Package 'ggWebGL'

May 4, 2026

Title Browser-Native 'WebGL' Rendering for R Graphics

Date 2026-04-29

Version 0.4.0

Author Frederic Bertrand [cre, aut] (ORCID:
<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

RoxygenNote 7.3.3

Depends R (>= 4.1)

Imports ggplot2, htmltools, htmlwidgets, rlang

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

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

NeedsCompilation no

Repository CRAN

Date/Publication 2026-05-04 12:10:08 UTC

Contents

`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
```

---

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)
```

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_mesh_webgl *WebGL Mesh Layer*

Description

Add a 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",
  ...,
  wireframe = FALSE,
  material = ggwebgl_material(wireframe = wireframe),
  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.

`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_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_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_surface_webgl *WebGL Triangulated Surface Layer*

Description

Add a regular-grid surface layer tagged for the ggWebGL mesh renderer. The built grid is triangulated before being sent to WebGL.

Usage

```
geom_surface_webgl(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  wireframe = FALSE,
  material = ggwebgl_material(shading = "lambert", wireframe = wireframe),
```

```
  normals = "auto",
  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.

`wireframe` Whether to request a wireframe overlay.

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

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

`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

```
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_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()
```

---

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)
```

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
```

`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_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,
  y,
  z = NULL,
  triangles = NULL,
  i = NULL,
  j = NULL,
  k = NULL,
  colour = NULL,
  rgba = NULL,
  alpha = NULL,
  id = NULL,
  normals = NULL,
  material = ggwebgl_material(),
  pick_id = NULL,
  panel_id = 1L,
```

```
  geom = "adapter_mesh",
  wireframe = NULL
)
```

Arguments

vertices	Data frame supplying vertex coordinates.
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.
normals	Optional vertex-normal matrix/data frame/vector or "auto".
material	Mesh material created by ggwebgl_material().
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 Surface Layer*

Description

Build a triangulated surface from a regular z matrix.

Usage

```
ggwebgl_layer_surface(
  z,
  x = NULL,
  y = NULL,
  colour = NULL,
  rgba = NULL,
  alpha = NULL,
  palette = "Terrain 2",
  normals = "auto",
  material = ggwebgl_material(shading = "lambert"),
  pick_id = NULL,
  panel_id = 1L,
  geom = "adapter_surface",
  wireframe = NULL
)
```

Arguments

z	Numeric matrix of height values.
x, y	Optional coordinate vectors. Defaults to matrix column and row indices.
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.
palette	HCL palette name used when colour or rgba is omitted.
normals	Normal-generation mode. "auto" computes vertex normals.
material	Surface material created by ggwebgl_material().
pick_id	Optional face picking ids.
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

```
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"),
  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" or "lambert".
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_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, or mesh 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")
)
```

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.

Value

A ggwebgl_timeline list.

Examples

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

---

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")
```

---

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
```

---

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)
```

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"),
  rendering = "visualization",
  panel_overlay = "auto",
  view = NULL,
  selection = NULL,
  dimension = "2d",
  camera = "orbit",
  projection = "orthographic",
  camera_state = list(),
  timeline = NULL,
  ...
)
```

Arguments

shader	Shader preset name or path identifier. Built-in modes are "default", "density_splat", "trajectory_age", and "trajectory_age_glow".
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.
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.

`timeline`
: Optional `ggwebgl_timeline()` specification for runtime playback controls.

`...`
: 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
```

Index