.

- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Additional arguments forwarded through `ggplot2::layer()`, including `ggplot2::stat_boxplot()` parameters such as `coef` and `width`, plus static aesthetics.

`outliers` Whether to display (`TRUE`) or discard (`FALSE`) outliers from the plot. Hiding or discarding outliers can be useful when, for example, raw data points need to be displayed on top of the boxplot. By discarding outliers, the axis limits will adapt to the box and whiskers only, not the full data range. If outliers need to be hidden and the axes needs to show the full data range, please use `outlier.shape = NA` instead.

`outlier.colour`, `outlier.color`, `outlier.fill`, `outlier.shape`, `outlier.size`, `outlier.stroke`, `outlier.alpha`
Default aesthetics for outliers. Set to `NULL` to inherit from the data's aesthetics.

`whisker.colour`, `whisker.color`, `whisker.linetype`, `whisker.linewidth`
Default aesthetics for the whiskers. Set to `NULL` to inherit from the data's aesthetics.

`staple.colour`, `staple.color`, `staple.linetype`, `staple.linewidth`
Default aesthetics for the staples. Set to `NULL` to inherit from the data's aesthetics. Note that staples don't appear unless the `staplewidth` argument is set to a non-zero size.

`median.colour`, `median.color`, `median.linetype`, `median.linewidth`
Default aesthetics for the median line. Set to `NULL` to inherit from the data's aesthetics.

`box.colour`, `box.color`, `box.linetype`, `box.linewidth`
Default aesthetics for the boxes. Set to `NULL` to inherit from the data's aesthetics.

`notch` If `FALSE` (default) make a standard box plot. If `TRUE`, make a notched box plot. Notches are used to compare groups; if the notches of two boxes do not overlap, this suggests that the medians are significantly different.

`notchwidth` For a notched box plot, width of the notch relative to the body (defaults to `notchwidth = 0.5`).

`staplewidth` The relative width of staples to the width of the box. Staples mark the ends of the whiskers with a line.

`varwidth` If `FALSE` (default) make a standard box plot. If `TRUE`, boxes are drawn with widths proportional to the square-roots of the number of observations in the groups (possibly weighted, using the `weight` aesthetic).

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

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

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes — If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

Value

A Layer ready for ggplot2.

Examples

```
box_data <- data.frame(group = rep(c("a", "b"), each = 6), value = c(1:6, 2:7))
ggplot2::ggplot(box_data, ggplot2::aes(group, value, fill = group)) +
  geom_boxplot_webgl()
```

---

geom_contour_webgl *WebGL Contour Line Layer*

Description

Add a line-contour layer tagged for the ggWebGL renderer. Contour generation is delegated to ggplot2::StatContour; this layer serializes the built contour paths as grouped WebGL line primitives. Filled contours are not rendered by this layer.

Usage

```
geom_contour_webgl(
  mapping = NULL,
  data = NULL,
  stat = "contour",
  position = "identity",
  ...,
  bins = NULL,
  binwidth = NULL,
  breaks = NULL,
  arrow = NULL,
  arrow.fill = NULL,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used to override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the geom part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`bins` Number of contour bins. Overridden by `breaks`.

`binwidth` The width of the contour bins. Overridden by `bins`.

`breaks` One of:

- Numeric vector to set the contour breaks
- A function that takes the range of the data and binwidth as input and returns breaks as output. A function can be created from a formula (e.g. ~ fullseq(.x, .y)).

Overrides `binwidth` and `bins`. By default, this is a vector of length ten with `pretty()` breaks.

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

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

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

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

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

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

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

Value

A `Layer` ready for `ggplot2`.

Examples

```
grid <- expand.grid(x = seq(-1, 1, length.out = 5), y = seq(-1, 1, length.out = 5))
grid$z <- with(grid, x^2 - y^2)

ggplot2::ggplot(grid, ggplot2::aes(x, y, z = z)) +
  geom_contour_webgl(bins = 4)
```

`geom_crossbar_webgl` *WebGL Crossbar Layer*

Description

Add a crossbar layer tagged for the ggWebGL renderer. Crossbars serialize to one filled rectangle payload for the body and one pure segment payload for the middle line.

Usage

```
geom_crossbar_webgl(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  middle.colour = NULL,
  middle.color = NULL,
  middle.linetype = NULL,
  middle.linewidth = NULL,
  box.colour = NULL,
  box.color = NULL,
  box.linetype = NULL,
  box.linewidth = NULL,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

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

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

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

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

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

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

`...` 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.

middle.colour, middle.color, middle.linetype, middle.linewidth
: Default aesthetics for the middle line. Set to NULL to inherit from the data's aesthetics.

box.colour, box.color, box.linetype, box.linewidth
: Default aesthetics for the boxes. Set to NULL to inherit from the data's aesthetics.

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

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

show.legend
: logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes
: If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

Value

A Layer ready for ggplot2.

Examples

```
crossbars <- data.frame(x = 1:3, y = c(2, 3, 2.5), ymin = c(1, 2, 1.8), ymax = c(3, 4, 3.2))
ggplot2::ggplot(crossbars, ggplot2::aes(x, y, ymin = ymin, ymax = ymax)) +
  geom_crossbar_webgl(width = 0.35, fill = "#93c5fd")
```

---

geom_density_webgl *WebGL Density Curve Layer*

Description

Add a density curve layer tagged for the ggWebGL renderer. Density estimation is delegated to ggplot2::StatDensity; this geom serializes the resulting curve as a line path. Filled densities are not rendered by this layer.

Usage

```
geom_density_webgl(
  mapping = NULL,
  data = NULL,
  stat = "density",
  position = "identity",
  ...,
  outline.type = "upper",
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	Statistical transformation to use. Defaults to "density" so ggplot2 computes density curves before WebGL serialization.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Additional arguments forwarded through ggplot2::layer(), including ggplot2::stat_density() parameters such as adjust, kernel, n, trim, bounds, and bw, plus static aesthetics.

outline.type	Type of the outline of the area; "both" draws both the upper and lower lines, "upper"/"lower" draws the respective lines only. "full" draws a closed polygon around the area.
lineend	Line end style (round, butt, square).
linejoin	Line join style (round, mitre, bevel).
linemitre	Line mitre limit (number greater than 1).
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

Value

A Layer ready for ggplot2.

Examples

```
values <- data.frame(x = seq(-2, 2, length.out = 40))

ggplot2::ggplot(values, ggplot2::aes(x)) +
  geom_density_webgl(colour = "#0f766e")
```

---

geom_density2d_webgl *WebGL 2D Density Contour Layer*

Description

Add a line-contour density layer tagged for the ggWebGL renderer. The 2D density estimate and contour lines are computed by ggplot2; this layer serializes the built contour paths as grouped WebGL line primitives. Filled density contours are not rendered by this layer.

Usage

```
geom_density2d_webgl(
  mapping = NULL,
  data = NULL,
  stat = "density_2d",
  position = "identity",
  ...,
  contour_var = "density",
```

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

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	Statistical transformation to use. Defaults to "density_2d" so ggplot2 computes two-dimensional density contours before WebGL serialization. This ggplot2 statistic uses MASS, which is listed in Suggests and guarded in examples/tests.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Additional arguments forwarded through ggplot2::layer(), including ggplot2::stat_density_2d() parameters such as h, n, bins, binwidth, and breaks, plus static aesthetics.
contour_var	Character string identifying the variable to contour by. Can be one of "density", "ndensity", or "count". See the section on computed variables for details.
lineend	Line end style (round, butt, square).
linejoin	Line join style (round, mitre, bevel).
linemitre	Line mitre limit (number greater than 1).
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes — If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

Value

A Layer ready for ggplot2.

Examples

```
density_points <- expand.grid(x = seq(-1, 1, length.out = 6), y = seq(-1, 1, length.out = 6))

if (requireNamespace("MASS", quietly = TRUE)) {
  ggplot2::ggplot(density_points, ggplot2::aes(x, y)) +
    geom_density2d_webgl(bins = 3)
}
```

---

geom_errorbar_webgl *WebGL Errorbar Layer*

Description

Add a vertical error-bar layer tagged for the ggWebGL renderer. The renderer consumes ggplot2-built range and cap columns and serializes them as pure segment primitives.

Usage

```
geom_errorbar_webgl(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  orientation = NA,
  lineend = "butt",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used to override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the geom part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

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

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

`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 `Layer` ready for `ggplot2`.

Examples

```
errors <- data.frame(x = 1:3, y = c(2, 3, 2.5), ymin = c(1, 2, 1.8), ymax = c(3, 4, 3.2))
ggplot2::ggplot(errors, ggplot2::aes(x, y, ymin = ymin, ymax = ymax)) +
  geom_errorbar_webgl(width = 0.2)
```

`geom_freqpoly_webgl` *WebGL Frequency Polygon Layer*

Description

Add a frequency polygon layer tagged for the `ggWebGL` renderer. Binning is delegated to `ggplot2::StatBin`; the WebGL layer consumes the built path coordinates.

Usage

```
geom_freqpoly_webgl(
  mapping = NULL,
  data = NULL,
  stat = "bin",
  position = "identity",
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	Statistical transformation to use. Defaults to "bin" so ggplot2 computes frequency polygon bins before WebGL serialization.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Additional arguments forwarded through ggplot2::layer(), including ggplot2::stat_bin() parameters such as bins, binwidth, breaks, boundary, center, closed, and pad, plus static aesthetics.
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 Layer ready for ggplot2.

Examples

```
values <- data.frame(x = c(0.1, 0.2, 0.7, 1.2, 1.6, 2.1))

ggplot2::ggplot(values, ggplot2::aes(x)) +
  geom_freqpoly_webgl(binwidth = 0.5, colour = "#2563eb")
```

---

geom_histogram_webgl *WebGL Histogram Layer*

Description

Add a histogram layer tagged for the ggWebGL renderer. Binning is delegated to ggplot2::StatBin; the WebGL layer consumes the built bar rectangles.

Usage

```
geom_histogram_webgl(
  mapping = NULL,
  data = NULL,
  stat = "bin",
  position = "stack",
  ...,
  binwidth = NULL,
  bins = NULL,
  orientation = NA,
  lineend = "butt",
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	Statistical transformation to use. Defaults to "bin" so ggplot2 computes histogram bins before WebGL serialization.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Additional arguments forwarded through ggplot2::layer(), including ggplot2::stat_bin() parameters such as breaks, boundary, center, closed, and pad, plus static aesthetics.
binwidth	The width of the bins. Can be specified as a numeric value or as a function that takes x after scale transformation as input and returns a single numeric value. When specifying a function along with a grouping structure, the function will be called once per group. The default is to use the number of bins in bins, covering the range of the data. You should always override this value, exploring multiple widths to find the best to illustrate the stories in your data. The bin width of a date variable is the number of days in each time; the bin width of a time variable is the number of seconds.
bins	Number of bins. Overridden by binwidth. Defaults to 30.
orientation	The orientation of the layer. The default (NA) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting orientation to either "x" or "y". See the *Orientation* section for more detail.
lineend	Line end style (round, butt, square).
linejoin	Line join style (round, mitre, bevel).

na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

Value

A Layer ready for ggplot2.

Examples

```
hist_data <- data.frame(x = c(0.1, 0.2, 0.7, 1.2, 1.6, 2.1))

ggplot2::ggplot(hist_data, ggplot2::aes(x)) +
  geom_histogram_webgl(binwidth = 0.5, fill = "#0f766e")
```

---

geom_label_webgl *WebGL Label Overlay Layer*

Description

Add label annotations tagged for the ggWebGL renderer. Labels are serialized as text overlay metadata with background-box styling metadata; they are not drawn as WebGL glyphs.

Usage

```
geom_label_webgl(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "nudge",
  ...,
  parse = FALSE,
  label.padding = grid::unit(0.25, "lines"),
  label.r = grid::unit(0.15, "lines"),
  border.colour = NULL,
  border.color = NULL,
  text.colour = NULL,
  text.color = NULL,
  size.unit = "mm",
  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.

`parse` If `TRUE`, the labels will be parsed into expressions and displayed as described in `?plotmath`.

`label.padding` Amount of padding around label. Defaults to 0.25 lines.

`label.r` Radius of rounded corners. Defaults to 0.15 lines.

`border.colour, border.color`
Colour of label border. When `NULL` (default), the `colour` aesthetic determines the colour of the label border. `border.color` is an alias for `border.colour`.

`text.colour, text.color`
Colour of the text. When `NULL` (default), the `colour` aesthetic determines the colour of the text. `text.color` is an alias for `text.colour`.

`size.unit` How the `size` aesthetic is interpreted: as millimetres (`"mm"`, default), points (`"pt"`), centimetres (`"cm"`), inches (`"in"`), or picas (`"pc"`).

`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 `Layer` ready for `ggplot2`.

Examples

```
labels <- data.frame(x = c(1, 2), y = c(2, 1), label = c("A", "B"))
ggplot2::ggplot(labels, ggplot2::aes(x, y, label = label)) +
  geom_point_webgl() +
  geom_label_webgl(fill = "#f8fafc", alpha = 0.9)
```

`geom_line_webgl` *WebGL Line Layer*

Description

Add a line layer that is tagged for the ggWebGL rendering pipeline. The layer is drawn through the browser WebGL renderer when passed to `ggplot_webgl()`.

Usage

```
geom_line_webgl(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

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

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

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

... — 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.

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

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

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes — If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

Value

A Layer ready for ggplot2.

Examples

```
line_data <- data.frame(
  x = c(1, 2, 3, 1, 2, 3),
  y = c(1, 2, 1, 2, 3, 2),
  group = c("a", "a", "a", "b", "b", "b")
)

line_plot <- ggplot2::ggplot(
  line_data,
  ggplot2::aes(x, y, group = group, colour = group)
) +
  geom_line_webgl(linewidth = 1) +
  theme_webgl(shader = "trajectory_age")

line_plot
```

---

geom_linerange_webgl *WebGL Linerange Layer*

Description

Add a vertical range layer tagged for the ggWebGL renderer. The renderer consumes ggplot2-built x, ymin, and ymax values and serializes ranges as pure segment primitives.

Usage

```
geom_linerange_webgl(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  orientation = NA,
  lineend = "butt",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`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 `Layer` ready for `ggplot2`.

Examples

```
ranges <- data.frame(x = 1:3, y = c(2, 3, 2.5), ymin = c(1, 2, 1.8), ymax = c(3, 4, 3.2))
ggplot2::ggplot(ranges, ggplot2::aes(x, y, ymin = ymin, ymax = ymax)) +
  geom_linerange_webgl(linewidth = 1)
```

---

geom_mesh_webgl *WebGL Unstructured Mesh Layer*

Description

Add an unstructured triangle mesh layer tagged for the `ggWebGL` 3D renderer. Mesh triangles are supplied with `i`, `j`, and `k` aesthetics using one-based vertex indices.

Usage

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

```
  ...,
  shading = c("mesh_lambert", "mesh_flat", "mesh_phong_simple", "mesh_scalar_colormap",
    "mesh_selection_highlight"),
  wireframe = FALSE,
  material = NULL,
  normals = NULL,
  pick_id = 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.

`...` 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.

`shading` Mesh shader mode. One of `"mesh_lambert"`, `"mesh_flat"`, `"mesh_phong_simple"`, `"mesh_scalar_colormap"`, or `"mesh_selection_highlight"`.

`wireframe` Whether to request a wireframe overlay.

`material` Mesh material created by `ggwebgl_material()`.

`normals` Optional vertex normals or `"auto"`.

`pick_id` Optional face picking ids.

`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 `Layer` ready for `ggplot2`.

Examples

```
vertices <- data.frame(
```

```
  x = c(0, 1, 0),
  y = c(0, 0, 1),
  z = c(0, 0, 0),
  i = c(1, NA, NA),
  j = c(2, NA, NA),
  k = c(3, NA, NA)
)
ggplot2::ggplot(vertices, ggplot2::aes(x, y, z = z, i = i, j = j, k = k)) +
  geom_mesh_webgl()
```

`geom_path_webgl` *WebGL Ordered Path Layer*

Description

Add an ordered two-dimensional path layer tagged for the `ggWebGL` rendering pipeline. Unlike `geom_line_webgl()`, this geom is based on `ggplot2::GeomPath` and preserves row order within each group.

Usage

```
geom_path_webgl(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  arrow = 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.

... — 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.

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

linejoin	Line join style (round, mitre, bevel).
linemitre	Line mitre limit (number greater than 1).
arrow	Arrow specification, as created by grid::arrow().
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 Layer ready for ggplot2.

Examples

```
path_data <- data.frame(
  x = c(3, 1, 2, 4),
  y = c(0.2, 0.8, 0.4, 0.6),
  frame = 1:4
)

path_plot <- ggplot2::ggplot(
  path_data,
  ggplot2::aes(x, y, frame = frame)
) +
  geom_path_webgl(linewidth = 1.2) +
  theme_webgl(shader = "trajectory_age")

path_plot
```

---

geom_path3d_webgl *WebGL Ordered 3D Path Layer*

Description

Add an ordered three-dimensional path layer that is tagged for the ggWebGL rendering pipeline. Unlike geom_line_webgl(), this geom is based on ggplot2::GeomPath and preserves row order within each group.

Usage

```
geom_path3d_webgl(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  arrow = 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.

`...` 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.

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

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

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

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

`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 `Layer` ready for `ggplot2`.

Examples

```
path_data <- data.frame(
  x = c(0, 0.4, 0.2, 0.8),
  y = c(0, 0.3, 0.7, 1),
  z = c(0, 0.2, 0.5, 0.9),
  frame = 1:4
)

path_plot <- ggplot2::ggplot(
  path_data,
  ggplot2::aes(x, y, z = z, frame = frame)
) +
  geom_path3d_webgl(linewidth = 1.2) +
  theme_webgl(shader = "trajectory_age")

path_plot
```

---

geom_point_webgl *WebGL Point Layer*

Description

Add a point layer that is tagged for the ggWebGL rendering pipeline. The layer is drawn through the browser WebGL renderer when passed to `ggplot_webgl()`.

Usage

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

Arguments

mapping	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

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

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

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

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

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

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

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

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

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

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 Layer ready for ggplot2.

Examples

```
point_plot <- ggplot2::ggplot(
  mtcars[1:8, ],
  ggplot2::aes(mpg, wt, colour = factor(cyl))
) +
  geom_point_webgl(size = 2) +
  theme_webgl()

point_plot
```

---

geom_pointrange_webgl *WebGL Pointrange Layer*

Description

Add a point plus vertical range layer tagged for the ggWebGL renderer. Pointranges serialize to one point payload and one pure segment payload.

Usage

```
geom_pointrange_webgl(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  orientation = NA,
  lineend = "butt",
```

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

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used to override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the

available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the geom part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the stat part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The key_glyph argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

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

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

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 Layer ready for ggplot2.

Examples

```
ranges <- data.frame(x = 1:3, y = c(2, 3, 2.5), ymin = c(1, 2, 1.8), ymax = c(3, 4, 3.2))
ggplot2::ggplot(ranges, ggplot2::aes(x, y, ymin = ymin, ymax = ymax)) +
  geom_pointrange_webgl()
```

geom_polygon_webgl *WebGL Simple Polygon Layer*

Description

Add a simple polygon layer tagged for the ggWebGL renderer. The renderer triangulates each ggplot2-built group as one simple, non-self-intersecting ring and sends the result through the existing mesh primitive. Holes, multipolygons, and self-intersections are not supported.

Usage

```
geom_polygon_webgl(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  rule = "evenodd",
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

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

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

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

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

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

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

`rule` Either `"evenodd"` or `"winding"`. If polygons with holes are being drawn (using the `subgroup` aesthetic) this argument defines how the hole coordinates are interpreted. See the examples in `grid::pathGrob()` for an explanation.

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

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

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

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

show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

Value

A Layer ready for ggplot2.

Examples

```
polygon_data <- data.frame(
  x = c(0, 1, 0.8, 0.2),
  y = c(0, 0.1, 1, 0.8)
)

ggplot2::ggplot(polygon_data, ggplot2::aes(x, y)) +
  geom_polygon_webgl(fill = "#38bdf8", alpha = 0.7)
```

---

geom_raster_webgl *WebGL Raster Layer*

Description

Add a raster layer that is tagged for the ggWebGL rendering pipeline. The layer is serialized into a texture-backed raster payload when passed to ggplot_webgl().

Usage

```
geom_raster_webgl(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  hjust = 0.5,
  vjust = 0.5,
  interpolate = FALSE,
  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.

`hjust, vjust` horizontal and vertical justification of the grob. Each justification value should be a number between 0 and 1. Defaults to 0.5 for both, centering each pixel over its data location.

`interpolate` If `TRUE` interpolate linearly, if `FALSE` (the default) don't interpolate.

`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 `Layer` ready for `ggplot2`.

Examples

```
raster_data <- expand.grid(x = 1:3, y = 1:2)
raster_data$z <- with(raster_data, x + y)

raster_plot <- ggplot2::ggplot(
  raster_data,
  ggplot2::aes(x, y, fill = z)
) +
  geom_raster_webgl(interpolate = TRUE) +
  ggplot2::scale_fill_gradient(low = "#0f172a", high = "#38bdf8") +
  theme_webgl()

raster_plot
```

geom_rect_webgl *WebGL Rectangle Layer*

Description

Add a rectangle layer tagged for the ggWebGL renderer. Boundaries are taken from the data built by ggplot2, so xmin, xmax, ymin, and ymax follow the same setup rules as ggplot2::geom_rect().

Usage

```
geom_rect_webgl(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  lineend = "butt",
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

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

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

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

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

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

Value

A Layer ready for ggplot2.

Examples

```
rects <- data.frame(
  xmin = c(0, 1.2),
  xmax = c(0.8, 2),
  ymin = c(0, 0.4),
  ymax = c(1, 1.4),
  group = c("a", "b")
)

ggplot2::ggplot(
  rects,
  ggplot2::aes(xmin = xmin, xmax = xmax, ymin = ymin, ymax = ymax, fill = group)
) +
  geom_rect_webgl(alpha = 0.7)
```

---

geom_ribbon_webgl *WebGL Ribbon Layer*

Description

Add a filled ribbon layer tagged for the ggWebGL renderer. The renderer consumes ggplot2-built x, ymin, and ymax values and draws each group/run as a filled triangle strip.

Usage

```
geom_ribbon_webgl(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  orientation = NA,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  outline.type = "both",
  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.

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

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

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

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

`outline.type` Type of the outline of the area; `"both"` draws both the upper and lower lines, `"upper"`/`"lower"` draws the respective lines only. `"full"` draws a closed polygon around the area.

`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 `Layer` ready for `ggplot2`.

Examples

```
ribbon_data <- data.frame(
  x = 1:4,
  ymin = c(0.1, 0.2, 0.1, 0.3),
  ymax = c(0.4, 0.7, 0.5, 0.8)
)

ggplot2::ggplot(ribbon_data, ggplot2::aes(x, ymin = ymin, ymax = ymax)) +
  geom_ribbon_webgl(fill = "#38bdf8", alpha = 0.6)
```

geom_rug_webgl *WebGL Rug Layer*

Description

Add rug marks tagged for the ggWebGL renderer. Rug marks are serialized as pure segment primitives with arrowheads disabled.

Usage

```
geom_rug_webgl(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  lineend = "butt",
  sides = "bl",
  outside = FALSE,
  length = grid::unit(0.03, "npc"),
  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.

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

`sides` A string that controls which sides of the plot the rugs appear on. It can be set to a string containing any of `"trbl"`, for top, right, bottom, and left.

`outside` logical that controls whether to move the rug tassels outside of the plot area. Default is off (FALSE). You will also need to use `coord_cartesian(clip = "off")`. When set to TRUE, also consider changing the sides argument to "tr". See examples.

`length` A `grid::unit()` object that sets the length of the rug lines. Use scale expansion to avoid overplotting of data.

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 Layer ready for ggplot2.

Examples

```
rug_data <- data.frame(x = c(1, 2, 3), y = c(2, 1, 3))
ggplot2::ggplot(rug_data, ggplot2::aes(x, y)) +
  geom_point_webgl() +
  geom_rug_webgl(sides = "bl")
```

---

geom_segment_webgl *WebGL Segment Layer*

Description

Add a pure line-segment layer tagged for the ggWebGL renderer. Segments use the vector primitive with arrowheads disabled; use geom_vector_webgl() when arrowheads are wanted.

Usage

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

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used to override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the geom part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`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 `Layer` ready for `ggplot2`.

Examples

```
segments <- data.frame(
  x = c(0, 1),
  y = c(0, 0.2),
  xend = c(0.8, 1.6),
  yend = c(0.7, 1)
)
ggplot2::ggplot(segments, ggplot2::aes(x, y, xend = xend, yend = yend)) +
  geom_segment_webgl(linewidth = 1.2)
```

---

geom_surface_webgl *WebGL Structured Grid Surface Layer*

Description

Add a rectilinear grid surface layer tagged for the `ggWebGL` structured surface renderer. Surface layers are sent to WebGL as indexed triangles with per-vertex positions, normals, colours, and surface metadata.

Usage

```
geom_surface_webgl(
  mapping = NULL,
  data = NULL,
  stat = StatSurfaceWebGL,
  position = "identity",
  ...,
  shading = c("surface_lambert", "surface_flat", "surface_height_colormap",
    "surface_uncertainty_alpha"),
  wireframe = FALSE,
  material = NULL,
  normals = "auto",
  contours = FALSE,
  contour_levels = NULL,
  contour_colour = "#1f2937",
  contour_width = 1,
  pick_id = 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.

`...` 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.

`shading` Surface shader mode. One of `"surface_lambert"`, `"surface_flat"`, `"surface_height_colormap"`, or `"surface_uncertainty_alpha"`.

`wireframe` Whether to draw a wireframe overlay.

`material` Surface material metadata created by `ggwebgl_material()`.

`normals` Normal-generation mode. `"auto"` computes vertex normals.

`contours` Whether to generate contour-line overlays on the R side.

`contour_levels` Optional numeric contour levels. Defaults to `pretty()` levels across the surface z range when `contours = TRUE`.

`contour_colour` Contour line colour.

`contour_width` Contour line width in renderer pixels.

`pick_id` Optional triangle picking ids.

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 Layer ready for ggplot2.

Examples

```
surface <- expand.grid(x = 1:3, y = 1:3)
surface$z <- with(surface, sin(x) + cos(y))
ggplot2::ggplot(surface, ggplot2::aes(x, y, z = z, fill = z)) +
  geom_surface_webgl()
```

---

geom_text_webgl *WebGL Text Overlay Layer*

Description

Add static text labels tagged for the ggWebGL renderer. Text is serialized as overlay metadata; it is not drawn as WebGL glyphs.

Usage

```
geom_text_webgl(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "nudge",
  ...,
  parse = FALSE,
  check_overlap = FALSE,
  size.unit = "mm",
  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.

`parse` If `TRUE`, the labels will be parsed into expressions and displayed as described in `?plotmath`.

`check_overlap` If `TRUE`, text that overlaps previous text in the same layer will not be plotted. `check_overlap` happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling `geom_text()`. Note that this argument is not supported by `geom_label()`.

`size.unit` How the `size` aesthetic is interpreted: as millimetres (`"mm"`, default), points (`"pt"`), centimetres (`"cm"`), inches (`"in"`), or picas (`"pc"`).

`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 `Layer` ready for `ggplot2`.

Examples

```
labels <- data.frame(x = c(1, 2), y = c(2, 1), label = c("left", "right"))
ggplot2::ggplot(labels, ggplot2::aes(x, y, label = label)) +
  geom_point_webgl() +
  geom_text_webgl(vjust = -0.6)
```

geom_tile_webgl *WebGL Tile Layer*

Description

Add a tile layer tagged for the ggWebGL renderer. Tile boundaries are read from ggplot2's built layer data, which preserves geom_tile() width, height, and irregular-spacing behavior.

Usage

```
geom_tile_webgl(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  lineend = "butt",
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

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

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

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

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

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

Value

A Layer ready for ggplot2.

Examples

```
tiles <- expand.grid(x = 1:3, y = 1:2)
tiles$value <- with(tiles, x + y)

ggplot2::ggplot(tiles, ggplot2::aes(x, y, fill = value)) +
  geom_tile_webgl(alpha = 0.85)
```

---

geom_vector_webgl *WebGL Vector Arrow Layer*

Description

Add a 2D or 3D vector-arrow layer tagged for the ggWebGL renderer.

Usage

```
geom_vector_webgl(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  head_size = 9,
  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.

`head_size` Arrowhead size in renderer pixels.

`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 Layer ready for ggplot2.

Examples

```
arrows <- data.frame(x = 1:3, y = 1:3, z = 0, xend = 1:3 + 0.3, yend = 1:3 + 0.2, zend = 0.2)
ggplot2::ggplot(arrows, ggplot2::aes(x, y, z = z, xend = xend, yend = yend, zend = zend)) +
  geom_vector_webgl()
```

---

geom_violin_webgl *WebGL Violin Layer*

Description

Add a violin layer tagged for the ggWebGL renderer. Density estimation and scaling are computed by ggplot2::stat_ydensity; the renderer converts each built violin group to a mesh-backed filled strip. Quantile guide lines and exact stroke parity with ggplot2::geom_violin() are not implemented.

Usage

```
geom_violin_webgl(
  mapping = NULL,
  data = NULL,
  stat = "ydensity",
  position = "dodge",
  ...,
  trim = TRUE,
  bounds = c(-Inf, Inf),
  scale = "area",
  na.rm = FALSE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	Statistical transformation to use. Defaults to "ydensity" so ggplot2 computes violin density and scaling before WebGL serialization.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Additional arguments forwarded through ggplot2::layer(), including ggplot2::stat_ydensity() parameters such as bw, adjust, kernel, n, and quantile-related arguments, plus static aesthetics.
trim	If TRUE (default), trim the tails of the violins to the range of the data. If FALSE, don't trim the tails.
bounds	Known lower and upper bounds for estimated data. Default c(-Inf, Inf) means that there are no (finite) bounds. If any bound is finite, boundary effect of default density estimation will be corrected by reflecting tails outside bounds around their closest edge. Data points outside of bounds are removed with a warning.
scale	if "area" (default), all violins have the same area (before trimming the tails). If "count", areas are scaled proportionally to the number of observations. If "width", all violins have the same maximum width.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
orientation	The orientation of the layer. The default (NA) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting orientation to either "x" or "y". See the *Orientation* section for more detail.

show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders().

Value

A Layer ready for ggplot2.

Examples

```
violin_data <- data.frame(
  group = rep(c("a", "b"), each = 24),
  value = c(seq(-1, 1, length.out = 24), seq(-0.4, 1.6, length.out = 24))
)

ggplot2::ggplot(violin_data, ggplot2::aes(group, value, fill = group)) +
  geom_violin_webgl(alpha = 0.65)
```

---

ggplot_webgl *Convert a ggplot to a ggWebGL Widget*

Description

Build a widget payload from a ggplot object. The current implementation renders supported point, line, raster, and fixed-scale facet layouts through the browser WebGL path while keeping unsupported layers explicit in the payload.

Usage

```
ggplot_webgl(plot, width = NULL, height = NULL, elementId = NULL)
```

Arguments

plot	A ggplot object.
width, height	Optional widget dimensions.
elementId	Optional DOM element id.

Value

An htmlwidget.

Examples

```
plot <- ggplot2::ggplot(
  mtcars[1:10, ],
  ggplot2::aes(mpg, wt, colour = factor(cyl))
) +
  geom_point_webgl(size = 2) +
  theme_webgl(shader = "default")

ggplot_webgl(plot, width = 420, height = 320)
```

---

ggWebGL *Create a ggWebGL htmlwidget*

Description

Low-level constructor for the package widget binding.

Usage

```
ggWebGL(x = list(), width = NULL, height = NULL, elementId = NULL)
```

Arguments

`x`	Named list describing a widget payload.
`width, height`	Optional widget dimensions passed through to `htmlwidgets::createWidget()`.
`elementId`	Optional DOM element id.

Value

An `htmlwidget`.

Examples

```
point_layer <- ggwebgl_layer_points(
  data.frame(x = c(0, 1, 2), y = c(2, 1, 0)),
  x = "x",
  y = "y"
)
spec <- ggwebgl_spec(layers = list(point_layer))

ggWebGL(spec, width = 320, height = 240)
```

ggwebgl_example_data *Load Packaged ggWebGL Example Data*

Description

Read one of the packaged real-data subsets used by the examples, vignettes, and benchmark scripts.

Usage

```
ggwebgl_example_data(
  name = c("volcano_dem", "storm_tracks", "dense_embedding", "diamonds_embedding")
)
```

Arguments

name	Name of the example dataset to load. "dense_embedding" is the stable public alias for the packaged dense point-cloud dataset. The legacy alias "diamonds_embedding" is kept for backward compatibility.

Value

A data frame for CSV-backed datasets or the saved R object for volcano_dem.

Examples

```
dem <- ggwebgl_example_data("volcano_dem")
str(dem)
```

ggwebgl_interactions *Define ggWebGL Runtime Interactions*

Description

Build a structured interaction specification for widget-owned hover, click, brush/lasso, camera, and Shiny event behavior. Existing character interactions vectors remain supported; this helper is the normalized contract for new code.

Usage

```
ggwebgl_interactions(
  hover = TRUE,
  click = TRUE,
  brush = FALSE,
  lasso = FALSE,
  camera = TRUE,
  shiny = TRUE
)
```

Arguments

hover	Enable hover picking and tooltip/event emission.
click	Enable click picking and event emission.
brush	Enable rectangular brush selection.
lasso	Enable lasso selection.
camera	Enable camera-state event emission for 3D interactions.
shiny	Enable Shiny event emission when a Shiny runtime is present.

Value

A ggwebgl_interactions list.

Examples

```
ggwebgl_interactions(brush = TRUE)
```

---

ggwebgl_layer_lines *Renderer-Ready Line Layer*

Description

Build a normalized line layer for downstream adapters.

Usage

```
ggwebgl_layer_lines(
  data,
  x,
  y,
  z = NULL,
  group = NULL,
  colour = NULL,
  rgba = NULL,
  alpha = NULL,
  width = NULL,
  age = NULL,
  frame = NULL,
  time = NULL,
  panel_id = 1L,
  geom = "adapter_lines"
)
```

Arguments

data	Optional data frame supplying columns referenced by other arguments.
x, y	Coordinate vectors or column names in data.
z	Optional z coordinate vector or column name for 3D scenes.
group	Optional path-group vector or column name. When omitted, all rows form one path.
colour	Optional colour vector or column name. Ignored when rgba is supplied.
rgba	Optional renderer-ready RGBA matrix/data frame with four columns, or vector of length n * 4, using values in [0, 1] or [0, 255].
alpha	Optional alpha vector or column name used with colour.
width	Optional line-width vector or column name in renderer pixels.
age	Optional normalized age vector or column name in [0, 1].
frame, time	Optional timeline frame or time vector or column name.
panel_id	Scalar panel identifier for this layer.
geom	Debug geom name recorded in the payload.

Value

A normalized line layer list.

Examples

```
lines <- data.frame(
  x = c(0, 1, 2, 0, 1, 2),
  y = c(0, 1, 0, 1, 2, 1),
  group = c("a", "a", "a", "b", "b", "b")
)

ggwebgl_layer_lines(
  lines,
  x = "x",
  y = "y",
  group = "group",
  colour = "#334155",
  alpha = 0.75,
  width = 2
)
```

ggwebgl_layer_mesh *Renderer-Ready Mesh Layer*

Description

Build an indexed triangle mesh layer for downstream adapters. Triangle indices are supplied as one-based R indices and normalized to zero-based WebGL indices in the returned payload.

Usage

```
ggwebgl_layer_mesh(
  vertices,
  x = NULL,
  y = NULL,
  z = NULL,
  triangles = NULL,
  i = NULL,
  j = NULL,
  k = NULL,
  colour = NULL,
  rgba = NULL,
  alpha = NULL,
  id = NULL,
  scalar = NULL,
  normals = NULL,
  material = ggwebgl_material(),
  shading = NULL,
  pick_id = NULL,
  panel_id = 1L,
  geom = "adapter_mesh",
  wireframe = NULL
)
```

Arguments

vertices	Data frame, ggwebgl_mesh object, or list accepted by as_mesh_webgl().
x, y	Coordinate vectors or column names in data.
z	Optional z coordinate vector or column name. Defaults to zero.
triangles	Optional data frame supplying triangle index columns.
i, j, k	One-based triangle index vectors or column names.
colour	Optional colour vector or column name. Ignored when rgba is supplied.
rgba	Optional renderer-ready RGBA matrix/data frame with four columns, or vector of length n * 4, using values in [0, 1] or [0, 255].
alpha	Optional alpha vector or column name used with colour.
id	Optional stable primitive id vector or column name for selection.

scalar	Optional per-vertex scalar vector or column name.
normals	Optional vertex-normal matrix/data frame/vector or "auto".
material	Mesh material created by ggwebgl_material().
shading	Optional mesh shader mode overriding material$shading.
pick_id	Optional face picking ids. Length must be one or the number of triangles.
panel_id	Scalar panel identifier for this layer.
geom	Debug geom name recorded in the payload.
wireframe	Legacy shortcut for material$wireframe.

Value

A normalized mesh layer list.

Examples

```
vertices <- data.frame(x = c(0, 1, 0), y = c(0, 0, 1), z = c(0, 0, 0))
triangles <- data.frame(i = 1L, j = 2L, k = 3L)
ggwebgl_layer_mesh(vertices, x = "x", y = "y", z = "z", triangles = triangles)
```

---

ggwebgl_layer_points *Renderer-Ready Point Layer*

Description

Build a normalized point layer for downstream adapters. Inputs must already represent renderer coordinates and styling; package-specific semantics should be resolved before calling this helper.

Usage

```
ggwebgl_layer_points(
  data,
  x,
  y,
  z = NULL,
  colour = NULL,
  rgba = NULL,
  alpha = NULL,
  size = NULL,
  age = NULL,
  label = NULL,
  id = NULL,
  frame = NULL,
  time = NULL,
  panel_id = 1L,
  geom = "adapter_points"
)
```

Arguments

data	Optional data frame supplying columns referenced by other arguments.
x, y	Coordinate vectors or column names in data.
z	Optional z coordinate vector or column name for 3D scenes.
colour	Optional colour vector or column name. Ignored when rgba is supplied.
rgba	Optional renderer-ready RGBA matrix/data frame with four columns, or vector of length n * 4, using values in [0, 1] or [0, 255].
alpha	Optional alpha vector or column name used with colour.
size	Optional point-size vector or column name in renderer pixels.
age	Optional normalized age vector or column name in [0, 1].
label	Optional hover label vector or column name.
id	Optional stable primitive id vector or column name for selection.
frame, time	Optional timeline frame or time vector or column name.
panel_id	Scalar panel identifier for this layer.
geom	Debug geom name recorded in the payload.

Value

A normalized point layer list.

Examples

```
points <- data.frame(
  x = c(0, 1, 2),
  y = c(2, 1, 0),
  colour = c("#0f766e", "#f97316", "#2563eb"),
  label = c("a", "b", "c")
)

ggwebgl_layer_points(
  points,
  x = "x",
  y = "y",
  colour = "colour",
  alpha = 0.6,
  size = 3,
  label = "label"
)
```

ggwebgl_layer_raster *Renderer-Ready Raster Layer*

Description

Build a normalized raster layer from RGBA byte payloads.

Usage

```
ggwebgl_layer_raster(
  rgba,
  width,
  height,
  xmin,
  xmax,
  ymin,
  ymax,
  interpolate = FALSE,
  panel_id = 1L,
  geom = "adapter_raster"
)
```

Arguments

rgba	Integer or numeric vector of length width * height * 4, using byte values in [0, 255].
width, height	Raster dimensions in cells.
xmin, xmax, ymin, ymax	Raster extent.
interpolate	Whether the WebGL texture should use linear filtering.
panel_id	Scalar panel identifier for this layer.
geom	Debug geom name recorded in the payload.

Value

A normalized raster layer list.

Examples

```
ggwebgl_layer_raster(
  rgba = rep(c(15L, 23L, 42L, 255L), 4L),
  width = 2L,
  height = 2L,
  xmin = 0,
  xmax = 1,
  ymin = 0,
  ymax = 1,
```

```
  interpolate = TRUE
)
```

---

ggwebgl_layer_surface *Renderer-Ready Structured Surface Layer*

Description

Build a first-class structured-grid surface layer from a numeric matrix or `surface_matrix()` object.

Usage

```
ggwebgl_layer_surface(
  z,
  x = NULL,
  y = NULL,
  colour = NULL,
  rgba = NULL,
  alpha = NULL,
  palette = "Terrain 2",
  shading = c("surface_lambert", "surface_flat", "surface_height_colormap",
    "surface_uncertainty_alpha"),
  normals = "auto",
  material = NULL,
  uncertainty = NULL,
  pick_id = NULL,
  panel_id = 1L,
  geom = "adapter_surface",
  wireframe = FALSE,
  contours = FALSE,
  contour_levels = NULL,
  contour_colour = "#1f2937",
  contour_width = 1
)
```

Arguments

`z`	Numeric matrix or `ggwebgl_surface_matrix` object.
`x, y`	Optional coordinate vectors for matrix input.
`colour`	Optional colour vector. Ignored when `rgba` is supplied.
`rgba`	Optional renderer-ready RGBA matrix/data frame with four columns, or vector of length `vertex_count * 4`, using values in `[0, 1]` or `[0, 255]`.
`alpha`	Optional alpha vector used with `colour`.
`palette`	HCL palette used when neither `colour` nor `rgba` is supplied.
`shading`	Surface shader mode.

normals	Normal-generation mode. "auto" computes vertex normals.
material	Surface material created by ggwebgl_material().
uncertainty	Optional per-vertex uncertainty values in [0, 1].
pick_id	Optional triangle picking ids.
panel_id	Scalar panel identifier.
geom	Debug geom name recorded in the payload.
wireframe	Whether to request a wireframe overlay.
contours	Whether to generate contour-line overlays on the R side.
contour_levels	Optional numeric contour levels.
contour_colour	Contour line colour.
contour_width	Contour line width in renderer pixels.

Value

A normalized structured surface layer list.

Examples

```
ggwebgl_layer_surface(volcano[1:4, 1:4])
```

---

ggwebgl_layer_vectors *Renderer-Ready Vector Arrow Layer*

Description

Build a vector-arrow layer for downstream adapters.

Usage

```
ggwebgl_layer_vectors(
  data,
  x,
  y,
  xend,
  yend,
  z = NULL,
  zend = NULL,
  colour = NULL,
  rgba = NULL,
  alpha = NULL,
  width = NULL,
  head_size = NULL,
  id = NULL,
  frame = NULL,
```

```
  time = NULL,
  panel_id = 1L,
  geom = "adapter_vectors"
)
```

Arguments

data	Optional data frame supplying columns referenced by other arguments.
x, y	Coordinate vectors or column names in data.
xend, yend	Arrow endpoint coordinates.
z	Optional z coordinate vector or column name for 3D scenes.
zend	Optional arrow endpoint z coordinate for 3D scenes. When z or zend is omitted it defaults to zero in 3D projection.
colour	Optional colour vector or column name. Ignored when rgba is supplied.
rgba	Optional renderer-ready RGBA matrix/data frame with four columns, or vector of length n * 4, using values in [0, 1] or [0, 255].
alpha	Optional alpha vector or column name used with colour.
width	Optional shaft width in renderer pixels.
head_size	Optional arrowhead size in renderer pixels.
id	Optional stable primitive id vector or column name for selection.
frame, time	Optional timeline frame or time vector or column name.
panel_id	Scalar panel identifier for this layer.
geom	Debug geom name recorded in the payload.

Value

A normalized vector layer list.

Examples

```
arrows <- data.frame(x = 0:1, y = 0:1, xend = c(0.5, 1.4), yend = c(0.2, 1.2))
ggwebgl_layer_vectors(arrows, x = "x", y = "y", xend = "xend", yend = "yend")
```

---

ggwebgl_magnify_region

Build a Linked Magnifying-Glass Zoom Scene

Description

Create a deterministic zoom view from a rectangular data region. The helper is renderer-generic: callers provide renderer-ready ggWebGL sources and the selected region, and ggWebGL derives either a two-panel zoom spec or a publication figure with a linked inset.

Usage

```
ggwebgl_magnify_region(
  source,
  region,
  display = c("panel", "inset"),
  source_panel = NULL,
  zoom_layers = NULL,
  global_panel_id = "global",
  zoom_panel_id = "local",
  global_label = "Global",
  zoom_label = "Zoomed region",
  box = TRUE,
  box_colour = "#334155",
  box_alpha = 0.65,
  box_width = 1.5,
  inset = list(left = 0.68, top = 0.06, width = 0.24, height = 0.24),
  interactive = FALSE,
  width = NULL,
  height = NULL,
  background = "white",
  preset = c("clean", "publication"),
  labels = NULL,
  webgl = NULL
)
```

Arguments

source	A ggplot, ggWebGL widget, ggwebgl_spec, or raw renderer payload accepted by ggWebGL().
region	Rectangle to magnify. Use either list(x = c(xmin, xmax), y = c(ymin, ymax)) or list(xmin = ..., xmax = ..., ymin = ..., ymax = ...).
display	One of "panel" or "inset".
source_panel	Optional panel id to magnify when source has multiple panels. Defaults to the first panel.
zoom_layers	Optional renderer-ready layers to use in the zoom view. When omitted, the source panel layers are reused.
global_panel_id, zoom_panel_id	Panel ids used in display = "panel".
global_label, zoom_label	Optional panel labels.
box	Whether to add a rectangle overlay to the global panel.
box_colour, box_alpha, box_width	Rectangle styling.
inset	Inset placement list for display = "inset" with fractional left, top, width, and height.

interactive	Whether a two-panel magnifier should let browser-side brush rectangles on the global panel update the zoom panel viewport live.
width, height	Optional publication figure dimensions for inset output.
background, preset	Publication figure styling for inset output.
labels	Optional labels for the derived renderer specs.
webgl	Optional renderer options for the derived specs. Defaults to the source webgl options.

Value

A ggwebgl_spec for display = "panel" or a ggwebgl_publication_figure for display = "inset".

Examples

```
source <- ggwebgl_spec(
  layers = list(
    ggwebgl_layer_points(
      data.frame(x = c(0, 1, 2, 3), y = c(0, 2, 1, 3)),
      x = "x",
      y = "y",
      colour = "#2563eb",
      alpha = 0.75,
      size = 4
    )
  )
)

ggwebgl_magnify_region(
  source,
  region = list(x = c(0.75, 2.25), y = c(0.75, 2.25)),
  display = "panel"
)
```

---

ggwebgl_material *Define ggWebGL Mesh Material*

Description

Build a renderer material specification for mesh and surface layers.

Usage

```
ggwebgl_material(
 shading = c("flat", "lambert", "mesh_flat", "mesh_lambert", "mesh_phong_simple",
    "mesh_scalar_colormap", "mesh_selection_highlight", "surface_flat",
    "surface_lambert", "surface_height_colormap", "surface_uncertainty_alpha"),
  ambient = 0.35,
```

```
  diffuse = 0.75,
  specular = 0,
  light_dir = c(0.35, 0.45, 0.82),
  wireframe = FALSE,
  cull = c("back", "none")
)
```

Arguments

shading	Shading model. "flat" and "lambert" are the stable material aliases; mesh shader aliases such as "mesh_lambert" and "mesh_scalar_colormap" and surface shader aliases such as "surface_lambert" and "surface_height_colormap" are accepted by their respective layer families.
ambient, diffuse, specular	Lighting coefficients.
light_dir	Directional light vector.
wireframe	Whether to request a wireframe overlay.
cull	Face-culling mode, "back" or "none".

Value

A ggwebgl_material list.

Examples

```
ggwebgl_material(shading = "lambert", wireframe = TRUE)
```

---

ggwebgl_mesh *Build a ggWebGL Mesh Helper Object*

Description

Build a lightweight unstructured mesh object from explicit vertex and triangle tables.

Usage

```
ggwebgl_mesh(
  vertices,
  triangles,
  x = "x",
  y = "y",
  z = "z",
  i = "i",
  j = "j",
  k = "k",
  scalar = NULL,
```

```
  id = NULL,
  normals = NULL,
  colour = NULL,
  rgba = NULL,
  alpha = NULL,
  pick_id = NULL
)
```

Arguments

vertices	Data frame with vertex coordinates.
triangles	Data frame with one-based triangle indices.
x, y, z	Vertex coordinate column names.
i, j, k	Triangle index column names.
scalar	Optional scalar column name or vector for per-vertex scalar colouring.
id	Optional vertex id column name or vector.
normals	Optional normal matrix/data frame/vector or column triplet named nx, ny, nz in vertices.
colour, rgba, alpha	Optional vertex colour inputs passed through to ggwebgl_layer_mesh().
pick_id	Optional face picking ids.

Value

A ggwebgl_mesh object.

Examples

```
vertices <- data.frame(x = c(0, 1, 0), y = c(0, 0, 1), z = c(0, 0, 0))
triangles <- data.frame(i = 1L, j = 2L, k = 3L)
ggwebgl_mesh(vertices, triangles)
```

---

ggwebgl_publication_figure

Build a Publication-Mode Figure Container from ggWebGL Panels

---

Description

Create a package-owned HTML container for publication capture. Each child panel is rendered through ggWebGL in publication mode unless it already declares a different rendering contract explicitly.

Usage

```
ggwebgl_publication_figure(
  panels,
  layout = c("single", "row", "grid"),
  labels = NULL,
  annotations = NULL,
  inset = NULL,
  background = "white",
  preset = c("clean", "publication"),
  width = NULL,
  height = NULL
)
```

Arguments

panels	A non-empty list of panel sources. Supported sources are ggplot objects, ggWebGL htmlwidgets, ggwebgl_spec objects, or raw renderer payloads accepted by ggWebGL(). Each element may also be a list with source plus optional show_panel_overlay.
layout	One of "single", "row", or "grid".
labels	Optional character vector of panel labels.
annotations	Optional list of figure-level text annotations. Each entry should contain text, x, and y, with optional size, colour, font, hjust, and vjust.
inset	Optional inset specification containing a panel source plus fractional left, top, width, and height.
background	Figure background colour.
preset	Publication styling preset. "publication" adds subtle panel borders and muted overlay text.
width, height	Optional figure dimensions in pixels.

Value

A browsable HTML container with class ggwebgl_publication_figure.

Examples

```
demo_spec <- ggwebgl_spec(
  layers = list(
    ggwebgl_layer_points(
      data.frame(x = c(0.15, 0.52, 0.84), y = c(0.20, 0.78, 0.42)),
      x = "x",
      y = "y",
      colour = c("#0f766e", "#f97316", "#2563eb"),
      alpha = 0.8,
      size = 5
    )
  )
)
```

```
figure <- ggwebgl_publication_figure(
  panels = list(demo_spec),
  width = 420,
  height = 280
)

inherits(figure, "ggwebgl_publication_figure")
```

---

ggwebgl_selection *Define ggWebGL Selection Behavior*

Description

Build a structured renderer-owned selection specification.

Usage

```
ggwebgl_selection(
  mode = c("none", "brush", "lasso", "brush_lasso"),
  highlight = TRUE,
  emit = TRUE
)
```

Arguments

mode	Selection mode: "none", "brush", "lasso", or "brush_lasso".
highlight	Whether selected primitives should be visibly highlighted.
emit	Whether selection payloads should be emitted to Shiny/callbacks.

Value

A ggwebgl_selection list.

Examples

```
ggwebgl_selection("brush_lasso")
```

ggwebgl_spec *Build a ggWebGL Specification from Renderer-Ready Layers*

Description

Build a ggWebGL Specification from Renderer-Ready Layers

Usage

```
ggwebgl_spec(
  layers,
  labels = list(),
  webgl = list(),
  grid = NULL,
  panels = NULL,
  messages = character(),
  timeline = NULL
)
```

Arguments

layers	A list of normalized point, line, raster, vector, ribbon, mesh, or surface layers.
labels	Optional labels list (title, subtitle, x, y).
webgl	Optional renderer options passed to theme_webgl().
grid	Optional list with rows and cols.
panels	Optional panel metadata list or data frame with panel_id, row, col, optional label, and optional viewport.
messages	Optional character vector of renderer messages.
timeline	Optional ggwebgl_timeline() specification.

Value

A classed ggwebgl_spec object accepted by ggWebGL().

Examples

```
panel_points <- ggwebgl_layer_points(
  data.frame(x = c(0, 1), y = c(1, 0)),
  x = "x",
  y = "y"
)
panel_lines <- ggwebgl_layer_lines(
  data.frame(x = c(0, 1, 2), y = c(0, 1, 0)),
  x = "x",
  y = "y",
  panel_id = "B"
```

```
)

spec <- ggwebgl_spec(
  layers = list(panel_points, panel_lines),
  labels = list(title = "adapter spec"),
  panels = data.frame(
    panel_id = c(1L, "B"),
    row = c(1L, 1L),
    col = c(1L, 2L),
    stringsAsFactors = FALSE
  )
)

spec$render$grid
```

---

ggwebgl_timeline *ggWebGL Timeline Controls*

Description

Build a lightweight runtime timeline specification for animated ggWebGL scenes. Layers can opt into timeline filtering with `frame` or `time` fields.

Usage

```
ggwebgl_timeline(
  frames = NULL,
  time = NULL,
  duration = NULL,
  loop = TRUE,
  autoplay = FALSE,
  speed = 1,
  controls = TRUE,
  filter = c("exact", "cumulative"),
  values = NULL,
  source = c("auto", "frame", "time"),
  mode = NULL,
  fps = NULL
)
```

Arguments

`frames`	Optional integer frame values.
`time`	Optional numeric time values.
`duration`	Optional playback duration in seconds.
`loop`	Whether playback should loop.
`autoplay`	Whether playback should start automatically.

speed	Playback speed multiplier.
controls	Whether the widget should show timeline controls.
filter	Timeline visibility mode. "exact" shows only samples matching the current frame or time. "cumulative" keeps samples up to the current frame or time.
values	Optional frame or time values. Use source to choose whether they populate the frame or time axis.
source	Timeline value source for values. "auto" uses frame values unless time is supplied.
mode	Optional alias for filter.
fps	Optional frames-per-second metadata for downstream controls.

Value

A ggwebgl_timeline list.

Examples

```
ggwebgl_timeline(frames = 1:4, autoplay = FALSE)
```

---

ggwebgl_transport *Configure ggWebGL Transport Options*

Description

ggwebgl_transport() controls how large renderer payloads are encoded before they are handed to the browser. Small scenes keep the existing JSON arrays by default; large point layers can use base64 typed arrays, palette color lookup, deterministic preview LOD, and progressive GPU upload.

Usage

```
ggwebgl_transport(
  mode = c("auto", "compact", "legacy"),
  threshold = 100000L,
  progressive = c("auto", "on", "off"),
  chunk_size = 100000L,
  position = c("float32", "quantized"),
  colors = c("auto", "palette", "rgba"),
  lod = c("auto", "grid", "none"),
  lod_max_points = 5000L
)
```

Arguments

mode	Transport mode. "auto" compacts point layers at or above threshold, "compact" compacts all point layers, and "legacy" keeps the original JSON-array payloads.
threshold	Row threshold used when mode = "auto".
progressive	Progressive upload policy. "auto" enables progressive upload for compact layers, "on" forces it, and "off" disables it.
chunk_size	Number of points per progressive upload chunk.
position	Position encoding, either "float32" or "quantized".
colors	Color encoding. "auto" uses palette lookup when beneficial and byte RGBA otherwise; "palette" requires palette lookup when possible; and "rgba" stores byte RGBA.
lod	LOD preview policy. "auto" generates deterministic grid previews for compact point layers, "grid" forces them, and "none" disables them.
lod_max_points	Maximum number of preview points generated by the grid LOD helper.

Value

A normalized ggwebgl_transport list.

Examples

```
ggwebgl_transport(threshold = 100000L, chunk_size = 50000L)
```

---

ggwebgl_view *Define a ggWebGL View Contract*

Description

Build a structured renderer view specification. This replaces the previous loose dimension, camera, projection, and camera_state fields while keeping them mirrored internally for older renderer paths.

Usage

```
ggwebgl_view(
  dimension = c("2d", "3d"),
  projection = c("orthographic", "perspective"),
  controller = NULL,
  state = list()
)
```

Arguments

dimension	Renderer dimensionality, "2d" or "3d".
projection	Projection mode, "orthographic" or "perspective".
controller	Interaction controller. Use "panzoom" for 2D scenes and "orbit" or "trackball" for 3D scenes.
state	Camera/view state list. Recognized fields include target, distance, rotation, up, fov, near, and far. Legacy yaw and pitch are converted to rotation.

Value

A ggwebgl_view list.

Examples

```
ggwebgl_view(dimension = "3d", controller = "trackball")
```

---

ggWebGLOutput *Shiny Output Binding for ggWebGL*

Description

Shiny Output Binding for ggWebGL

Usage

```
ggWebGLOutput(outputId, width = "100%", height = "480px")
```

Arguments

outputId	Output variable to read from.
width, height	Widget dimensions.

Value

A Shiny output tag.

Examples

```
ui <- shiny::fluidPage(
  ggWebGLOutput("plot", height = "320px")
)

ui
```

renderGgWebGL *Render a ggWebGL Widget in Shiny*

Description

Render a ggWebGL Widget in Shiny

Usage

```
renderGgWebGL(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

expr	An expression returning a ggWebGL widget.
env	Evaluation environment.
quoted	Whether expr is quoted.

Value

A Shiny render function.

Examples

```
server <- function(input, output, session) {
  output$plot <- renderGgWebGL({
    ggplot_webgl(
      ggplot2::ggplot(
        mtcars[1:8, ],
        ggplot2::aes(mpg, wt, colour = factor(cyl))
      ) +
        geom_point_webgl(size = 2) +
        theme_webgl()
    )
  })
}

server
```

scale_time_webgl *Add Timeline Metadata to a ggplot*

Description

`scale_time_webgl()` is not a visual ggplot2 scale. It records timeline metadata for `ggplot_webgl()` so frame or time values can be normalized into `render$timeline`.

Usage

```
scale_time_webgl(
  source = c("auto", "frame", "time"),
  values = NULL,
  mode = c("exact", "cumulative"),
  fps = NULL,
  speed = 1,
  loop = FALSE,
  label = NULL,
  format = NULL
)
```

Arguments

`source`	Timeline column source. `"auto"` prefers a built `time` column when present and otherwise uses `frame`.
`values`	Optional explicit timeline values. When omitted, values are derived from built layer `time` or `frame` columns.
`mode`	Timeline filtering intent: `"exact"` or `"cumulative"`.
`fps`	Optional frames-per-second metadata.
`speed`	Positive playback-speed multiplier.
`loop`	Whether future playback controls should loop.
`label`	Optional timeline label metadata.
`format`	Optional display-format metadata.

Value

An object that can be added to a `ggplot`.

Examples

```
df <- data.frame(x = c(0, 1, 2), y = c(0, 1, 0), frame = 1:3)
plot <- ggplot2::ggplot(df, ggplot2::aes(x, y, frame = frame)) +
  geom_point_webgl() +
  scale_time_webgl(source = "frame", mode = "exact")
plot$ggwebgl$time_scale$source
```

snapshot_ggwebgl *Capture a ggWebGL Scene as a Static Image*

Description

Build a ggWebGL widget if needed, hide interactive chrome for export, and capture a clean static image through the browser-backed widget path.

Usage

```
snapshot_ggwebgl(
  x,
  file,
  width = 1800L,
  height = 1200L,
  format = NULL,
  dpi = 300,
  background = "white",
  preset = c("clean", "publication"),
  selfcontained = FALSE,
  wait_seconds = 3,
  show_panel_overlay = FALSE,
  elementId = NULL
)
```

Arguments

x	A ggplot, ggWebGL htmlwidget, ggwebgl_spec, raw renderer payload accepted by ggWebGL(), or a ggwebgl_publication_figure().
file	Output file path.
width, height	Output size in pixels.
format	Optional image format. When omitted, it is inferred from file.
dpi	Output density metadata used when writing the image.
background	Background colour used for the final flattened image.
preset	Export preset. "clean" removes UI chrome; "publication" also applies subtle panel-strip and panel-frame styling for publication capture.
selfcontained	Passed through to htmlwidgets::saveWidget() for the temporary export widget.
wait_seconds	Delay before capture to allow the widget to finish rendering.
show_panel_overlay	Whether facet/panel overlays should remain visible in the captured output.
elementId	Optional DOM element id passed when x must first be turned into a widget.

Value

The normalized output file path, invisibly.

Examples

```
old <- options(ggwebgl.reset_processx_supervisor = TRUE)
on.exit(options(old), add = TRUE)

tiny_spec <- ggwebgl_spec(
  layers = list(
    ggwebgl_layer_points(
      data.frame(x = c(0.15, 0.5, 0.82), y = c(0.25, 0.78, 0.4)),
      x = "x",
      y = "y",
      colour = c("#0f766e", "#f97316", "#2563eb"),
      alpha = 0.8,
      size = 5
    )
  ),
  webgl = list(shader = "default", interactions = character())
)

out <- tempfile(fileext = ".jpg")
snapshot_ggwebgl(
  tiny_spec,
  out,
  width = 320,
  height = 220,
  format = "jpeg",
  preset = "clean",
  wait_seconds = 0.25
)
file.exists(out)
```

---

stat_surface_webgl *WebGL Structured Grid Surface Stat*

Description

Validate and order regular `(x, y, z)` triples for `geom_surface_webgl()`. Missing or duplicate cells are rejected because structured surfaces are triangulated without interpolation.

Usage

```
stat_surface_webgl(
  mapping = NULL,
  data = NULL,
  geom = GeomSurfaceWebGL,
```

```
  position = "identity",
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
geom	The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following: • A Geom ggproto subclass, for example GeomPoint. • A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point". • For more information and other ways to specify the geom, see the layer geom documentation.
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.

`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 `Layer` ready for `ggplot2`.

Examples

```
surface <- expand.grid(x = 1:3, y = 1:3)
surface$z <- with(surface, x * y)
ggplot2::ggplot(surface, ggplot2::aes(x, y, z = z)) +
  stat_surface_webgl()
```

---

surface_matrix *Structured Surface Matrix*

Description

Create a structured surface-grid object from a numeric matrix. Rows map to `y`, columns map to `x`, and the matrix values map to `z`.

Usage

```
surface_matrix(z, x = NULL, y = NULL)
```

Arguments

z	Numeric matrix of height values.
x	Optional x-coordinate vector with one value per matrix column.
y	Optional y-coordinate vector with one value per matrix row.

Value

A ggwebgl_surface_matrix object accepted by ggwebgl_layer_surface().

Examples

```
surface_matrix(volcano[1:3, 1:3])
```

---

theme_webgl *Add WebGL Rendering Options to a ggplot*

Description

Attach WebGL-specific rendering metadata to a ggplot object. The returned object is consumed by ggplot_webgl() and stored on the plot as plot$ggwebgl.

Usage

```
theme_webgl(
  shader = "default",
  antialias = TRUE,
  transparent = TRUE,
  buffer_size = 65536L,
  interactions = c("pan", "zoom"),
  interactions_spec = NULL,
  rendering = "visualization",
  panel_overlay = "auto",
  view = NULL,
  selection = NULL,
  dimension = "2d",
  camera = "orbit",
  projection = "orthographic",
  camera_state = list(),
  depth_test = NULL,
  blend_mode = "auto",
  timeline = NULL,
  transport = NULL,
  ...
)
```

Arguments

shader	Shader preset name or path identifier. Built-in modes include point shaders ("default", "density_splat", "uncertainty_alpha", "point_sprite_glow"), trajectory shaders ("trajectory_age", "trajectory_age_glow", "trajectory_velocity", "trajectory_direction"), and raster shaders ("raster_texture", "raster_threshold", "raster_contour_overlay"). Mesh and surface layers usually select their shader through material/shading arguments.
antialias	Logical scalar; whether antialiasing should be requested.
transparent	Logical scalar; whether the drawing surface should allow transparency.
buffer_size	Integer scalar giving the initial buffer allocation used by the eventual renderer.
interactions	Legacy character vector of interaction modes to enable. New code should use selection = ggwebgl_selection(...) for brush/lasso behavior.
interactions_spec	Optional ggwebgl_interactions() object. This is the preferred structured interaction contract for hover, click, brush, lasso, camera, and Shiny event behavior.
rendering	Rendering contract mode. "visualization" keeps the current interactive widget chrome. "publication" suppresses presentation chrome by default and is intended for clean figure capture.
panel_overlay	Panel overlay display mode. "auto" shows panel strips and frames for faceted plots, "show" forces them on, and "hide" removes them.
view	Optional ggwebgl_view() object. This is the preferred structured view/camera contract.
selection	Optional ggwebgl_selection() object. This is the preferred selection contract.
dimension, camera, projection, camera_state	Legacy view fields retained as an internal migration shim.
depth_test	Logical scalar. NULL uses the renderer default: disabled for 2D scenes and enabled for 3D scenes. Set explicitly to override.
blend_mode	Primitive blending mode: "auto", "alpha", "additive", or "premultiplied".
timeline	Optional ggwebgl_timeline() specification for runtime playback controls.
transport	Optional ggwebgl_transport() object controlling compact typed-array payloads, LOD previews, and progressive upload for large point layers.
...	Reserved for future backend-specific options.

Value

An object that can be added to a ggplot.

Examples

```
plot <- ggplot2::ggplot(mtcars, ggplot2::aes(mpg, wt, colour = factor(cyl))) +
  ggplot2::geom_point() +
  theme_webgl(
    shader = "density_splat",
```

```
    selection = ggwebgl_selection("none")
  )

plot$ggwebgl
```

`updateGgWebGLTimeline` *Update a ggWebGL Timeline from Shiny*

Description

Send a timeline update message to a rendered ggWebGL widget. The browser applies the update to the widget identified by `outputId` and emits the updated state back as `input$<outputId>_timeline`.

Usage

```
updateGgWebGLTimeline(
  session,
  outputId,
  value = NULL,
  index = NULL,
  playing = NULL,
  speed = NULL,
  loop = NULL
)
```

Arguments

`session`	A Shiny session object.
`outputId`	Output id passed to `ggWebGLOutput()`. Inside Shiny modules, pass the unnamespaced id; `session$ns()` is applied by this helper.
`value`	Optional frame or time value to select.
`index`	Optional 1-based timeline index to select. Exactly one of `value` or `index` may be supplied.
`playing`	Optional logical scalar controlling playback.
`speed`	Optional positive playback-speed multiplier.
`loop`	Optional logical scalar controlling loop playback.

Value

`NULL`, invisibly.

Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  server <- function(input, output, session) {
    shiny::observeEvent(input$next_frame, {
      updateGgWebGLTimeline(session, "plot", index = 2)
    })
  }
}
```

---

webgl_spec *Build ggWebGL Renderer Options*

Description

webgl_spec() creates a normalized renderer option list for ggplot_webgl(), ggwebgl_spec(), and downstream adapter code. It is a compact user-facing wrapper around the same internal normalization used by theme_webgl().

Usage

```
webgl_spec(
  camera = c("panzoom", "orbit", "trackball"),
  projection = c("orthographic", "perspective"),
  dimension = NULL,
  depth_test = NULL,
  blend_mode = c("auto", "alpha", "additive", "premultiplied"),
  shader = NULL,
  interactions = NULL,
  view = NULL,
  selection = NULL,
  timeline = NULL,
  transport = NULL,
  ...
)
```

Arguments

camera	Camera/controller mode. "panzoom" targets 2D scenes; "orbit" and "trackball" target 3D scenes.
projection	Projection mode, "orthographic" or "perspective".
dimension	Optional renderer dimensionality. When omitted, "orbit" and "trackball" imply "3d" while "panzoom" implies "2d".
depth_test	Logical scalar. NULL enables depth testing for 3D scenes and disables it for 2D scenes.
blend_mode	Primitive blending mode: "auto", "alpha", "additive", or "premultiplied".
shader	Optional shader preset.

interactions	Optional interaction mode vector.
view	Optional ggwebgl_view() object. If supplied, it takes precedence over camera, projection, and dimension.
selection	Optional ggwebgl_selection() object.
timeline	Optional ggwebgl_timeline() object.
transport	Optional ggwebgl_transport() object.
...	Additional renderer options stored under webgl$extra.

Value

A normalized renderer option list.

Examples

```
webgl_spec(camera = "orbit", projection = "perspective")
```

Index